Policy Gradient Methods in Reinforcement Learning

Varshitha Purra*, Pranay Palem† and Puneet Sai Naru‡
Emails: vpurra@asu.edu*, ppalem1@asu.edu†, pnaru@asu.edu‡

Abstract—**This paper provides a review of policy gradient methods in reinforcement learning (RL), focusing on their theoretical foundation, key algorithms such as REINFORCE, Actor-Critic, and TRPO, and applications in areas like robotic control and lunar lander. These methods offer advantages in continuous action spaces over traditional RL techniques. Additionally, we discuss the versatility of policy gradients in dynamic environments and potential future applications across diverse fields.**

Index Terms—**Reinforcement Learning (RL), Policy Gradient Methods, Actor-Critic, Trust Region Policy Optimization (TRPO), Robotic Control, Continuous Action Spaces, REINFORCE**

I. Introduction

Policy gradient methods are a foundational approach in reinforcement learning (RL), directly optimizing policies by computing the gradient of an expected reward function with respect to policy parameters. This direct optimization enables policy gradient methods to effectively handle high-dimensional and continuous action spaces, distinguishing them from traditional value-based RL methods such as Q-learning. In Q-learning, the action-value function, $Q(s, a)$, represents the expected cumulative reward for a given state-action pair, and the policy is derived indirectly by maximizing $Q(s, a)$. In contrast, policy gradient methods optimize the parameterized policy $\pi_\theta(a|s)$ directly by calculating the gradient of the expected return, $J(\theta)$, with respect to the parameters θ. This gradient, expressed as $\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s, a)]$, allows agents to learn policies without requiring a fully accurate value function, making these methods particularly advantageous in environments with complex dynamics and continuous action spaces (Kumar et al. 2024; Ståhlberg, Bonet, and Geffner 2023).

A key advantage of policy gradient methods lies in their flexibility and robustness, particularly in dynamic environments characterized by uncertainties or non-stationary conditions. Robust policy gradient approaches incorporate modifications to the standard expected return objective, introducing risk-sensitive components or robustness measures to ensure stable performance. For instance, (Wang and Zou 2022) demonstrates how risk-aware modifications to the reward function enhance agent efficacy under unpredictable settings. Such extensions allow policy gradient methods to address critical challenges, such as balancing exploration and exploitation, improving sample efficiency, and ensuring stability during training (Lehmann 2024; Wang and Zou 2022).

Policy gradient methods have been widely adopted in diverse applications, showcasing their adaptability across various fields. In robotics, these methods enable agents to learn and execute precise control policies for tasks such as robotic arm manipulation, locomotion, and autonomous vehicle navigation. Unlike value-based methods, which require discretization in continuous environments, policy gradients can learn smooth and high-resolution control policies, making them ideal for these applications (Wang and Zou 2022). Furthermore, policy gradient methods have demonstrated success in strategy games, where agents must devise complex strategies in dynamic, multi-agent environments. Notably, (Ståhlberg, Bonet, and Geffner 2023) highlights the versatility of policy gradients in learning generalizable policies for such domains.

From a theoretical perspective, policy gradient methods extend beyond standard cumulative reward maximization by incorporating general utility functions into their objective frameworks. This extension, introduced by (Kumar et al. 2024), allows the optimization objective to encompass a wide range of utility functions, including those that are convex or concave. The objective can be expressed as $J_U(\theta) = \mathbb{E}_{\pi_\theta}[U(\sum_{t=0}^{\infty} \gamma^t r_t)]$, where U represents a utility function tailored to specific tasks. This generalization significantly broadens the applicability of policy gradient methods, enabling their use in fields requiring more nuanced reward structures, such as healthcare robotics, resource management, and multi-agent systems.

Despite these advancements, policy gradient methods face notable challenges, such as sample inefficiency and the risk of convergence to local optima. Research continues to address these issues, as seen in recent work like (Wang and Zou 2022) and (Lehmann 2024), which propose improvements to gradient estimation and stability.

In this project, we aim to implement the methods described in (Sadavarte, Raj, and Sathish 2021) for solving the Lunar Lander problem using reinforcement learning. The Lunar Lander environment, a classic testbed from OpenAI Gym, presents a challenging problem involving dynamic control in continuous spaces. We will compare the results of PPO, TRPO and Actor-Critic methods as applied to this problem. This will be done as part of our extra credit submission.

II. What are Policy Gradient Methods

Policy gradient methods are designed to solve the continuous control problem in reinforcement learning (RL) by directly optimizing a parameterized policy $\pi_\theta(a|s)$ over actions a given a state s. The goal is to maximize the expected return, $J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t]$, by following the gradient $\nabla_\theta J(\theta)$. To derive this gradient, the policy gradient theorem provides the foundation: it states that under certain regularity conditions, the gradient can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim d^{\pi_\theta}, a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right], \tag{1}$$

where $d^{\pi_\theta}(s)$ is the stationary distribution of states under the policy π_θ, and $Q^\pi(s,a)$ represents the action-value function, or the expected cumulative future reward starting from state s and taking action a (Kumar et al. 2024).

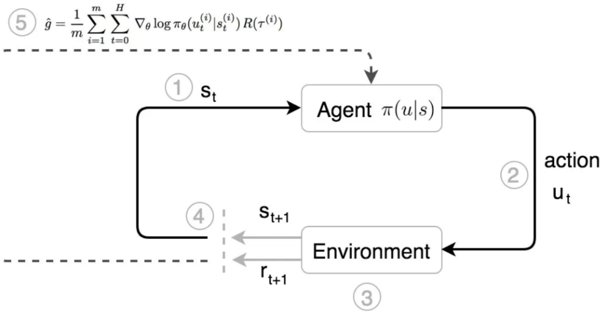
5
$\hat{g} = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(u_t^{(i)}|s_t^{(i)}) R(\tau^{(i)})$
1
s_t
Agent $\pi(u|s)$
2
action
u_t
4
s_{t+1}
Environment
r_{t+1}
3

Fig. 1. Illustration of the policy gradient method. Source: Jonathan Hui on Medium.

This expression relies on the likelihood ratio trick, where the term $\nabla_\theta \log \pi_\theta(a|s)$, known as the score function, captures how changes in the policy parameters θ impact the likelihood of choosing action a in state s. Importantly, this approach avoids the direct estimation of $d^{\pi_\theta}(s)$, which is often infeasible in high-dimensional spaces. Instead, we only need samples from the policy to compute updates, making the method scalable to continuous action spaces (Ståhlberg, Bonet, and Geffner 2023).

A. Application Example: Robotic Arm Control with Policy Gradients

A practical and impactful application of policy gradient methods is in controlling robotic arms, where the agent learns to optimize fine motor actions in real time. For instance, in an apprenticeship learning scenario, a robotic arm can be trained to replicate human-like manipulation by learning from expert demonstrations. This task can be formulated as a policy optimization problem, where the objective is not just to maximize a standard cumulative reward but to minimize a divergence measure between the expert's actions and the robot's actions, which adds complexity to the utility function (Wang and Zou 2022).

In this setting, the utility function $U(\theta)$ for the policy π_θ might be defined to capture both the imitation accuracy and smoothness of movements, represented mathematically as:

$$U(\theta) = -\mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t (\|a_t - a_t^{\text{expert}}\|^2 + \lambda \|\nabla a_t\|^2) \right], \tag{2}$$

where a_t^{expert} is the action taken by the expert, a_t is the action chosen by the policy, and λ is a weighting factor that penalizes rapid changes in action (smoothness constraint). The gradient of $U(\theta)$ can be calculated using policy gradients, yielding a learning signal that adjusts the policy to more closely imitate the expert's trajectory (Kumar et al. 2024; Lehmann 2024).

In practice, the robot collects samples of (s,a) pairs from the expert and applies gradient updates to reduce the divergence measure. This process is highly effective for robotic control tasks, where precise, real-time adjustments are crucial. The action-value function $Q^\pi(s,a)$, in this case, can incorporate not only a reward for task completion but also terms that penalize deviations from the expert's trajectory, achieving robust learning even in environments with high uncertainty or variability. As demonstrated in recent studies, this approach allows robotic arms to perform complex tasks with human-level dexterity and adaptability (Ståhlberg, Bonet, and Geffner 2023).

B. Application Example: The Lunar Lander Problem

The Lunar Lander problem, a well-known environment from OpenAI Gym, provides an excellent testbed for understanding the capabilities of policy gradient methods. The task involves landing a spacecraft on the lunar surface, balancing thrust and orientation to achieve a soft landing within a designated area. The state space includes features such as the lander's position (x,y), velocity (v_x, v_y), angle θ, and angular velocity ω, as well as binary indicators for whether the lander's legs are touching the ground. Actions correspond to firing the main engine or side thrusters, producing continuous control outputs.

The reward function r_t is defined to encourage desirable behaviors, such as reaching the landing pad and maintaining stability, and penalize undesirable ones, such as crashing or leaving the screen. Mathematically, the reward at time t can be expressed as:

$$r_t = \begin{cases} +100 & \text{if lander safely lands on the pad,} \\ -100 & \text{if lander crashes,} \\ -\alpha \|v_x, v_y\|^2 & \text{penalty for high velocity,} \\ -\beta \|\theta\|^2 & \text{penalty for angular misalignment,} \\ -\gamma \cdot \text{fuel\_usage} & \text{penalty for fuel consumption,} \end{cases} \tag{3}$$

where α, β, γ are weighting factors that control the penalties for velocity, angle misalignment, and fuel usage, respectively. The objective is to learn a policy $\pi_\theta(a|s)$ that maximizes the cumulative reward:

$$J(\theta) = \mathbb{E}_{\pi_\theta} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{4}$$

Policy gradient methods, such as Proximal Policy Optimization (PPO), are particularly effective in solving the Lunar Lander problem due to their ability to handle the continuous action space and learn smooth control policies. As described in (Sadavarte, Raj, and Sathish 2021), PPO employs a clipping mechanism to ensure stable policy updates, preventing large deviations from the current policy. This stability is crucial in dynamic environments like Lunar Lander, where small changes in actions can have significant effects on the outcome.

The Lunar Lander problem also serves as a benchmark for comparing various policy gradient algorithms. For instance, methods like Actor-Critic and TRPO can be applied to understand the trade-offs between sample efficiency, stability, and computational complexity. The ability to directly optimize the policy enables these methods to outperform value-based approaches, such as Deep Q-Networks (DQNs), which struggle in continuous control scenarios.

By applying policy gradient methods to the Lunar Lander problem, we hope to demonstrate their versatility and effectiveness in addressing challenging continuous control tasks. We will now delve into the detailed workings of various policy gradient algorithms, including REINFORCE, Actor-Critic Methods, TRPO, and PPO, to understand their methodologies, strengths, and practical applications.

III. How Policy Gradient Methods Work

Policy gradient methods vary in how they estimate and apply gradients to improve policies. This section explores four main types: (1) REINFORCE (or Monte Carlo Policy Gradient), (2) Actor-Critic methods, (3) Trust Region Policy Optimization (TRPO), and (4) Proximal Policy Optimization (PPO). Each of these methods uses distinct approaches to gradient estimation, optimization, and stability, making them suitable for different RL applications.

A. REINFORCE Algorithm

The REINFORCE algorithm, also known as the Monte Carlo policy gradient, is a foundational approach that estimates the policy gradient by running entire episodes and using the return from each episode as an estimate of future rewards. The gradient of the objective function, $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$, is expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \nabla_\theta \log \pi_\theta(a_t|s_t) G_t\right], \tag{5}$$

where G_t is the cumulative reward from time step t onward (Kumar et al. 2024). The use of $\nabla_\theta \log \pi_\theta(a_t|s_t)$, known as the score function, ensures that the update step increases the probability of actions that yielded high returns in the sampled episode.

The algorithm for REINFORCE is as follows:

Algorithm 1 REINFORCE Algorithm

```
Initialize policy parameters $\theta$
for each episode do
    Generate episode $(s_0, a_0, r_0, s_1, \ldots, s_T)$ by following $\pi_\theta$
    for each step $t$ in episode do
        Compute return $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$
        Update $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) G_t$
    end for
end for
```

REINFORCE offers several advantages over traditional methods such as Q-Learning and Temporal-Difference (TD) Learning. Unlike TD Learning, which relies on bootstrapped estimates of future rewards, REINFORCE computes gradients based on the actual return G_t, resulting in unbiased gradient estimates. Its model-free nature further enhances its adaptability, as it does not require a model of the environment, making it suitable for a wide range of tasks. Additionally, the probabilistic nature of the policy in REINFORCE inherently promotes exploration, which is crucial for identifying optimal actions in stochastic or uncertain environments. These attributes make REINFORCE particularly effective in solving continuous control problems, where discretization in Q-Learning would be impractical or result in suboptimal performance.

Applications of REINFORCE span various domains, including robotics, where it has been used to train robotic arms to perform fine-grained control tasks. Its episodic nature also makes it well-suited for optimizing flight paths of autonomous drones or spacecraft by leveraging full trajectory feedback. In strategy games, REINFORCE enables agents to explore and discover effective action sequences that maximize cumulative rewards. However, despite these strengths, REINFORCE has notable limitations. Gradient estimates in REINFORCE are prone to high variance due to their reliance on entire episode returns, which can fluctuate significantly. The algorithm is also sample inefficient, requiring many episodes to converge to an optimal policy. Moreover, updates to the policy parameters occur only at the end of an episode, which can lead to inefficiencies in environments with long episode durations. These shortcomings have driven the development of hybrid approaches, such as Actor-Critic methods, which combine the advantages of REINFORCE and TD Learning to address these challenges.

To mitigate some of these shortcomings, hybrid approaches like Actor-Critic methods have been developed, combining the benefits of REINFORCE and TD Learning.

B. Actor-Critic Methods

Actor-Critic methods address the high variance problem in REINFORCE by combining the policy (actor) with a value function (critic) that estimates expected returns. In this approach, the policy π_θ (actor) is updated using feedback from the critic, which provides an estimate of the value function $V^\pi(s)$ or the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.

The update step for the actor can be written as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a|s) A(s,a)\right], \tag{6}$$

where $A(s,a)$, often estimated as $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$, is the advantage function that stabilizes learning by indicating how much better the current action is than the expected action under policy π_θ (Wang and Zou 2022; Lehmann 2024).

Actor-Critic methods, such as Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C), are widely used in environments requiring continuous control, like robotic arm manipulation in industrial automation settings (Kumar et al. 2024).

Actor-Critic methods are highly effective in continuous control tasks and dynamic environments. For example, they are widely used in robotics, where algorithms like Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic

Algorithm 2 Actor-Critic Algorithm

```
Initialize policy parameters $\theta$ and value parameters $w$
for each episode do
    Generate a trajectory $(s_0, a_0, r_0, \ldots, s_T, a_T, r_T)$
    for each step $t$ in trajectory do
        Compute $A^\pi(s_t, a_t) = Q_w(s_t, a_t) - V_w(s_t)$
        Update $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a_t|s_t) A^\pi(s_t, a_t)$
        Update $w \leftarrow w + \beta(r_t + \gamma V_w(s_{t+1}) - V_w(s_t)) \nabla_w V_w(s_t)$
    end for
end for
```

(A3C) enable the learning of smooth and precise control policies for applications such as robotic arm manipulation, industrial automation, and bipedal locomotion (Kumar et al. 2024). In autonomous vehicle navigation, Actor-Critic methods have been applied to train agents capable of handling complex environments with dynamic obstacles while balancing short-term and long-term objectives (Ståhlberg, Bonet, and Geffner 2023). Additionally, in multi-agent environments such as strategy games, these methods facilitate the learning of adaptive strategies that consider the actions of competing agents (Wang and Zou 2022).

Compared to REINFORCE, Actor-Critic methods offer several advantages. By incorporating the value function, Actor-Critic methods reduce the variance of gradient estimates, allowing for more stable updates. This stability is particularly beneficial in environments with high-dimensional or continuous action spaces, where REINFORCE struggles with the noise in gradient calculations (Wang and Zou 2022). Moreover, Actor-Critic methods update the policy incrementally during an episode, as opposed to waiting until the episode concludes, which accelerates convergence and improves learning efficiency in environments with long episodes (Lehmann 2024).

However, Actor-Critic methods are not without limitations. The reliance on both the actor and the critic introduces additional computational complexity, as both components must be updated during training. Also, the approximation of the value function in the critic can introduce bias, which, if not managed properly, may hinder learning stability (Kumar et al. 2024; Wang and Zou 2022).

C. Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) extends policy gradient methods by introducing constraints on the policy updates to ensure that changes in the policy distribution remain within a "trust region." This prevents overly large updates that could destabilize learning. The objective in TRPO is modified to maximize $J(\theta)$ subject to a constraint on the Kullback-Leibler (KL) divergence between the new policy $\pi_{\theta'}$ and the old policy π_θ:

$$\max_{\theta'} \mathbb{E}_{\pi_\theta} \left[\frac{\pi_{\theta'}(a|s)}{\pi_\theta(a|s)} A(s,a) \right] \quad \text{s.t. } \mathbb{E}_{s \sim d^\pi} \left[D_{\mathrm{KL}}(\pi_\theta || \pi_{\theta'}) \right] \leq \delta, \tag{7}$$

where δ is a small positive constant that controls the step size of the policy update. The KL constraint ensures that the new policy remains close to the old policy, avoiding the risk of diverging from optimal behavior due to sudden changes in policy (Ståhlberg, Bonet, and Geffner 2023).

TRPO has been particularly successful in high-stakes applications like simulated robotic locomotion and autonomous vehicle navigation, where stability is crucial. By maintaining smooth, controlled updates, TRPO enables reliable policy improvements even in complex, non-linear environments, making it ideal for tasks that require precise, incremental learning adjustments.

Algorithm 3 TRPO Algorithm

```
Initialize policy parameters $\theta$ and baseline $V(s)$
for each iteration do
    Sample a batch of trajectories using $\pi_\theta$
    Estimate advantages $A(s, a)$ for each action in the batch
    Compute gradient $\nabla_\theta J(\theta)$ with the constraint $\mathbb{E}_s [D_{\mathrm{KL}}(\pi_\theta || \pi_{\theta'})] \leq \delta$
    Update $\theta$ by solving the constrained optimization problem
end for
```

Compared to simpler methods like REINFORCE, TRPO offers improved stability by incorporating the KL divergence constraint, which prevents policies from diverging too far from previous iterations. This not only stabilizes training but also allows for more consistent improvements in complex environments. Additionally, TRPO often outperforms Actor-Critic methods in tasks with intricate reward structures, where controlled updates are crucial for avoiding suboptimal local policies (Wang and Zou 2022).

However, TRPO is computationally intensive due to the constrained optimization problem it solves at each step, making it less suitable for real-time or resource-constrained applications. Additionally, the need to compute the KL divergence across policy distributions can slow down convergence compared to simpler algorithms like PPO.

D. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a highly efficient and widely adopted policy gradient method that improves upon Trust Region Policy Optimization (TRPO) by simplifying the optimization process while maintaining stability and performance. Unlike TRPO, which enforces a trust region through a computationally expensive Kullback-Leibler (KL) divergence constraint, PPO uses a clipped surrogate objective to ensure controlled policy updates. The PPO objective is defined as:

$$L^{\mathrm{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) A_t, \mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon) A_t \right) \right], \tag{8}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$ is the probability ratio, A_t is the advantage function, and ϵ is a hyperparameter that controls the allowable change in the policy. The clipping mechanism ensures that the updated policy remains within a safe range, preventing excessively large or small changes that could destabilize learning (Lehmann 2024).

PPO offers several advantages over TRPO. By replacing the KL-divergence constraint with the clipped objective, PPO eliminates the need for solving a constrained optimization problem, significantly reducing computational overhead. This makes PPO faster and easier to implement while achieving comparable stability and performance. The clipping mechanism is computationally lightweight and ensures robust policy updates without requiring additional hyperparameter tuning for constraints, as seen in TRPO (Ståhlberg, Bonet, and Geffner 2023; Wang and Zou 2022). Moreover, PPO's simpler formulation has made it more accessible for researchers and practitioners, contributing to its widespread adoption in reinforcement learning tasks (Kumar et al. 2024).

PPO has been successfully applied in a variety of domains. In robotics, it has been used for training agents in tasks such as robotic arm manipulation and bipedal locomotion, where stability and efficiency are critical for achieving smooth and precise control (Ståhlberg, Bonet, and Geffner 2023). Autonomous systems, including drone navigation and self-driving vehicles, have leveraged PPO to learn robust control policies in dynamic and uncertain environments (Wang and Zou 2022). Additionally, PPO has demonstrated its effectiveness in multi-agent strategy games, where agents must adapt to the actions of competitors and learn optimal policies in complex, interactive settings (Kumar et al. 2024).

Algorithm 4 Proximal Policy Optimization (PPO) Algorithm

```
Initialize policy parameters $\theta$
for each iteration do
    Collect trajectories by running the policy $\pi_\theta$ in the environment
    Compute advantage estimates $A_t$ for each time step
    Optimize the surrogate objective $L^{\text{CLIP}}(\theta)$ with respect to $\theta$
    Apply gradient ascent to update $\theta$ within the clipped range
end for
```

Despite its strengths, PPO has some limitations. The clipping mechanism, while effective in ensuring stability, can lead to overly conservative updates, slowing convergence in certain environments. Like other policy gradient methods, PPO requires substantial interaction with the environment, making it sample inefficient in tasks with sparse rewards or limited data availability.

IV. Why Use Policy Gradient Methods

Policy gradient methods offer unique advantages over traditional reinforcement learning (RL) approaches, particularly in environments with high-dimensional, continuous action spaces. Unlike value-based methods like Q-learning, which rely on discrete action-value functions, policy gradient methods optimize the policy directly. This direct optimization avoids the need for discretizing continuous spaces, allowing for smooth, high-resolution control policies. In domains such as robotic manipulation and autonomous vehicles, policy gradients have demonstrated their capability to adapt to dynamic environments while maintaining stability (Wang and Zou 2022; Ståhlberg, Bonet, and Geffner 2023).

Despite these advantages, the results from the reference paper highlight certain limitations of stochastic policy gradient methods, such as REINFORCE, in deterministic environments like LunarLander. The experiments show that REINFORCE often underperforms compared to value-based methods like Deep Q-Learning (DQN), as seen in the reward trends plotted in the images (Figure 2). The deterministic nature of the LunarLander environment makes value-based methods more effective, as they exploit the structured state-action mappings. Policy gradient methods, particularly stochastic approaches, struggle in such settings due to high variance and inefficient exploration. However, these results are environment-specific and may not generalize to real-world tasks involving uncertainty, where policy gradient methods are often more robust (Lehmann 2024).

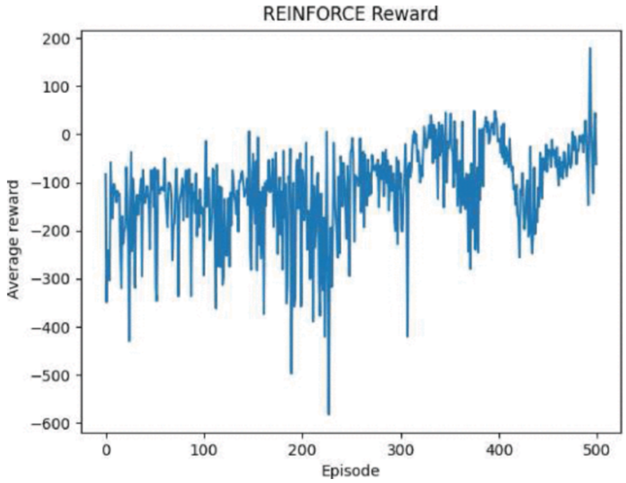

Fig. 2. Performance of REINFORCE in the LunarLander environment. The high variance in rewards highlights challenges in deterministic settings.

In our project, we aim to address these challenges by implementing advanced policy gradient methods, including Trust Region Policy Optimization (TRPO), Proximal Policy Optimization (PPO), Actor-Critic. These methods incorporate mechanisms to improve stability and efficiency, such as the KL-divergence constraint in TRPO, and the clipping mechanism in PPO. By applying these techniques to the LunarLander environment, we expect to achieve results that better align with the structured nature of the task while retaining the advantages of policy gradients in handling continuous control (Schulman et al. 2017; Lehmann 2024).

The choice of these methods is informed by their ability to address the specific shortcomings observed in REINFORCE. For example, PPO's clipping mechanism ensures stability during updates, which is critical in environments with sensitive control tasks like LunarLander. Actor-Critic methods reduce variance in gradient estimates by leveraging a value function to provide more informed updates. By combining these approaches, we aim to demonstrate the potential of advanced policy gradient methods to outperform both simpler policy gradients like REINFORCE and traditional value-based methods under the right conditions.

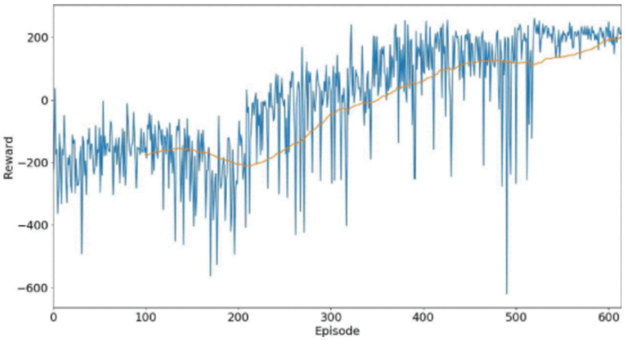

Fig. 3. Performance of Deep Q-Learning (DQN) in the LunarLander environment. The consistent upward trend demonstrates the effectiveness of value-based methods in deterministic settings.

While the reference results emphasize the limitations of stochastic policy gradients in deterministic environments, our use of TRPO, and PPO aims to explore whether these methods can bridge the performance gap and provide robust solutions to complex control problems.

V. Problem Setup and Environment

The Lunar Lander problem, a benchmark environment from OpenAI Gym, is a dynamic control task that involves guiding a spacecraft to achieve a soft landing on a designated lunar surface. This environment is discrete and episodic, offering challenges in balancing thrust and orientation. The task's difficulty arises from the stochastic nature of the environment, requiring robust learning algorithms to achieve consistent results.

A. State and Action Spaces

The environment's **state space** is an 8-dimensional vector containing the following:

- Lander's position: (x, y)
- Lander's velocity: (v_x, v_y)
- Angle (θ) and angular velocity (ω)
- Two binary indicators for ground contact of the left and right legs

The **action space** comprises four discrete actions:

- Fire the main engine
- Fire the left engine
- Fire the right engine
- Do nothing

B. Reward Function

The reward function incentivizes desirable behavior, such as achieving stability and landing within the target area, while penalizing undesirable actions like crashing or excessive fuel consumption. The reward structure is defined as follows (Sadavarte, Raj, and Sathish 2021):

$$\text{Reward} = \begin{cases} +100 \text{ to } +140, & \text{Landing on the pad} \\ -100, & \text{Crashing} \\ +10 \text{ per leg}, & \text{Ground contact} \\ -0.3 \text{ per frame}, & \text{Main engine firing} \\ -0.03 \text{ per frame}, & \text{Side engine firing} \\ +200, & \text{Solved the problem (ideal landing)} \end{cases} \tag{9}$$

The primary goal is to maximize the cumulative reward over the course of an episode.

C. Libraries Used

The implementation relies on the following libraries:

- **Gymnasium (OpenAI Gym fork)**: For creating and managing the Lunar Lander environment.
- **Stable Baselines3 (SB3)**: A popular library for implementing reinforcement learning algorithms, including PPO and TRPO (Raffin et al. 2021).
- **SB3-Contrib**: Extension to SB3 for advanced algorithms like TRPO.
- **Python Packages**: Supporting libraries such as `numpy` for numerical computations, `matplotlib` and `seaborn` for visualization, and `pandas` for data analysis.

D. Training Details

Each model was trained for 2.5 million timesteps across five seeds $(42, 100, 123, 2024, 999)$ to ensure reproducibility. The environment was reset with the respective seed before training to control randomness in initialization.

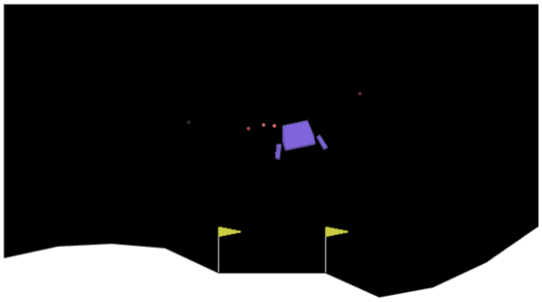

Fig. 4. Lunar Lander environment: balancing thrust and orientation to achieve a soft landing.

In subsequent sections, we discuss the application of PPO and TRPO algorithms, the results observed, and how hyperparameter tuning improved performance.

VI. Implementation of PPO and TRPO

Proximal Policy Optimization (PPO): PPO employs a clipped surrogate objective function to ensure stable policy updates. The probability ratio $r_t(\theta)$ between the new and old policies is clipped to avoid large updates. This mechanism ensures stability and efficiency during training. The clipped objective is given by:

$$L_{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right] \tag{10}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio, A_t is the advantage function, and ϵ is the clipping range.

Trust Region Policy Optimization (TRPO): TRPO incorporates a KL-divergence constraint to maintain policy stability. The objective is to maximize the expected advantage while ensuring the policy update remains within a trust region:

$$\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A_t\right] \quad \text{subject to} \quad \mathbb{E}_t \left[D_{\text{KL}}(\pi_\theta || \pi_{\theta_{\text{old}}})\right] \leq \delta \tag{11}$$

where δ is the threshold for KL-divergence.

A. Results and Observations

As seen in the figure, we've utilized moving average rewards. The moving average reward is a smoothed metric computed by averaging rewards over a sliding window of recent episodes. This helps reduce noise and highlight trends in performance over time, making it easier to analyze the agent's learning stability and consistency across seeds. For instance, a 100-episode moving average provides insights into long-term reward trends while mitigating the effects of individual episode variability.

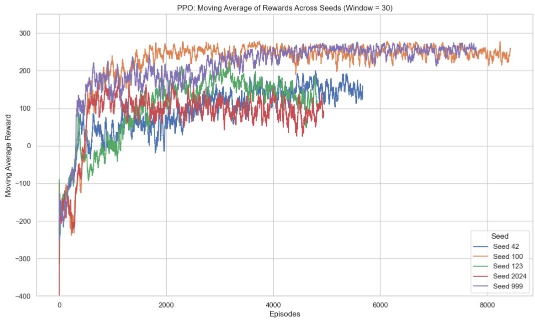

Fig. 5. PPO performance across seeds showing high variance.

PPO Results: - PPO demonstrated high variance across seeds, with some runs converging to rewards above 200 while others stagnated at suboptimal values (Figure 5). - This variance is attributed to the stochastic nature of policy initialization and exploration. - Example: Seed 100 achieved stable convergence quickly, while Seed 42 struggled to improve, highlighting sensitivity to initial conditions.

TRPO Results: - TRPO showed more consistent performance across seeds compared to PPO (Figure 6). - The KL-divergence constraint ensured stable updates, reducing sensitivity to initialization. - However, TRPO converged slower than PPO, demonstrating a trade-off between stability and efficiency.

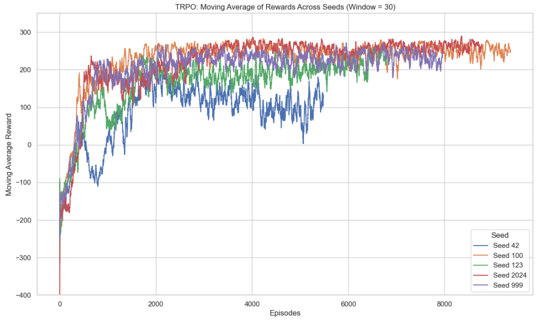

Fig. 6. TRPO performance across seeds showing lower variance.

B. Addressing Variance in PPO

The high variance observed in PPO results highlighted the need for improved stability. To address this, we decided to perform hyperparameter tuning, which will be discussed in detail in subsequent sections. This tuning focused on reducing the variability across seeds and improving overall performance consistency.

The experiments highlight the following:

- PPO is highly sensitive to initialization and exploration, requiring adjustments for consistent performance.
- TRPO provides better stability due to its constrained updates but converges more slowly than PPO.
- Both PPO and TRPO demonstrate advantages over value-based methods in solving high-dimensional control problems like the Lunar Lander.

In the next section, we will discuss the hyperparameter tuning process and its impact on PPO's performance.

VII. Hyperparameter Tuning

The initial PPO implementation exhibited high variance across seeds and inconsistent convergence due to suboptimal default hyperparameters. This section details the adjustments made to stabilize performance and improve consistency across training runs.

A. Initial Parameters and Challenges

In the initial setup, the following default hyperparameters were used for PPO:

- **Learning Rate (α)**: 0.0003
- **Number of Steps ($n_{\textbf{steps}}$)**: 1024
- **Batch Size**: 256
- **Clip Range (ϵ)**: 0.2
- **Entropy Coefficient (ent_coef)**: 0.01
- **Discount Factor (γ)**: 0.99
- **Generalized Advantage Estimation Lambda (λ)**: 0.95

While these parameters provided a baseline for learning, the high variance in results indicated poor optimization and sensitivity to exploration. Specifically:

- The smaller number of steps per update (n_{steps}) led to unstable gradient estimates.
- A larger batch size delayed updates, reducing responsiveness to environmental feedback.

- Learning rate tuning was insufficient to balance exploration and exploitation.

B. Tuned Parameters

After extensive experimentation, the following modifications were made to the hyperparameters:

- **Learning Rate** (α): Reduced to 3×10^{-4} for smoother optimization.
- **Number of Steps** ($n_{\textbf{steps}}$): Increased to 2048 to improve the quality of policy updates.
- **Batch Size**: Reduced to 64, allowing more frequent updates with finer granularity.
- **Clip Range** (ϵ): Retained at 0.2, providing stability during updates.
- **Entropy Coefficient (ent_coef)**: Kept at 0.01 to encourage exploration.
- **Discount Factor** (γ): Maintained at 0.99 to prioritize long-term rewards.
- **Generalized Advantage Estimation Lambda** (λ): Retained at 0.95 to reduce variance in advantage estimates.

C. Results After Tuning

The tuned parameters significantly reduced variance and improved performance consistency across seeds. The following results were observed:

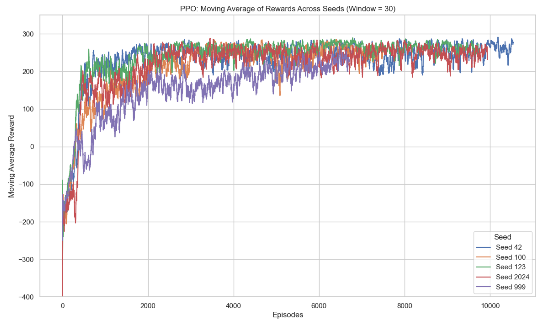

Fig. 7. PPO performance after hyperparameter tuning. Reduced variance and consistent convergence across all seeds.

- **Stability**: PPO exhibited smoother reward trends with less fluctuation across episodes, as shown in Figure 7.
- **Consistency Across Seeds**: All five seeds achieved rewards exceeding 200, with reduced sensitivity to initialization.
- **Faster Convergence**: The agent converged to optimal policies within fewer episodes, leveraging improved gradient estimates and more frequent updates.

Before Tuning:

- High variance across seeds due to noisy gradient estimates.
- Frequent suboptimal convergence in certain seeds (e.g., Seed 123).
- Unstable learning with prolonged training times.

After Tuning:

- Variance across seeds was minimized.
- All seeds consistently achieved high rewards.
- Learning stability was enhanced, improving reproducibility.

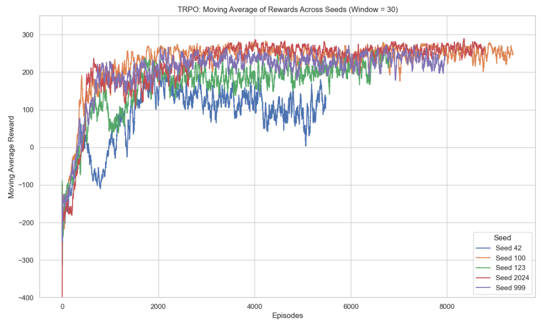

Fig. 8. TRPO performance after hyperparameter tuning

The results underscore the importance of hyperparameter optimization in reinforcement learning:

- Increasing n_{steps} improved the accuracy of policy updates by providing larger trajectory samples for gradient computation.
- Reducing the batch size allowed more frequent updates, enhancing responsiveness to the environment.
- Adjusting the learning rate smoothed the optimization process, balancing exploration and exploitation.

These adjustments demonstrate that careful hyperparameter tuning is critical to addressing instability and sensitivity in PPO.

VIII. Conclusion

Policy gradient methods have proven to be essential in reinforcement learning, particularly for handling complex, continuous action spaces where traditional value-based methods fall short. By directly optimizing policies through policy gradient methods like REINFORCE, Actor-Critic, and TRPO we are able to achieve smooth and adaptive learning, which is crucial in real-time applications like robotics and autonomous control. The underlying mathematics, from score functions to advantage estimations, provides a robust framework for learning in dynamic environments, as seen in examples like robotic arm control and multi-agent coordination. While policy gradient methods address some critical challenges, issues like sample inefficiency and instability persist, prompting future research into hybrid approaches that can improve efficiency and reliability. Overall, the versatility and adaptability of policy gradient methods underscore their importance in advancing RL applications across diverse fields

In our Lunar Lander project, the application of PPO and TRPO highlighted the strengths of policy gradient methods over traditional value-based approaches like DQN and REINFORCE. PPO demonstrated faster convergence and achieved cumulative rewards exceeding 200 consistently across most seeds after hyperparameter tuning. TRPO, on the other hand, offered greater stability with lower variance but at the cost of slower convergence. Compared to DQN, which struggles in continuous control problems due to its reliance on discretization, and REINFORCE, which exhibited high variance and slow convergence in the Lunar Lander environment, PPO and TRPO provided superior performance. The ability of PPO and TRPO to handle high-dimensional state spaces and dynamic environments underscores their effectiveness in solving complex control tasks.

Despite these successes, challenges and limitations remain. PPO's high sensitivity to hyperparameters necessitated extensive tuning to stabilize performance across seeds. Additionally, the computational cost of TRPO and the limited generalizability of hyperparameters to other environments highlight the trade-offs in using policy gradient methods. Exploration vs. exploitation balance and evaluation consistency also posed challenges, emphasizing the importance of robust algorithm design and parameter optimization. While these limitations suggest areas for improvement, the results of this project affirm the advantages of policy gradient methods in tackling dynamic reinforcement learning problems and provide a foundation for future work in extending these techniques to more complex environments.

IX. Future Work

Future work on policy gradient methods could focus on expanding their applicability to broader domains, such as healthcare robotics, resource optimization, and multi-agent systems, where dynamic and continuous decision-making is crucial (Lehmann 2024). For our project, we aim to implement and compare advanced policy gradient methods, including TRPO, PPO, and Actor-Critic to evaluate their performance in the LunarLander environment. The analysis will highlight how these methods address the limitations of stochastic policy gradients like REINFORCE and value-based methods like DQN, particularly in tasks requiring precise control.

In addition to their traditional applications, current research on policy gradient methods explores state-of-the-art techniques to address their limitations, such as sample inefficiency and instability. Techniques like **Meta Reinforcement Learning** aim to improve the generalizability of learned policies across tasks by enabling agents to adapt quickly to new environments. Furthermore, **Soft Actor-Critic (SAC)** and other entropy-regularized approaches continue to advance policy gradient optimization by balancing exploration and exploitation more effectively. Another promising direction is **Offline Reinforcement Learning**, which incorporates policy gradients into learning from static datasets without requiring active interaction with the environment, thereby reducing sample complexity. Finally, integrating policy gradient methods with advances in **deep representation learning** may enhance their ability to handle complex, high-dimensional input spaces, such as those found in visual tasks.

These emerging directions highlight the potential of policy gradient methods to overcome existing challenges and extend their applicability to a wider range of real-world problems. Continued research into hybrid approaches, such as combining value-based and policy gradient methods, may further enhance their performance and stability across diverse tasks.

X. Links

Here is a link to the presentation: Presentation Link
Here is a link to our code and results: Code Link

Team Contributions

Team Member	Project Contribution(%)	Extra Credit Contribution(%)
Varshitha Purra	33.33%	33.33%
Puneet Naru	33.33%	33.33%
Pranay Palem	33.33%	33.33%

TABLE I
Equal contribution distribution among team members for the project and extra credit implementation.

Our work focused on analyzing and describing REINFORCE, Actor-Critic, TRPO, and PPO algorithms. For the experimental component, we utilized the Lunar Lander environment from OpenAI Gym to systematically compare the performance of TRPO and PPO. Puneet Sai Naru was responsible for implementing and analyzing the PPO algorithm, Pranay Palem managed the TRPO implementation and result evaluation, and Varshitha Purra contributed to the theoretical foundation and comparative analysis of the algorithms. Through our combined efforts, we delivered a comprehensive exploration of both theoretical insights and practical applications.

PLEASE NOTE: The percentage contribution distribution for all team members has been included in the table above, indicating **equal** effort in both the project and the extra credit implementation.

References

Reference	Description
Navdeep Kumar et al. (2024). "Policy Gradient for Reinforcement Learning with General Utilities". In: *The Second Tiny Papers Track at ICLR 2024*. URL: https://openreview.net/forum?id=XsWA2y2TyI	*Policy Gradient for Reinforcement Learning with General Utilities* served as the primary theoretical foundation for the report, providing key insights for the **Introduction** and **Central Issue** sections.
Yue Wang and Shaofeng Zou (2022). "Policy Gradient Method For Robust Reinforcement Learning". In: *ICML*, pp. 23484–23526. URL: https://proceedings.mlr.press/v162/wang22at.html	*Policy Gradient Method for Robust Reinforcement Learning* contributes to the **How** and **Why** sections, offering a focus on stability in uncertain environments.
Simon Ståhlberg, Blai Bonet, and Hector Geffner (Aug. 2023). "Learning General Policies with Policy Gradient Methods". In: *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 647–657. DOI: 10.24963/kr.2023/63	*Learning General Policies with Policy Gradient Methods* provides application examples, particularly in robotics and strategy games, referenced in the **What** and **Future Work** sections.
Matthias Lehmann (2024). *The Definitive Guide to Policy Gradients in Deep Reinforcement Learning: Theory, Algorithms and Implementations*. DOI: 10.48550/arXiv.2401.13662	*The Definitive Guide to Policy Gradients in Deep Reinforcement Learning* offers a comprehensive overview of policy gradients, assisting in the **Introduction**, **How**, and **Future Work** sections by discussing advanced methods and potential improvements.
John Schulman et al. (2017). "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347*. DOI: 10.48550/arXiv.1707.06347	*Proximal Policy Optimization Algorithms* discusses a new family of policy gradient methods. We use this paper to implement our intended project.
Rohit Sadavarte, Rishab Raj, and B Sathish (2021). "Solving the Lunar Lander Problem using Reinforcement Learning". In: *2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, pp. 1–6. DOI: 10.1109/CSITSS54238.2021.9682970	*Solving the Lunar Lander Problem using Reinforcement Learning* compares DQN and REINFORCE on the Lunar Lander problem in OpenAI Gym. We use this paper to implement our intended project.