Reproduction of Proximal Policy Optimization Algorithm

Erdoğan Yıldız
Computer Engineering MSc.
Hacettepe University
Ankara, Turkey
yildizcontact@gmail.com

I. Introduction

Since its introduction in 2017 by OpenAI, the Proximal Policy Optimization (PPO) Algorithm [1] has significantly influenced the field of Reinforcement Learning (RL), addressing a variety of tasks across diverse domains. Widely regarded as one of the leading state-of-the-art RL algorithms, PPO has demonstrated its versatility and efficacy in numerous applications, ranging from robotics and video games to energy management. A notable recent application includes its integration into GPT-4, enhancing the model's performance through Reinforcement Learning from Human Feedback (RLHF) [2]. This integration, as highlighted by the prominent computer scientist Andrej Karpathy, is instrumental in the success of GPT-4, which is considered a groundbreaking achievement in artificial intelligence [3].

The primary motivation behind choosing PPO for this reproducibility project was driven by a desire to deepen both theoretical and practical comprehension of Reinforcement Learning. Additionally, this project aims to demystify the underlying mechanics of the PPO algorithm, offering a detailed exploration of its mathematical and computational aspects.

This report will provide comprehensive insights into the process of reproducing the PPO algorithm. It will encompass a range of topics from foundational concepts to advanced technical implementations within the realms of RL and PPO. The report will also include critical discussions and personal reflections where pertinent.

II. Background and Preliminary Concepts

Reinforcement Learning (RL) constitutes a specialized domain within Machine Learning, dedicated to addressing sequential decision-making problems. This is primarily achieved by modeling such problems as Markov Decision Processes (MDPs). RL aims to devise an optimal decision-making agent within an MDP, tasked with the maximization of cumulative rewards. The scope of RL extends across diverse practical domains, including robotic manipulation, traffic control optimization, strategic gameplay, and financial trading. These areas typically involve intricate sequential decision-making challenges, where conventional control theory and supervised machine learning methodologies may prove inadequate due to complexities in explicit mathematical modeling or the collection of well-defined supervised datasets.

This section provides a concise overview of key concepts integral to the understanding of Reinforcement Learning.

A. Markov Decision Process

The Markov Decision Process (MDP) is a mathematical framework designed for modeling stochastic sequential decision-making. It can be considered an extension of Markov Chains, incorporating not only states and transition probabilities between these states but also integrating actions and rewards.

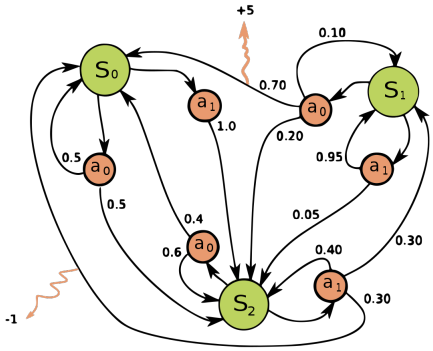

Fig. 1. Illustration of a Markov Decision Process (adapted from [4])

A key characteristic of MDPs is the First-order Markov Property, which posits that the probability of transitioning to the next state is dependent solely on the current state and action, expressed as:

$$P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, ..., s_0, a_0)$$

An MDP is fully described by:

- S: the state space, encapsulating all possible states.
- A: the action space, encompassing all possible actions.
- $P(s'|s, a)$: the environment model, denoting the probability of transitioning to the next state given the current state and action.
- $R(s, a)$: the reward model, reflecting the reward received for a given state and action. Typically, this is designed by the developer and is often independent of the action, thus it can be simplified as $R(s)$. [4]

Utilizing an MDP framework enables the application of various mathematical methods such as dynamic programming and temporal difference learning in stochastic decision-making processes.

It is important to note that while Reinforcement Learning often assumes MDP properties in systems, many practical problems are non-Markovian. Nevertheless, these systems can often be approximated to exhibit Markovian behavior through certain techniques. Even in cases where perfect Markovian representation is unattainable, empirical evidence suggests that RL solutions can still effectively operate. [5]

B. How Reinforcement Learning Works

In the paradigm of Reinforcement Learning (RL), two principal entities are recognized: the agent and the environment. The agent embodies the decision-making entity, which, at each timestep, observes the current state of the environment and executes an action. Each action undertaken by the agent leads to a change in the environment and brings about a reward. This reward, coming from the environment, serves as a feedback mechanism to guide the agent towards desirable behaviors. The agent's ultimate objective is to find the optimal policy — one that maximizes the expected cumulative future rewards through continual interactions with the environment.

The environment represents the context in which the agent operates. It responds to the agent's actions by providing updated states and rewards. The rewards, quantified numerically, are designed to indicate the desirability of states and actions as dictated by the environment's architect. A state or observation refers to the perceivable aspect of the environment's current condition as perceived by the agent. For instance, in a gaming environment, states could encompass the player's position, the locations of other entities, and similar factors.

Fig. 2. Interaction between Agent and Environment in Reinforcement Learning

The environment can be conceptualized as a stochastic process, the inner workings of which remain largely unknown to the agent. The agent, akin to a player in a game or an operator of robotic mechanisms, engages in continuous data acquisition through interactions with the environment. Its goal is to infer the environmental dynamics indirectly, using reward feedback to inform its decisions. Through this process, the agent strives to develop a profound comprehension of the environmental dynamics, enabling it to take actions that promise the greatest sum of future rewards.

C. Some Key Reinforcement Learning Concepts

1) Policy, $\pi(s)$: A policy is a function that maps each state to an action, essentially serving as a decision-making guide within an agent. The primary goal of an agent is to find out the optimal policy $\pi^*(s)$, which maximizes the cumulative sum of discounted future rewards during an interaction episode with the environment.

2) Return, $G(t)$: Return is defined as the total sum of discounted future rewards, formulated as:

$$G(t) = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

where γ is the discount factor, ranging between 0 and 1. It balances the focus between immediate and future rewards, ensuring mathematical convergence.

3) State Value, $V_\pi(s)$: The state value is the expected return for a given state under a policy π, denoted as:

$$V_\pi(s) = E_\pi[G(t)|s]$$

It represents the anticipated return if the agent adheres to policy π starting from state s.

4) State-Action Value, $Q_\pi(s, a)$: This value indicates the expected return for taking action a in state s and thereafter following policy π:

$$Q_\pi(s, a) = E_\pi[G(t)|s, a]$$

5) Trajectory, τ: A trajectory is a sequence representing the agent's continuous interactions with the environment, consisting of states, actions, and rewards.

6) Bellman Equation: Fundamental to RL, the Bellman Equation provides a recursive relationship between the value of a current state and its successor:

$$V_\pi(s) = E_\pi[R_{\text{next}} + \gamma V_\pi(s_{\text{next}})|s]$$

It asserts that the expected return for a state s under policy π equates to the expected next reward plus the discounted value of the next state. This equation is predicated on the Markov Property, implying that future states depend only on the current state and action, not on the sequence of past states and actions.

D. Continuous and Discrete Spaces

In Reinforcement Learning, state and action spaces can be either continuous or discrete. Discrete spaces consist of distinct, separate values, while continuous spaces involve a range of values.

Examples: In chess, each piece's board position is a discrete state, and moving a piece to a specific square is a discrete action. Conversely, in autonomous vehicle control, the car's speed and position are continuous states, and adjusting the steering wheel angle is a continuous action.

E. On-Policy vs Off-Policy Learning

These terms describe how an RL agent learns from its environment.

1) On-Policy Methods: In on-policy learning, the policy used for making decisions is the same as the one being optimized. An example is SARSA [6], and notably, PPO also falls under this category.

2) Off-Policy Methods: Off-policy learning involves optimizing a different policy from the one used to generate behavior. Q-learning [7] is a prominent off-policy method, where learning can occur from experiences gathered by different policies.

F. Two Main Approaches in Reinforcement Learning

1) Value-Based Methods: Value-based approaches focus on calculating state values (V(s)) or state-action values (Q(s, a)). These methods help in determining the most promising states and actions in terms of expected returns, indirectly guiding policy development.

2) Policy-Based Methods: Policy-based methods directly optimize the policy itself, aiming to select the best action at each step. These methods are advantageous in high-dimensional or continuous action spaces and permit the learning of flexible policies, such as stochastic ones. However, they generally require more interactions with the environment for effective learning.

PPO is an example of a policy-based method, renowned for its efficiency and robust performance across various benchmarks.

III. The Journey of Reproduction

A. Theoretical Path

The reproduction of the Proximal Policy Optimization (PPO) paper required extensive research and a deep understanding of Reinforcement Learning (RL) concepts. Initial stages focused on RL fundamentals, progressing through Markov Decision Processes, and evaluating dynamic programming methods like policy iteration and value iteration. Recognizing the limitations in practical scenarios, the study transitioned to Monte Carlo Estimation and Temporal Difference Learning methods, including Q-Learning and SARSA.

Subsequent phases involved examining Deep Q Networks [8], addressing challenges in continuous state space and exploring solutions to core RL issues like "moving targets" and "correlated samples." This detailed initial study of value-based methods established a strong foundation for the subsequent exploration of policy-based methods, ultimately leading to an investigation of the PPO algorithm.

Theoretical foundations were augmented through practical implementations, utilizing a variety of online resources, including courses, blog posts, and source codes, bridging the gap between theoretical knowledge and practical application.

B. Challenges

The comprehensive understanding required labor-intensive implementations and examinations of various methods. RL's reliance on probability and statistics concepts necessitated continuous learning and knowledge updating.

The transition from theoretical understanding to practical implementation presented significant challenges, involving the integration of diverse code snippets and solutions to tailor them for specific use cases.

Moreover, the inherent noise and fragility in RL posed difficulties in distinguishing between implementation errors and issues arising from hyperparameters or random seeds.

Despite these challenges, a solid understanding of RL basics and the intricacies of the PPO algorithm and a successful implementation were achieved.

IV. Policy Gradient Algorithms Background

A. REINFORCE Algorithm

The foundation of policy gradient algorithms can be understood through key equations and concepts, avoiding excessive detail. The evaluation of a policy utilizes an objective function, representing the return of a randomly selected trajectory under the policy. The goal is to maximize this value, enhancing the return of any trajectory sampled by the policy.

The objective function is denoted as:

$$J(\theta) = E[g(\mathcal{T}^{\pi_\theta})]$$

where θ represents policy parameters, $\mathcal{T}^{\pi_\theta}$ a random sequence of trajectories following policy π_θ, and $g(\tau)$ the return of a sample trajectory.

The gradient of this objective concerning θ can be calculated, facilitating the application of gradient ascent/descent methods. Utilizing the logarithmic transformation (the log trick), the gradient is estimated as:

$$\nabla_\theta J(\theta) = E[\nabla_\theta \log p(\tau_i) g(\tau_i)]$$

This formulation allows for estimating the gradient using sampled trajectories, essentially adjusting the log probability of trajectories that yield high returns. The equation can be approximated as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \log p(\tau_i) g(\tau_i)$$

Further simplification leverages the fact that the gradient of the log probability of a trajectory depends solely on the policy, leading to:

$$\nabla_\theta J(\theta) = E[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t, s_t) g(\tau_i)]$$

For gradient estimation using sampled trajectories, the following approximation is employed:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (g(\tau_i) \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t, s_t))$$

This formulation underpins the vanilla REINFORCE algorithm [9], which serves as the basis for subsequent policy gradient algorithms, including PPO.

To explain the practical implementation of the policy gradient theory discussed previously, the following presents the pseudocode for the vanilla REINFORCE algorithm. This algorithmic representation serves to bridge the gap between theoretical understanding and its application in policy gradient methods.

Algorithm 1 Vanilla REINFORCE Algorithm

Initialize policy parameters θ
while not converged **do**
    $\tau \leftarrow (s_0, a_0, r_0, \ldots, s_t, a_t, r_t, s_{t+1})$ Collect on-policy trajectory
    $\nabla \leftarrow 0$ Initialize gradient accumulator
    $G \leftarrow 0$ Initialize return
    for $t = T - 1$ **downto** 0 **do**
        $G \leftarrow \gamma G + r_t$
        $\nabla \leftarrow \nabla + \nabla \log \pi(a_t|s_t; \theta) G$
    end for
    $\theta \leftarrow \theta + \alpha \nabla$ Update parameters
end while

The implementation of REINFORCE employs 'rewards to go' or 'causal return' instead of full trajectory returns to ensure that past rewards do not influence the probability of future state-action pairs. This approach aims to refine the policy based on the future implications of state actions.

B. Actor-Critic Algorithms

Actor-Critic algorithms [10] were developed in response to the high variance and unstable learning characteristics inherent in the REINFORCE algorithm. The variance in return estimates in REINFORCE can lead to sample inefficiency, necessitating a larger number of samples for reliable estimation.

Actor-Critic approaches introduce a value estimator, named the critic, to address the high variance issue of vanilla REINFORCE. This critic, evaluating either state or state-action values, allows policy updates without needing complete trajectory information for return estimation. Consequently, the integration of a critic alongside the actor (usually a policy network) can significantly reduce variance, enhancing sample efficiency and stabilizing the learning process.

To transition from the REINFORCE to Actor-Critic methods, consider the refined gradient of the objective function:

$$\nabla_\theta J(\theta) = E[\nabla_\theta \log \pi(a_t, s_t) g(\tau_i)]$$

In the actor-critic framework, the return term can be substituted with lower-variance alternatives:

$$E[\nabla_\theta \log \pi(a_t, s_t) Q_w(s, a))]$$

or

$$E[\nabla_\theta \log \pi(a_t, s_t) A_w(s, a))]$$

Here, $A(s, a) = Q(s, a) - V(s)$ represents the advantage function, indicating the relative value of a state-action pair compared to the value of the state. Utilizing $Q(s, a)$ instead of the traditional "rewards to go" enables one-step updates and provides more stable estimates with lower variance at the cost of some value estimation bias. Employing the advantage function further reduces variance due to its normalized properties, without adding extra bias.

The pseudocode for a basic Actor-Critic algorithm is presented below:

Algorithm 2 Actor-Critic Algorithm [11]

Initialize parameters $\theta, \mathbf{w}$ and learning rates α_θ, α_w; sample $a \sim \pi_\theta(a|s)$.
for $t = 1, \ldots, T$ **do**
    Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$
    Then sample the next action $a' \sim \pi_\theta(a'|s')$
    Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \log \pi_\theta(a|s)$
    Compute the correction (TD error) for action-value at time t:
    $\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$
    Update the parameters of Q function:
    $\mathbf{w} \leftarrow \mathbf{w} + \alpha_w \delta_t \nabla_w Q_w(s, a)$
    Move to the next action and state: $a \leftarrow a'$, $s \leftarrow s'$
end for

V. Proximal Policy Optimization Theory

In the original paper on Proximal Policy Optimization (PPO) [1], the fundamental framework of the policy gradient method is elaborated. This framework is encapsulated in the formulation of an objective policy function $L^{PG}(\theta)$, which is expressed as:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right].$$

Herein, π_θ denotes the policy in question, whereas $\hat{A}_t$ signifies the estimated advantage at timestep t. The term $\hat{\mathbb{E}}_t$ represents the empirical average calculated over a finite batch of samples. The principal concept of this approach is to estimate policy gradients through the calculation of the gradients of the average of the logarithm of the probabilities of state actions, multiplied by the corresponding advantage estimates, over a collection of samples garnered by the policy π_θ.

To enhance sample efficiency, it may be advantageous to reuse these samples for more than a single gradient step towards the optimization objective. However, this approach encounters two primary challenges:

- Initially, post the first update, the samples no longer correspond to those collected under the current policy, leading to off-policy updates that potentially result in unstable learning.
- Secondly, even if the issue of deviation from the original policy is rectified using importance sampling, empirical evidence suggests that multiple gradient updates with these samples often result in excessively large and detrimental policy updates.

TRPO [12] offers a resolution to these challenges by integrating importance sampling with a strict constraint on the objective, as expressed in the following formulation:

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right]$$
$$\text{subject to} \quad \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta.$$

Where:

- $\pi_{\theta_{old}}$ is the vector of policy parameters prior to the update.
- $\hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]]$ represents the Kullback-Leibler (KL) divergence [13] between the old and the new policy, quantifying the extent of deviation of one probability distribution from another, expected distribution.
- δ is a small constant defining the size of the "trust region."

The Proximal Policy Optimization (PPO) algorithm enhances policy gradient methods by employing an advanced technique that utilizes the importance sampling ratio repeatedly. This is achieved by avoiding harmful policy updates and ensuring the updated policy remains within a designated "trust region" enforced by a Kullback-Leibler (KL) divergence constraint.

PPO introduces an innovative and more refined objective function, the "clipped surrogate objective," which lacks hard constraints and incorporates an additional mechanism. This mechanism disregards variations in the probability ratio, provided there is an opportunity to enhance the policy. The objective is mathematically represented as:

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

where:

- $r_t(\theta)$ is the probability ratio, also known as the importance sampling ratio, defined by $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ with $r(\theta_{\text{old}}) = 1$.
- ε is the clip range hyperparameter, typically set to 0.2.

Absent the "min" operator, the objective function already fulfills two critical roles: addressing the issues of off-sample handling and neglecting detrimental policy updates that could lead to catastrophic forgetting. The inclusion of the "min" operator provides an added advantage by disregarding the ratio limitation when there is a prospect for policy improvement.

The behavior of the "clipped surrogate objective" under varying conditions of advantage is depicted in the subsequent figure from the original paper.

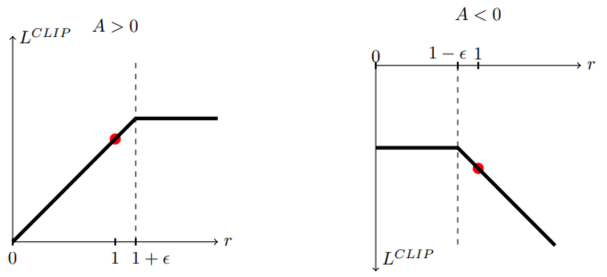

Fig. 3. Plots showing one term (i.e., a single timestep) of the surrogate function as a function of the probability ratio r, for positive advantages (left) and negative advantages (right). The red circle on each plot shows the starting point for the optimization, i.e., r = 1. ([1])

The objective function's behavior is intuitive: for a positive advantage, if the probability associated with a particular state-action pair decreases subsequent to the initial update, the gradient remains non-zero due to the absence of clipping. Consequently, even if the new ratio falls beneath $1-\varepsilon$, there is still an opportunity to augment the sample probability in the subsequent update. Conversely, with a negative advantage, a decline in the state-action probability is anticipated. However, should the importance sampling ratio increase, the PPO objective function offers a corrective opportunity by refraining from clipping the term.

Further examination of the of in Figure 3 reveals that if the state-action probability is updated in the anticipated direction, the term is clipped, rendering its gradients zero, thereby exerting no influence on the impending update.

Additionally, PPO, akin to various other Actor-Critic Algorithms, facilitates the operation of multiple environments in parallel, substantially mitigating the issue of sample correlation—often referred to as "the problem of correlated samples."

In conclusion, the 'clipped surrogate objective' provides the Proximal Policy Optimization (PPO) algorithm with a strong policy update mechanism. This function is crucial in improving sample efficiency by enabling the repeated use of the same sample set. The ease of its implementation is another notable feature that helps maintain stability in policy updates.

It is important to mention that the seminal PPO study evaluates several loss function configurations. However, this report has focused on elucidating and employing the version identified as the most efficacious according to the original research findings.

VI. Technical Concepts Regarding Implementation

In this section, we delve into several key technical concepts important to Proximal Policy Optimization (PPO) and its implementation. The focus will be on presenting the fundamental ideas, underlying intuitions, and essential equations pertinent to PPO, while consciously avoiding unnecessary details. This approach aims to provide a clear and concise overview of the key aspects of PPO necessary for understanding its practical applications.

A. Importance Sampling

The method of importance sampling is utilized to estimate the expected values with respect to one distribution by employing samples from another. This methodology is particularly advantageous in the context of PPO implementations. It facilitates the execution of multiple updates using the same rollout samples. Post the initial update, these samples become off-policy; however, importance sampling enables the use of these samples as if they were sampled under the current policy distribution.

Consider the task of computing the expected value of a function $f(X)$, where X is a random variable adhering to the distribution $P_A(x)$. In scenarios where only samples from a distribution $P_B(x)$ are available, the expectation can be reformulated by multiplying the integrand by the ratio $\frac{P_B(x)}{P_B(x)}$, which is identity one. The derivation is outlined as follows:

$$\begin{aligned} E_{X\sim P_A(x)}[f(X)] &= \int_X P_A(x)f(x)dx \\ &= \int_X \frac{P_A(x)}{P_B(x)}P_B(x)f(x)dx \quad \text{if } P_B(x) \neq 0 \\ &= \int_X P_B(x)\frac{P_A(x)}{P_B(x)}f(x)dx \\ &= E_{X\sim P_B(x)}\left[\frac{P_A(x)}{P_B(x)}f(X)\right] \end{aligned}$$

The term $\frac{P_A(x)}{P_B(x)}$ is referred to as the importance sampling ratio. Utilizing samples from $P_B(x)$, we can now approximate the expectation with respect to $P_A(x)$. This ratio inherently prioritizes samples more likely to be drawn from $P_A(x)$ while discounting those less likely to be representative of $P_A(x)$.

B. Multivariate Gaussian Distribution

The multivariate Gaussian distribution serves as a generalized model for representing a high-dimensional normal distribution, which is inclusive of the correlation between features. The original paper on PPO references the application of a Gaussian distribution for action sampling. In contrast, the multivariate variant has been adopted in this implementation due to its more comprehensive nature. It not only accommodates multiple Gaussian distributions but also captures the interrelations between their dimensions.

A Multivariate Gaussian Distribution is characterized by two principal components:

- **Mean Vector (μ)**: This is a vector of means $\mu = [\mu_1, \ldots, \mu_n]$, with μ_i denoting the mean of the i-th feature.
- **Covariance Matrix (Σ)**: A symmetric matrix that defines the variances and covariances of the distribution. The variances of individual features are represented along the diagonal, whereas the covariances between pairs of features are represented by the off-diagonal elements.

In the context of policy networks, the Multivariate Gaussian Distribution is employed to represent the action distribution. It is noteworthy that for the current implementation, the off-diagonal elements of the covariance matrix have been set to zero, thereby assuming the independence of each action variable, as was suggested by the original authors. This assumption simplifies the model to independent Gaussian distributions, yet the multivariate framework offers a more adaptable infrastructure for potential exploratory developments.

Moreover, the implementation explores various configurations of the covariance matrix, including fixed variances on the diagonal elements, decreasing variances over time, and fully learnable variances as posited by the original authors. These variants and their respective impacts will be elaborated upon in subsequent sections.

A question may arise regarding the computation of gradients through a Gaussian distribution. To address this, the "reparameterization trick" [16] is employed, leveraging the automatic differentiation capabilities provided by the Torch library, as referenced in the literature.

C. Learning Rate Annealing

The practice of learning rate annealing [14] relates to the strategic adaptation of the learning rate throughout the training process, which often involves a systematic reduction over time—a technique known as learning rate decay. There are many methods related to the adjustment of the learning rate, each designed to enhance the stability of the learning process. Although not explicitly addressed within the original PPO paper, this technique has been integrated into the present implementation. It is suggested that the use of learning rate annealing has significantly helped in achieving favorable outcomes.

D. Adaptive Moment Estimation (Adam) Optimization

Adaptive Moment Estimation, commonly abbreviated as Adam, is an optimization algorithm that has gained widespread adoption in the field of deep learning [15]. It synthesizes the concepts of Momentum and RMSprop optimization techniques. In essence, Adam leverages the moving averages of the gradients, which serves to mitigate the fluctuations inherent in the update process, thereby facilitating navigation through local minima due to the Momentum component. Concurrently, it employs adaptive learning rates for each weight in the network through the RMSprop component, thereby optimizing the learning process with a degree of specificity to each parameter. The authors of the PPO study explicitly utilized the Adam optimizer, and this decision is mirrored in the current implementation.

E. Entropy Bonus

An entropy bonus term is suggested in the original PPO paper as an optional augmentation to the objective function with the purpose of promoting exploratory behavior. Entropy, in the context of a probability distribution, quantifies the level of uncertainty or the unpredictability associated with the outcomes. A common measure of entropy within the realm

of information theory is the Shannon Entropy, which, for a discrete probability distribution, is defined as follows:

$$H(X) = -\sum_i P(x_i) \log P(x_i)$$

For a Multivariate Gaussian Distribution, entropy is expressed by the equation:

$$H(X) = \frac{1}{2} \log \left((2\pi e)^n |\Sigma| \right)$$

where $|\Sigma|$ denotes the determinant of the covariance matrix. A higher entropy value for a policy distribution implies greater unpredictability and, by extension, a greater degree of exploration. Within the implemented model, the entropy term has been incorporated into the actor loss, scaled by an entropy coefficient, which is preset to zero.

F. KL Divergence

The Kullback-Leibler (KL) divergence [13] serves as a metric for quantifying the divergence between two distinct probability distributions. For discrete distributions P and Q defined over the same probability space, the KL divergence is formulated as:

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

While the KL divergence was initially introduced as a strict constraint in the Trust Region Policy Optimization (TRPO [12]) framework, the authors of the PPO paper explored an alternate objective function with a KL penalty term. Despite this, empirical evidence suggests that this alternative yields worse performance compared to the "clipped surrogate objective", leading to its exclusion from standard practice. In the current implementation, an optional KL divergence checkpoint is employed post-batch updates to halt learning upon detection of significant divergence between successive policy iterations.

G. Gradient Clipping

Gradient clipping is a protection against the escalation of gradients to extreme magnitudes, a phenomenon known as exploding gradients. This preventative technique is widely adopted across various policy gradient methodologies. Although not specified in the PPO paper, gradient clipping has been optionally integrated into the present implementation. This inclusion helps to strengthen the training process against potential instabilities that come from excessively large gradient values.

H. Generalized Advantage Estimation (GAE)

Generalized Advantage Estimation (GAE) [17] is a technique employed in policy gradient algorithms to improve performance. It is acknowledged that while one-step advantage estimates are effective due to their diminished variance, they introduce a bias. Conversely, using "rewards to go" instead of one-step estimates augments variance without biasing the estimation.

GAE offers a mechanism to compute a generalized advantage value, incorporating a novel hyperparameter λ to navigate the bias-variance tradeoff for the advantage estimates.

- For $\lambda = 0$, the GAE converges to the one-step advantage estimates, analogous to the Temporal Difference (TD) error, characterized by lower variance but heightened bias.
- For $\lambda = 1$, it aligns with the Monte Carlo return estimates, known as "rewards to go", which bear increased variance but mitigate the bias.

The generalized formula for GAE is expressed as:

$$A_t^{\text{GAE}}(\lambda) = \sum_{k=0}^{\infty} (\gamma\lambda)^k \delta_{t+k}$$

where δ_t signifies the TD error at time t, defined by:

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Thus, for $\lambda = 0$, the GAE simplifies to the TD error, which is also the one-step advantage estimate:

$$A_t^{\text{GAE}}(0) = \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

For $\lambda = 1$, the equation condenses to the sum of discounted returns:

$$A_t^{\text{GAE}}(1) = \sum_{l=0}^{T-t-1} \gamma^l r_{t+l} - V(s_t)$$

In the original PPO research [1], the GAE was employed with $\lambda = 0.95$, optimizing the balance between bias and variance. To enhance stability, the GAE values are further normalized by subtracting their mean and scaling by their standard deviation within the implemented model.

VII. Implementation Details

In this section, we clarify the implementation specifics of the Proximal Policy Optimization algorithm. The discussion will encompass concise explanations and, where relevant, comparative analyses.

A. Programming Language

The language of choice for the implementation is Python. The original PPO paper does not mandate the use of any particular programming language. Python, however, is favored due to its straightforward syntax, versatility, and the extensive suite of libraries available for machine learning and reinforcement learning applications.

Further reinforcing this choice is the fact that the authors employed the MuJoCo physics engine [18] for seven simulated robotic tasks, which are encapsulated within the OpenAI Gym framework [19], a platform tailored for Python.

For machine learning operations, PyTorch has been adopted due to its dynamic computation graphs and integrated automatic differentiation engine. It offers user-friendly interfaces for defining and optimizing neural networks.

B. Simplified Codebase Structure

The codebase is architecturally segmented into four fundamental modules, arranged hierarchically. At the apex of this hierarchy is the Trainer module, which is responsible for interfacing with the environment, processing hyperparameters specific to PPO, and managing the visualization of results. It yields trained models and performance charts as outputs. The Trainer delegates computational tasks to the PPO class, which encapsulates the algorithm's logic. This class, in turn, interacts with ancillary classes dedicated to network operations and algorithmic computations, as depicted in the accompanying figure.

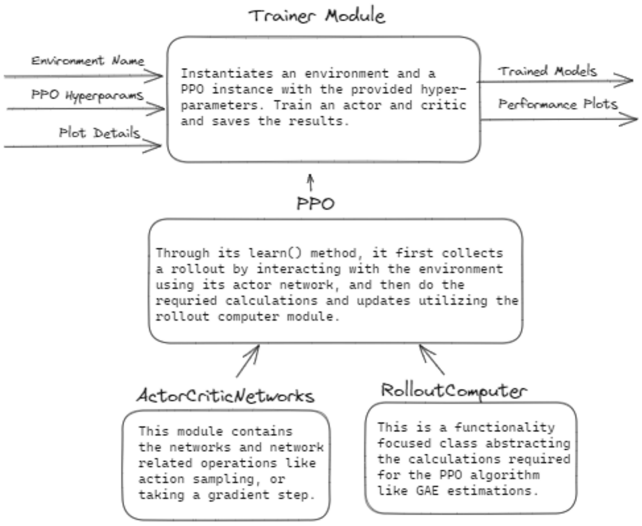

Fig. 4. Simplified Software Structure

The codebase will be made available for perusal to provide a comprehensive understanding of the underlying source code.

C. Networks

The PPO framework [1], as described in the literature [12], [20], recommends the use of a Multilayer Perceptron (MLP) comprising two hidden layers, each consisting of 64 units with tanh activation functions. This configuration is prescribed for the policy network, which outputs means for action distributions with variable standard deviations. While the paper refrains from specifying the architecture of the value network, it distinctly mentions that the policy and value networks do not share parameters.

In the current implementation, two analogous networks have been employed, each featuring two hidden layers of 64 units. These networks are designed to contain different activation functions, namely ReLU and tanh, with the type of activation being a selectable parameter.

The value network is configured to output a singular value, indicative of the state value estimate. In contrast, the policy network generates a mean vector for the action distribution, instrumental in constructing a multivariate Gaussian distribution. Additionally, the policy network is optionally equipped to output a standard deviation, derived from state-independent learnable parameters.

The implementation has been subjected to an empirical evaluation of four distinct methodologies for standard deviation estimation:

1) Learnable parameters contingent on the states, representing the logarithm of the standard deviation for numerical stability, yielded suboptimal performance.
2) Learnable state-independent parameters, indicative of the log standard deviation as recommended by the paper, manifested satisfactory results.
3) Fixed standard deviations, devoid of learnability, also performed adequately.
4) Decaying standard deviations, which diminish over time, implying a gradual transition from exploration to exploitation. This method emerged as the most efficacious, substantiated by the superior results it produced, and thus was established as the default hyperparameter configuration.

It is important to note that for all the mentioned approaches to standard deviation estimation, the covariances for the multivariate Gaussian distribution were constrained to zero, thereby focusing exclusively on the variances.

D. Hyperparameters

The implementation has been engineered with a increased degree of flexibility, incorporating optional optimizations that can be conveniently modulated through an extensive set of hyperparameters. These parameters are integral to the PPO instance and provide a thorough way to measure the influence of various optimizations:

- **seed** (*int*): Ensures the reproducibility of results.
- **rollout_len** (*int*): Dictates the number of steps in a rollout.
- **max_episode_len** (*int*): Imposes a stringent limit on the number of steps in an episode.
- **gamma** (*float*): The discount rate for future rewards.
- **gae_lambda** (*float*): Modulates the bias-variance tradeoff in advantage estimation.
- **n_updates_per_iteration** (*int*): Specifies the frequency of network updates after each rollout.
- **clip** (*float*): The clipping parameter for the PPO clipped surrogate objective.
- **lr** (*float*): The initial learning rate for the actor and critic networks.
- **min_lr** (*float*): The lower threshold for learning rate, facilitating decay.
- **batchify** (*bool*): Activates batch learning when set to true.
- **batch_size** (*int*): Applicable when batch learning is enabled, determining the size of each batch.
- **clip_grad** (*bool*): Engages gradient clipping to mitigate exploding gradients.
- **max_grad** (*float*): The maximum permissible gradient magnitude, relevant only if gradient clipping is activated.
- **max_kl_divergence** (*float*): The upper bound for KL divergence during a rollout update.

- **ent_coeff** (*float*): Adjusts the significance of the entropy bonus in the actor loss.
- **advantage_calc** (*str*): Selects the method for advantage calculation, either ”classic” or ”gae.”
- **learn_std** (*bool*): Determines if the standard deviation is to be learned adaptively.
- **std_start** (*float*): Sets the initial standard deviation when adaptive learning is not employed.
- **std_end** (*float*): Governs the decay of standard deviation, thereby modulating exploration over time.
- **tanh_acts** (*bool*): Determines the use of tanh activations versus ReLU for the networks.

These hyperparameters offer a way to adjust the PPO to different environments, goals, and desired results, and are crucial in dealing with the complex world of reinforcement learning.

E. Detailed Pseudocode

Utilizing logarithms of probabilities rather than direct probabilities is a standard practice within policy gradient algorithms, primarily due to the benefits of numerical stability and simplification of calculations.

Algorithm 3 PPO Algorithm Pseudocode

initialize the actor and critic networks
for total_timesteps **do**
    collect a rollout
    calculate rewards to go for the rollout states
    calculate value estimates for each rollout state using the critic.
    calculate advantage estimates
    for K steps: **do**
        for batches of rollout samples: **do**
            calculate current log probabilities for a batch of state actions.
            calculate importance sampling ratios for the state action probs for the batch.
            calculate current value estimates for the batch of states.
            update the actor using the clipped surrogate objective function (consisting of ratios and advantage estimates), optionally add entropy bonus.
            update critic using MSE loss between current V estimates and returns to go.
        end for
        check for KL divergence and optionally stop learning from this rollout.
    end for
    optionally reduce the learning rate
    optionally reduce the standard deviation
    log the average episodic lengths and rewards for the rollout
end for

F. Rollout Collection

In the rollout collection phase, the agent engages with the environment for a minimum duration equivalent to the predefined ”rollout length.” A single rollout may constitute a segment of an episode or a composite of several successive episodes. While Proximal Policy Optimization (PPO) is compatible with learning concurrently across multiple environments, the original paper does not elaborate on this parallelization. Consequently, the implementation described in this report is limited to a single environment setup. During each interaction, the transition data is recorded, including the current state, action taken, log probability of the action, subsequent state, reward received, termination flag, and a truncation flag, the latter being specific to the recent versions of gym environments.

G. Selected Benchmark

The benchmark employed in the original study comprises seven simulated robotic tasks within the OpenAI Gym framework [19]. These tasks, characterized by continuous control challenges, are powered by the MuJoCo physics engine [18].

The authors carried out a comparative analysis of three distinct objective functions, each with unique hyperparameter configurations. Upon identifying the superior performance of the clipped surrogate objective function, they proceeded to compare the Proximal Policy Optimization (PPO) algorithm with other contemporaneous algorithms. This comparison was performed across the same seven tasks, with each algorithm subjected to training for one million timesteps and evaluated using three distinct random seed values.

The findings reported in the paper suggest that PPO surpasses the majority of previously established methods, particularly in the domain of continuous control tasks.

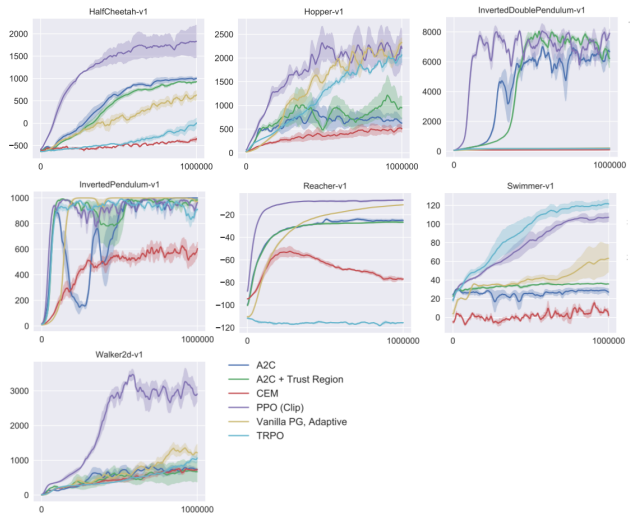

Fig. 5. Comparison of several algorithms on several MuJoCo environments, training for one million timesteps. [1]

In reproducing this study, updated versions (v4) of the previously mentioned MuJoCo environments have been utilized,

as the initial versions (v1) are now obsolete. The aim is to validate the findings of the original paper by achieving comparable results. A thorough review of the environment documentation has been conducted to ensure that no substantive changes have been made that could potentially change the complexity or the evaluative integrity of the benchmark tasks.

VIII. Results

The inherent variability of Reinforcement Learning (RL), attributable to its sensitivity to hyperparameter modifications and random seeds, presents considerable challenges in debugging implementations. To mitigate these issues, the implementation has been designed to be highly adaptable, making the process of experimentation more efficient.

Numerous configurations have been trialed, yet for the sake of conciseness, this report will discuss only two, with a subsequent analysis of the most efficacious hyperparameters based on the outcomes.

The objective was to replicate the performance graphs akin to Figure 5 from the original PPO paper, which illustrates the algorithm's efficacy across varied continuous control tasks. In doing so, only the PPO(Clip) performance aspect of these original performance plots, as specified in the figure, was considered. Consistent with the authors' methodology, three distinct seeds were employed for each training instance, and the mean rewards per rollout, inclusive of variances, were charted.

A. Closest Configuration to the Paper

The initial objective was to closely mirror the parameters and configurations recommended in the original study.

Figure 6 displays the results obtained from the Proximal Policy Optimization (PPO) algorithm. This implementation followed the configurations specified in the original paper.

Although the original paper did not provide all hyperparameters, those that were specified were rigorously followed. For unspecified parameters, standard default values were employed.

The configuration included: a rollout length of 2048, a learning rate (lr) of 3×10^{-4}, ten updates per iteration, batch size of 64, a discount rate (γ) of 0.99, Generalized Advantage Estimation (GAE) with a λ value of 0.95, adaptable standard deviations (learn_std), tanh activation functions (tanh_acts), and a clipping parameter set at 0.2.

Even though the suggested parameters were closely copied, there were noticeable differences in learning performance when compared to the results in the paper. Specifically, for the HalfCheetah environment, the expected average rollout reward was approximately 2000, but our implementation achieved only around 700. Similar disparities were observed in other environments: Hopper (expected: 2000, actual: 500), InvertedDoublePendulum (expected: 8000, actual: 1000), Swimmer (expected: 120, actual: 35), and Walker2d (expected: 3000, actual: 400). Conversely, for the InvertedPendulum and Reacher environments, the results were comparably close to the original, albeit with slight differences.

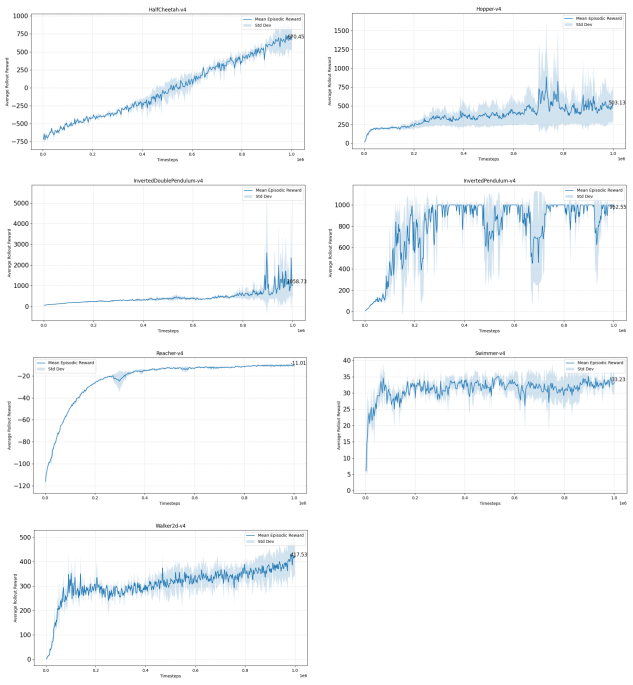

Fig. 6. Performance of the PPO Algorithm in Seven MuJoCo Continuous Control Environments, Replicating the Configuration Advised in the Original Paper, with Training Extended to One Million Timesteps.

These discrepancies might be attributed to the inherent stochasticity in Reinforcement Learning (RL), the influence of initial seed values, or subtle variances in implementation details and the unspecified hyperparameters.

B. Optimized Configuration and Performance

This subsection explores the configuration that yielded the most effective results for the PPO algorithm. A meticulous selection and adjustment of hyperparameters led to significant improvements in performance, as detailed below and illustrated in Figure 7.

Configuration Details:

- Rollout length: 4096
- Initial learning rate (lr): 5×10^{-3}, decaying to 1×10^{-4}
- Updates per iteration: 10, without batch processing
- Discount rate (γ): 0.99
- Generalized Advantage Estimation (GAE) with λ of 0.98
- Fixed standard deviations, starting at 0.25 and decaying to 0.01
- Activation functions: ReLU (with tanh_acts set to false)
- Clip parameter: 0.2

Performance Analysis: The refined configuration emphasized the PPO algorithm's adaptability and robustness. In certain environments, such as HalfCheetah and InvertedDoublePendulum, the algorithm not only achieved but exceeded the benchmarks set in the original paper. Specifically, in the HalfCheetah environment, our implementation achieved an average rollout reward of approximately 3300, surpassing the expected 2000. Comparable or superior performances were

noted in other environments: Hopper (1500), InvertedDoublePendulum (9000), Inverted Pendulum (1000), and Reacher (-6).

However, in the Swimmer and Walker2d environments, the results did not meet the expected benchmarks, registering at roughly 50 (expected 120) and 1400 (expected 3000) respectively.

Convergence and Hyperparameter Tuning: An interesting observation was the slower convergence rate compared to the original implementation. This may be attributed to the refined hyperparameter tuning or potential variations in environmental dynamics, possibly due to updates in the environment versions, although no substantial changes in environment characteristics were noted.

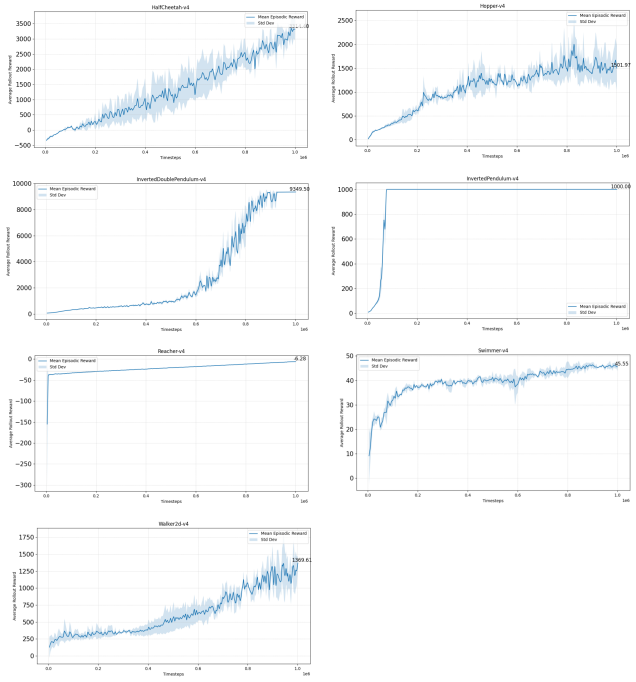

Fig. 7. Performance of the PPO Algorithm in Seven MuJoCo Continuous Control Environments with Optimized Configuration, Trained for One Million Timesteps.

C. Observational Analysis of Hyperparameter Efficacy

In the course of my experiments, some hyperparameters have shown a bigger impact on the performance outcomes. The most important of these is the calculation of advantage estimates. After a lot of testing with different ways of calculating these estimates, it has become clear that even small changes can quickly reduce the effectiveness of the model. Optimal results were achieved employing normalized Generalized Advantage Estimation (GAE) with a lambda value of 0.98, leveraging its capability to modulate the bias-variance trade-off in the estimates.

An equally important aspect was the employment of batch processing for rollout updates. Contrary to the original authors' endorsement of batch updates utilizing the Adam optimizer, my experiments yielded superior results when the entire rollout was used for updates. Although larger batches are theoretically anticipated to facilitate smoother updates, smaller batches have the potential to avoid local optima. Nevertheless, in my specific implementation, I found no benefits to batch processing. It is important to note that my implementation was executed on a CPU, negating memory constraints. However, in scenarios with more complex environments or where computational time is of the essence, batch processing may well be a must.

The learning rate also proved to be critical. While a standard rate of 3×10^{-4} is commonly adopted, I observed better results when using a decreasing learning rate, starting at 5×10^{-3} and reducing to 1×10^{-4}. It is suggested that higher initial learning rates speed up convergence, and then gradually lowering the rate helps to stabilize the learning.

Further incremental improvements were noted with longer rollout lengths, presumably because more varied and therefore less correlated samples were collected, leading to more consistent learning performance.

The 'clip' parameter within the objective function undoubtedly has a major influence, with my preference being to stick to the 0.2 recommendation suggested in the original paper.

In conclusion, the integration of a decayed variance vector as a function of time slightly outperformed learnable standard deviations in my context.

Conversely, the residual hyperparameters examined did not present substantial effects on performance, provided their values were not exceedingly altered. This was true for modifications in activation functions, such as substituting ReLU with tanh, variations in the number of updates per rollout, and the implementation of gradient clipping.

IX. Conclusion

This reproducibility report has carried out a detailed study of the basic theories behind Reinforcement Learning (RL), focusing particularly on policy gradient methods, and especially, Proximal Policy Optimization (PPO). The journey involved a thorough review of both the theoretical background and the practical details of implementing PPO. The findings and experiments done in this report not only confirm the effectiveness of the PPO algorithm but also provide a detailed explanation of its theoretical and practical sides.

The task of compiling this report has been as informative as it was challenging. It serves not only as an informative guide for the reader but also as a valuable learning experience in my academic journey. The insights and knowledge garnered through this process extend beyond the scope of this paper, laying the groundwork for my future research and applications.

Looking forward, the plan is to use the developed codebase and the insights from this study to possibly apply or improve the PPO algorithm in a specific setting. This future project aims to dive deeper into and perhaps innovate in the field of RL, building on the strong foundation created by this thorough analysis of PPO.

X. Acknowledgments

I express my gratitude to those who have contributed to my academic and practical education in reinforcement learning. Special thanks to the Lazy Programmer Team for their foundational course, which provided clarity in the core ideas of reinforcement learning. Appreciation is also due to Nazım Kemal Üre for the enriching ITU BLG638E Deep Reinforcement Learning course. Thanks to Mehmet Önder Efe for his supportive guidance and encouragement, which was pivotal in my exploration of interests and execution of this study. Lastly, my sincere thanks to the open-source AI community, notably Eric Yang Yu, Adrien Lucas Ecoffet, and many others whose names are too numerous to list but whose contributions are no less appreciated, for their invaluable contributions and collaborative spirit.

References

[1] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, Jul. 2017, revised Aug. 2017. [Online]. Available: https://arxiv.org/abs/1707.06347.

[2] L. Ouyang, J. Wu, and others, "Training language models to follow instructions with human feedback," arXiv preprint arXiv:2203.02155, 2022. [Online]. Available: https://arxiv.org/abs/2203.02155.

[3] Microsoft Developer, "State of GPT — BRK216HFS," YouTube, May 25, 2023. [Online Video]. Available: https://www.youtube.com/watch?v=bZQun8Y4L2A. [Accessed: Oct. 1, 2023].

[4] Wikipedia Contributors, "Markov Decision Process — Wikipedia, The Free Encyclopedia," [Online]. Available: https://en.wikipedia.org/wiki/Markov_decision_process. [Accessed: Nov. 1, 2023].

[5] [Siddharth Chandak, Pratik Shah, Vivek S Borkar, Parth Dodhia], "Reinforcement Learning in Non-Markovian Environments," arXiv preprint arXiv:2211.01595, 2022. [Online]. Available: https://arxiv.org/abs/2211.01595.

[6] Rummery, G. A. and Niranjan, M., "On-Line Q-Learning Using Connectionist Systems," November 1994. [Online]. Available: https://www.researchgate.net/publication/2500611_On-Line_Q-Learning_Using_Connectionist_Systems.

[7] Watkins, C. J. C. H. and Dayan, P., "Q-Learning," Machine Learning, vol. 8, pp. 279-292, 1992. Kluwer Academic Publishers, Boston.

[8] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., "Human-level control through deep reinforcement learning," Nature, vol. 518, pp. 529-533, Feb. 2015.

[9] Williams, R. J., "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Machine Learning, vol. 8, pp. 229-256, May 1992.

[10] Konda, V. R. and Tsitsiklis, J. N., "Actor-Critic Algorithms," Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, 02139. [Online]. Available: https://www.researchgate.net/publication/2354219_Actor-Critic_Algorithms.

[11] Weng, L., "Policy Gradient Algorithms," April 8, 2018. [Online]. Available: https://lilianweng.github.io/posts/2018-04-08-policy-gradient/.

[12] Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P., "High-Dimensional Continuous Control Using Generalized Advantage Estimation," arXiv preprint arXiv:1506.02438, 2015. [Online]. Available: https://arxiv.org/abs/1506.02438.

[13] Kullback, S. and Leibler, R. A., "On Information and Sufficiency," Ann. Math. Statist., vol. 22, no. 1, pp. 79-86, Mar. 1951. DOI: 10.1214/aoms/1177729694. [Online]. Available: https://www.jstor.org/stable/2236703.

[14] Nakamura, K., Derbel, B., Won, K.-J., Hong, B.-W., "Learning-Rate Annealing Methods for Deep Neural Networks," Electronics, vol. 10, no. 16, article 2029, Aug. 2021. DOI:10.3390/electronics10162029. [Online]. Available: https://www.researchgate.net/publication/354071446_Learning-Rate_Annealing_Methods_for_Deep_Neural_Networks.

[15] Kingma, D. P. and Ba, J., "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980, Dec. 2014, revised Jan. 2017. [Online]. Available: https://arxiv.org/abs/1412.6980.

[16] Kingma, D. P. and Welling, M., "Auto-Encoding Variational Bayes," arXiv preprint arXiv:1312.6114, Dec. 2013, revised Dec. 2022. DOI: 10.48550/arXiv.1312.6114. [Online]. Available: https://arxiv.org/abs/1312.6114

[17] Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P., "High-Dimensional Continuous Control Using Generalized Advantage Estimation," arXiv preprint arXiv:1506.02438, Jun. 2015, revised Oct. 2018. DOI: 10.48550/arXiv.1506.02438. [Online]. Available: https://arxiv.org/abs/1506.02438

[18] 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*, Vilamoura-Algarve, Portugal, Oct. 2012. DOI: 10.1109/IROS.2012.6386109. [Online]. Available: https://ieeexplore.ieee.org/document/6386109

[19] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W., "OpenAI Gym," arXiv preprint arXiv:1606.01540, Jun. 2016. DOI: 10.48550/arXiv.1606.01540. [Online]. Available: https://arxiv.org/abs/1606.01540

[20] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P., "Benchmarking Deep Reinforcement Learning for Continuous Control," arXiv preprint arXiv:1604.06778, Apr. 2016, revised May 2016. DOI: 10.48550/arXiv.1604.06778. [Online]. Available: https://arxiv.org/abs/1604.06778