Preferential Proximal Policy Optimization

Tamilselvan Balasuntharam
Ontario Tech University
Oshawa, ON, Canada
tamilselvan.balasuntharam@ontariotechu.net

Heidar Davoudi
Ontario Tech University
Oshawa, ON, Canada
heidar.davoudi@ontariotechu.ca

Mehran Ebrahimi
Ontario Tech University
Oshawa, ON, Canada
mehran.ebrahimi@ontariotechu.ca

2023 International Conference on Machine Learning and Applications (ICMLA) | 979-8-3503-4534-6/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICMLA58977.2023.00048

***Abstract*—The Proximal Policy Optimization (PPO) is a policy gradient approach providing state-of-the-art performance in many domains through the "surrogate" objective function using stochastic gradient ascent. While PPO is an appealing approach in reinforcement learning, it does not consider the importance of states (a frequently seen state in a successful trajectory) in policy/value function updates. In this work, we introduce Preferential Proximal Policy Optimization (P3O) which incorporates the importance of these states into parameter updates. First, we determine the importance of each state based on the variance of the action probabilities given a particular state multiplied by the value function, normalized and smoothed using the Exponentially Weighted Moving Average. Then, we incorporate the state's importance in the surrogate objective function. That is, we redefine value and advantage estimation objectives functions in the PPO approach. Unlike other related approaches, we select the importance of states automatically which can be used for any algorithm utilizing a value function. Empirical evaluations across six Atari environments demonstrate that our approach significantly outperforms the baseline (vanilla PPO) across different tested environments, highlighting the value of our proposed method in learning complex environments.**

I. Introduction

Many Reinforcement learning (RL) approaches rely on value functions to estimate the expected return from a given state, which in turns, guide agents in selecting appropriate actions. However, not all states possess equal significance. Recently, Anand and Precup [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 bootstrapping the target, while those with high preference witness more frequent updates and exert a greater influence on bootstrapping the target. They showed this state-dependent preference technique can enhance the effectiveness of the value function in reinforcement learning by selectively emphasizing high-value states and suppressing low-value states.

While the PTD approach provides a mechanism to incorporate state importance in Temporal Deference learning regime, it is explored and tested in simple environment such as cart-pole problem, where calculating beta states (i.e., $\beta(s)$) is based on knowledge of environment variables, such as cart pole position, velocity, and angle. For more complex environment (such as Atari 2600 games) with numerous different states and actions adapting approaches such as PPO leveraging state importance is very useful. Finally, PTD approaches define beta states manually based on the domain knowledge from the environment. This makes the applicability of the PTD approach limited to simple environments.

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 [2], we determine state importance SI for state s. Then, we smooth SI using Exponentially Weighted Moving Average [3] 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).

Subsequently, we modify the Generalized Advantage Estimation (GAE) algorithm [4] 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.

Lastly, we integrate the automatically computed $\beta(s)$ values and GBAE into the Proximal Policy Optimization (PPO) algorithm [5], creating the Preferential Proximal Policy Optimization (P3O) algorithm. In particular, P3O replaces traditional advantages with β-advantages and updates the value function according to $\beta(s)$ values. We compare P3O with PPO to demonstrate that our approach achieves greater performance in Atari environments [6], including MsPacman, Seaquest, Breakout, River Raid and Qbert. Our contributions are summarized as follow:

1) We incorporate state importance into the GAE algorithm based on modified objective function formulation tailored to our proposed 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.
4) We demonstrate the efficacy of proposed P3O in image-based environments, highlighting the versatility of the algorithm across diverse settings.

II. Background

We model our problem as a Markov decision process (MDP) with discrete action space A and state space S. The action space depends on the environment (such as Atari 2600 games

1946-0759/23/$31.00 ©2023 IEEE
DOI 10.1109/ICMLA58977.2023.00048

like Breakout, Seaquest, and Enduro). The agent interacts with the environment in state $s_t \in S$ and selects an action $a_t \in A$ based on its current policy, $\pi_\theta(a_t|s_t)$. Upon executing an action, the environment provides a reward r_{t+1} based on the current state s_t and action a_t with t denoting the current timestep. The primary objective of the agent is to update the network parameters θ in order to discover the optimal θ which gives the best of set of rewards for the policy and value functions.

Our proposed algorithm builds upon key components such as Preferential Temporal Difference Learning (PTD), Generalized Advantage Esimtation (GAE), and Proximal Policy Optimization (PPO). The following sections provide concise overviews of these essential elements.

A. Policy Gradient

Policy gradient algorithms, including Advantage Actor Critic (A2C) [7], PPO [5], and Phasic Policy Gradient (PPG) [8], 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 in its general form as follows:

$$\nabla_\theta J(\theta) = E[\sum_{t=0}^{\infty} \nabla_\theta log \pi_\theta(a_t|s_t)\ G_t]. \tag{1}$$

In this expression, G_t can be replaced with TD advantage $\delta_t = r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$ where V^π is the value function generated by the critic, Advantage function $A^\pi(s_t, a_t)$, GAE $\hat{A}_t^{GAE(\gamma,\lambda)}$, or the return G_t^β in PTD approach. The goal in policy gradient methods is to maximize the expected total reward by repeatedly estimating the gradient.

B. Preferential Temporal Difference Learning

Anand and Precup [1] employed a preference function β, to emphasize the importance of the state for the update as well as the target. The preference function is within the range of [0, 1] and if $\beta(s) = 0$, the value function will not be updated when the state is visited, and if $\beta(s) = 1$, the state will be fully updated. This allows *skipping* states that are partially observable and generate the information for fully observable states instead. There is a trade-off, as not updating a state may cause its value to be completely inaccurate and consequently create a biased update when used as a target for previous states. They [1] proposed beta returns G_t^β to avoid biased updates when $\beta(s) = 0$. They introduced a β return that bootstraps according to preference:

$$G_t^\beta = r_t + \gamma[\beta(s_{t+1})V(s_{t+1}) + (1 - \beta(s_{t+1}))G_{t+1}^\beta]. \tag{2}$$

Moreover, they created both the offline PTD algorithm and online PTD algorithm. This means that the PTD algorithm can be updated after every episode or set of timesteps using the offline version, or updated every timestep while the algorithm is interacting with the environment.

[1]

III. Methodology

To fully leverage the capabilities of our proposed algorithm, we incorporate three essential components. First, we compute the State Importance SI, which is passed through the Exponentially Weighted Moving Average (EWMA) function to smooth the SI values. We then apply Min-Max Normalization to normalize these SI values within the range [0, 1], which become our beta values $\beta(s)$. Next, we employ the Generalized Beta Advantage Estimation (GBAE) to generate beta advantages $\hat{A}_t^{GBAE}$. These beta advantages are used to calculate the policy loss for the P3O algorithm. We then compute the value loss by using regression on the mean-squared error with the generated 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.

A. Generalized Beta Advantage Estimation

We build GBAE upon GAE [4] by incorporating state preference function $\beta(s)$ into the advantage estimating function (3). 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 algorithm focuses on finding a balance between bias and variance in the advantage estimation by combining the properties of Temporal Difference (TD) learning and Monte Carlo (MC) methods [4]. In GAE, the parameter λ controls the balance between bias and variance, with $\lambda = 0$ adapting TD properties and $\lambda = 1$ demonstrating MC properties. However in GBAE, with $\lambda = 0$ the algorithm adapts TD properties and $\lambda = 1$ demonstrates β return with baseline properties (G_t^β - $V(s_t)$).

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

$$\hat{A}_t^{GBAE(\gamma,\lambda)} = \delta_t + (1 - \beta(s_{t+1}))(\gamma\lambda)A_{t+1}^{GBAE}, \tag{3}$$

where γ is the discount factor, typically set to 0.99, and λ is a horizon control variable which allows the model more focused on immediate results or distant future rewards with the constrained $0 \leq \lambda \leq 1$; β_{t+1} represents the preference function for the next time-step; δ 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

[1]Github Link: https://github.com/MegaTlash/P3O.

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. We found that this biasing mechanism enhances the performance of our algorithm by incorporating state preferences into the PPO policy optimization process.

B. Automatic Calculation of β function

The proposed algorithm is built upon the concept of state importance. We utilize $\beta(s)$ in the Preferential Temporal Difference (PTD) approach to assign different levels of significance to various states.

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. That is, states are considered important as they introduce a bias in the algorithm towards actions that lead to these extreme rewards. Conversely, states yielding comparable rewards, with a near-uniform distribution of action probabilities given a particular state, are deemed unimportant.

State Importance SI is calculated as the variance of the probability distribution $p(a_t|s_t)$ multiplied by the absolute of $V^{\pi}(s_t)$.

$$SI(s_t) = Var[p(a_t|s_t)] * |V^{\pi}(s_t)| \tag{4}$$

where $V^{\pi}(s)$ represents the value function from the critic. However, the evaluation of state importance is not a standalone process. In order to consider the impact of significant states on their neighboring states over time, we incorporate an Exponentially Weighted Moving Average (EWMA) smoothing technique. This approach guarantees that states preceding and following a crucial state receive an enhancement in their importance scores. As a result, the agent's learning is guided through a more effective sequence of updates.

The resulting SI values are then passed through the EWMA function [3],

$$x_t = \alpha * SI(s_t) + (1 - \alpha) * x_{t-1}. \tag{5}$$

Here, x_t is the smoothed value at time t, and α is the smoothing factor. α 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]. The normalized smoothed state importance measure eliminates the necessity of predefining $\beta(s)$ conditions, as suggested by the authors of Preferential Temporal Difference Learning [1]. Consequently, we directly integrate this measure into our proposed Generalized Beta Advantage Estimation (GBAE) framework, as defined in (3). By doing so, we enhance both the PPO policy and value functions, further augmenting the capabilities of our algorithm.

C. Preferential Proximal Policy Optimization

Preferential Proximal Policy Optimization (P3O) is a modified Proximal Policy Optimization algorithm [5] 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:

$$L_t^{CLIP}(\theta) = min(r_t(\theta)\hat{A}_t^{GBAE}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^{GBAE} \tag{6}$$

where θ 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 π_θ and the old policy $\pi_{\theta_{old}}$, $\hat{A}_t^{GBAE(\gamma,\lambda)}$ is GBAE, and ϵ is a hyperparameter 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 III-B . 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

$$L_t^{VF}(\theta) = \sum_{n=1}^{N} \beta(s_n)(V_\theta(s_n) - \hat{V}_h)^2. \tag{7}$$

$V_\theta(s_t)$ is the predicted value of state s_t by the value function parameterized by θ, 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 [5] to create the P3O objective function

$$L_t^{CLIP+VF+S}(\theta) = E_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]. \tag{8}$$

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 following pseudo-code for P3O is a modification of OpenAI's Spinning Up documentation [9].

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

Algorithm 1 Preferential Proximal Policy Optimization (P3O)

Input: Initial policy/value function parameters θ_0
Output: Final policy/value function parameters θ_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 τ_i
    Compute rewards from environment (r_t)
    Compute method for Automatically Calculating Beta States $\beta(s)$ (section III-B)
    Compute $\hat{A}_t^{GBAE}$ using GBAE with beta states $\beta(s_t)$ based on current value function Eqn. (3)
    Calculate policy loss using PPO-Clip objective and $\hat{A}_t^{GBAE}$ Eqn. (6)
    Calculate value function loss using mean-squared error Eqn. (7)
    Combine policy loss, value loss, and entropy bonus; update network Eqn. (8)
    Determine θ_{k+1} based on Eqn. (8) objective function
end for

complex environments and provides a more flexible approach to learning in situations with non-trivial observation spaces.

IV. Experiments

In this section, we present the results of our proposed Preferential Proximal Policy Optimization (P3O) algorithm and compare it with the baseline Proximal Policy Optimization (PPO) [5] across 6 different Atari 2600 environments. We evaluate the performance of both algorithms by comparing their average returns over the last 10 episodes (Fig. 1) and returns for each episode (Fig. 2) after training for 20 million timesteps. We also calculated the variance for each environment (Fig. 3) to showcase if each algorithms have similar or different variance.

A. 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 P3) algorithm was adapted from CleanRL [10]. Both models employed a learning rate of 1×10^{-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 γ and the trace decay parameter λ 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.

B. Atari 2600 Environments

We utilized a suite of six Atari 2600 environments [6] provided by OpenAI's Gym toolkit [11] to train and evaluate our algorithms. These challenging environments, requiring complex decision-making, serve as popular benchmarks for RL algorithms:

- **Breakout:** A classic arcade game (1976) where players use a paddle to break bricks with a bouncing ball. The agent receives points for breaking bricks.
- **Freeway:** Released in 1981, this game involves controlling a chicken to cross a busy freeway while dodging cars. The player receives points for each successful crossing.
- **Qbert:** A 1982 puzzle game where the character Qbert navigates a pyramid structure, avoiding enemies and changing cube colors. The agent receives by changing color of the cubes to their destination or defeating enimes. They also gain points for completing a level.
- **River Raid:** A scrolling shooter game (1982) in which agent control a helicopter, destroying enemies and collecting fuel over a river. The game awards points for destroying enemies and collecting fuel.
- **MsPacman:** Developed in 1982, agent navigate Ms. Pac-Man through a maze, eating dots and avoiding ghosts. The agent can also eat power pellets that allow Ms. Pac-Man to eat the ghosts for a limited time, gaining extra points.
- **Seaquest:** A 1983 game in which players control a submarine to rescue divers, avoid mines, and collect treasure. The agent is rewarded for rescuing divers, collecting treasure, avoiding mines, destroying enemies and resurfacing for air.

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.

1) 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 [12], 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.

2) Initialization of Networks: In our work, we used orthogonal initialization for the 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 [13]. This can prevent issues such as the vanishing or exploding gradient problem. We found that this initialization strategy worked well in our context and helped to maintain a balanced importance

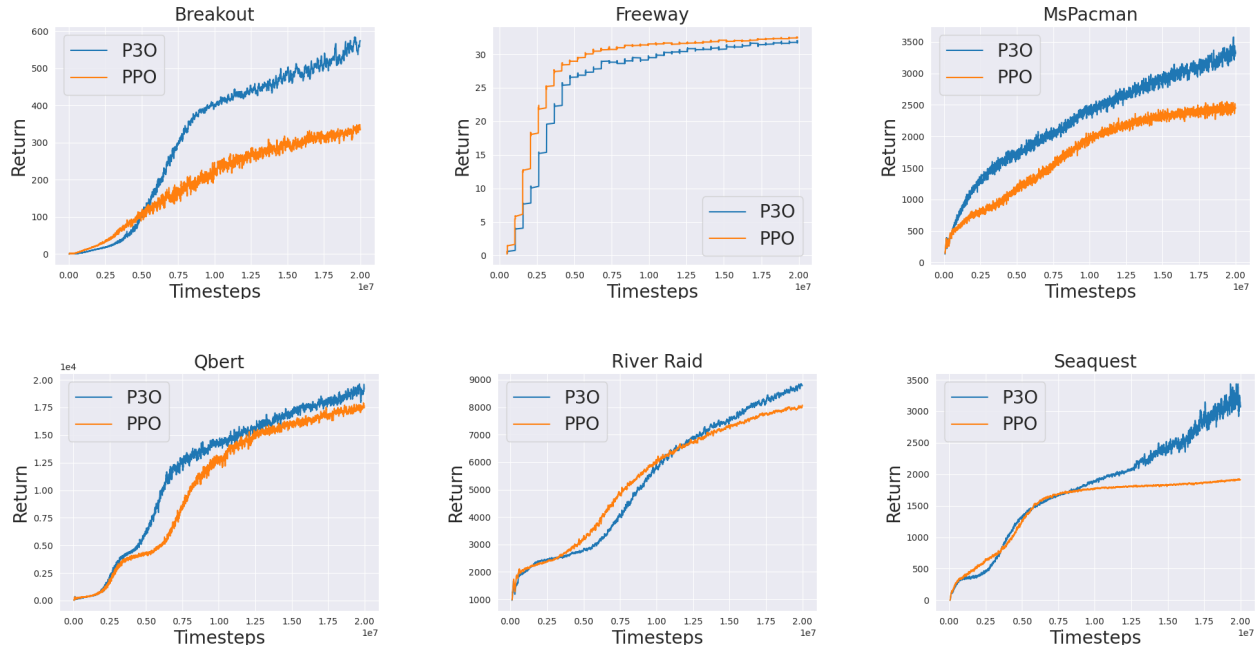

Fig. 1: These figures represents the average returns of each environment where the blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO). We first took the average of the last ten episodes and then calculated the average between 10 different runs at each episode. We then graphed the average of each episode to determine which model has better returns through training.

across states in the critic's learning process. We did experiment with other initialization strategies but found that orthogonal initialization provided the best results in our experiments.

C. Training Results

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 [14].

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. 1 for Breakout demonstrates that the final 10 episodes' average, across the 10 environments, reached a peak reward of 600. Examining Fig. 2, 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. 3 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. 2, where, starting from 15 million timesteps, there is a more significant increase in the variation of returns per episode.

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. 3, 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. 1 and Fig. 2. 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. 3. 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. 2

	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 I: 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.

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 Fig. 2, 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. 3, where, around the 15-20 million timesteps, the algorithm displays a considerably higher variance.

D. 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 I.

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, River Raid and MsPacman 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. In the River Raid environment, the algorithm demonstrated an impressive surge of approximately 800 in comparison to PPO. 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. In the Freeway environment, PPO have extremely similar rewards P3O that its comparison is negligible.

V. Discussion

The experimental results obtained from our study suggest that the proposed P3O algorithm demonstrates potential as a RL algorithm for Atari environments. P3O outperforms the baseline Proximal Policy Optimization (PPO) algorithm 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, River Raid 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.

The variance of the P3O algorithm and the baseline PPO algorithm is generally comparable, with exceptions such as Breakout, Seaquest, and River Raid, where the variance is higher.This is due because of the increase of 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 and River Raid environment, the algorithm for each episode was able to score very high returns in one episode and then in another episode it would have normal returns. This can be because the algorithm was able to discover a novel scoring method from these high-importance states, resulting in a significant reward increase.

In conclusion, our experimental results indicate that P3O is a promising RL algorithm for Atari environments, with improved performance in 5/6 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. However, further analysis is needed to ascertain the algorithm's effectiveness across a more extensive range of environments.

VI. Related Work

Anand and Precup [1] presented a comprehensive and insightful study on Preferential Temporal Difference (PTD) Learning. They evaluated their approach in grid-based tasks with fully observable and partially observable states and the Cart-Pole environment [11]. In the Cart-Pole environment, 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. However, this method is impractical for environments with non-trivial observation spaces (e.g., MuJoCo Environments [15]) or when states are represented as images, such as in Atari environments and uses predefined static beta value.

A related family of algorithms is Emphatic Temporal Difference (ETD) [16] algorithms. These algorithms employ an interest function to reweight updates. 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

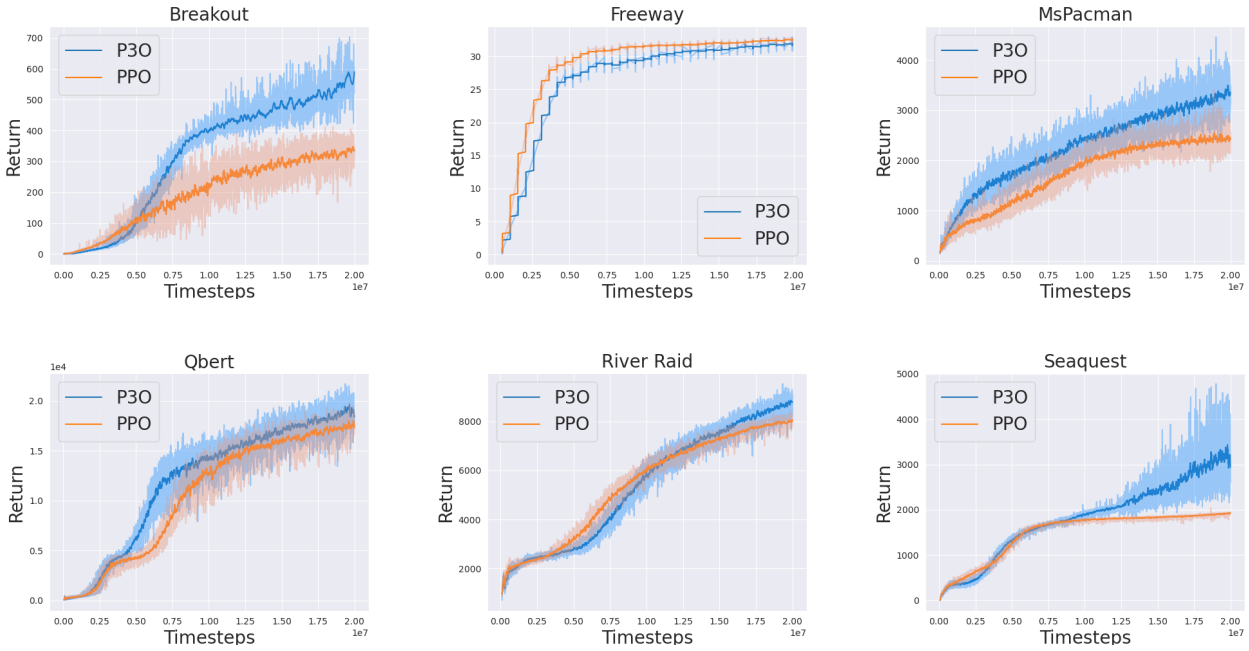

Fig. 2: These figures represents returns of each environment where it takes 10 different runs at each episode and computes the average. This is different from taking the mean of the last ten episodes and plots what return the model provides for each episode. The blue line is the proposed algorithm (P3O) and the orange line is the baseline model (PPO). The darker line for both orange and blue are the smoothed version of the returns of PPO and P3O which is displayed with the lighter version.

according to the agent's preference, ETD adopts a trajectory-based update method, while PTD employs a state-dependent parameter to adjust the updates.

The paper "Emphatic Temporal Difference Learning for Deep Reinforcement Learning Agents" [17] 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 reinforcement learning 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 reinforcement learning 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.

Preferential Temporal Difference (PTD) learning shares certain connections with the work of Chelu, Precup and Hasselt [18], who developed a backward model for predicting potential predecessors of observed states. They employed a planner-aware model, where the planner is responsible for credit assignment but the model is not model free.

VII. Conclusion and Future Work

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.

For future research, we propose assessing P3O in Mujoco environments [15], 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 Mujoco's environments. Additionally, we recommend testing the algorithm in partially observable environments, as P3O's employment of SI could result in

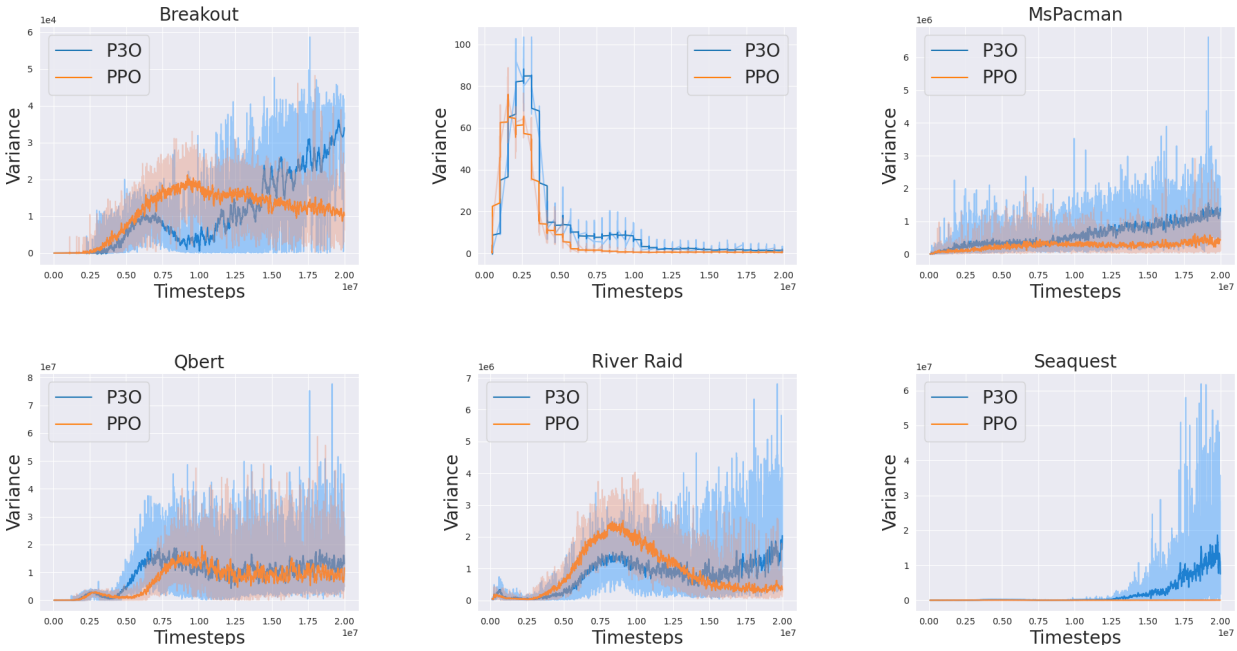

Fig. 3: To quantify the variance of the proposed algorithm and baseline model, we computed the variance of their performance over 10 independent randomly seeded runs on each environment. Specifically, we retrieved the variance of the total rewards obtained at each episode and plotted the results on a 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. The darker line for both orange and blue are the smoothed version of the returns of PPO and P3O which is displayed with the lighter version.

enhanced performance by allocating greater attention to states that yield high importance.

Acknowledgment

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

References

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

[2] I. Karino, Y. Ohmura, and Y. Kuniyoshi, "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*. Springer, 2020, pp. 366–378.

[3] J. S. Hunter, "The exponentially weighted moving average," *Journal of quality technology*, vol. 18, no. 4, pp. 203–210, 1986.

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

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

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

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

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

[9] OpenAI. (2018) Spinning up openai. https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[10] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

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

[12] J. Weng, M. Lin, S. Huang, B. Liu, D. Makoviichuk, V. Makoviychuk, Z. Liu, Y. Song, T. Luo, Y. Jiang, Z. Xu, and S. Yan, "EnvPool: A highly parallel reinforcement learning environment execution engine," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 22 409–22 421.

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

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

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

[16] R. S. Sutton, A. R. Mahmood, and M. White, "An emphatic approach to the problem of off-policy temporal-difference learning," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2603–2631, 2016.

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

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