

Learning Optimal Lunar Landings: A PPO-Clip with Beta Distribution Approach

Erfan Mohagheghian

emohagheghian3@gatech.edu

Department of Computer Sciences

Georgia Institute of Technology

Abstract

In this project, I explore the application of Proximal Policy Optimization (PPO-Clip) with a Beta distribution [Ref.1] to train an agent in the Lunar Lander v3 environment with a continuous action space. Traditional reinforcement learning algorithms often struggle with continuous action spaces due to the complexity of exploration and policy representation. By leveraging a Beta distribution, my policy network can effectively model bounded action spaces, improving stability and performance. Compared to Deep Deterministic Policy Gradient (DDPG), PPO-Clip offers a more stable training process by avoiding the overestimation bias associated with Q-learning-based methods. My approach integrates deep reinforcement learning techniques with function approximation using a neural network. The results demonstrate the efficacy of PPO-Clip in learning optimal landing strategies, highlighting the importance of proper action distribution modeling in reinforcement learning.

1 Introduction

Reinforcement learning (RL) has shown significant advancements in solving complex control tasks [Ref.4]. The Lunar Lander v3 environment presents a challenging problem requiring precise thrust control for a soft landing. Traditional discrete-action RL methods, such as Deep Q-Networks (DQN), are unsuitable for continuous control tasks. While Deep Deterministic Policy Gradient (DDPG) [Ref.2] is a widely used approach for continuous action spaces, it suffers from instability due to reliance on deterministic policies and target Q-network updates. Proximal Policy Optimization (PPO) [Ref.3] provides a more stable alternative by using clipped objective functions to prevent large policy updates. In this study, I implement PPO-Clip with a Beta distribution to improve the policy's ability to model bounded continuous actions. I detail the neural network architecture, reward shaping, and training process, emphasizing improvements over standard Gaussian-based policies and the advantages over DDPG.

2 Background and Related Work

2.1 Lunar Lander Simulation

Lunar Lander is a reinforcement learning simulation where an agent controls a spacecraft attempting a safe landing on the lunar surface. This section describes the simulation environment and key components used to train the landing agent.

- **State Space \mathbf{S} :** The lander's state is defined by an 8-dimensional vector including its x-y position, velocities, orientation (angle), angular velocity, and leg contact sensors. This comprehensive representation provides continuous feedback on the spacecraft's spatial configuration relative to the landing pad.

- **Action Space A:** The action space is continuous and consists of two dimensions: throttle control (ranging from 0 to 1) and directional thrust (ranging from -1 to 1). These actions enable the agent to finely adjust the lander’s descent and orientation during the landing phase.
- **Challenges:**
 - **Precision Control:** The lander must apply exact thrust adjustments to ensure a soft landing without overshooting or crashing.
 - **Dynamic Environment:** The simulation includes realistic physics such as gravity, momentum, and varying terrain, which add complexity to maintaining stability during descent.
 - **Continuous Action Representation:** Modeling a bounded continuous action space requires careful policy design to prevent outputs from exceeding the valid range, ensuring safe and effective control.

3 Methods: PPO-Clip with Beta Distribution

3.1 Network Architectures

My implementation uses two separate neural networks for the policy and value functions:

Policy Network (PolicyNet): The policy network takes an 8-dimensional state vector as input and passes it through two fully connected layers (each with 150 neurons) using `tanh` activations. Instead of a Gaussian policy, the network outputs parameters for a Beta distribution via two separate heads — one for the α parameter and one for the β parameter. Both heads apply a softplus activation (with an offset of +1.0) to ensure that the parameters remain greater than one, which is important for shaping the Beta distribution. This design allows me to directly sample actions within the $[0, 1]$ interval.

Value Network (ValueNet): The value network uses a similar architecture with two hidden layers (each with 150 neurons and `tanh` activations) and outputs a scalar estimate of the state value. This value estimate is later used for computing the advantage function during training.

Weights for both networks are initialized using Xavier (Glorot) normalization to promote stable learning.

3.2 PPO Agent and Loss Computation

The `PPOAgent` class encapsulates the policy and value networks along with their respective Adam optimizers and learning rate schedulers (using `StepLR`). The agent is responsible for computing losses and updating network parameters as follows:

Policy Loss: For each state, the policy network produces the α and β parameters that define a Beta distribution. I sample actions from this distribution and compute their log probabilities. The policy loss is derived using the PPO-Clip objective:

$$L(\theta) = -\mathbb{E}[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)] - \text{entropy bonus}, \quad (1)$$

where the probability ratio $r_t(\theta)$ is computed as the exponentiation of the difference between new and old log probabilities. An entropy bonus (weighted by a decaying coefficient) is subtracted to encourage exploration.

Value Loss: The value network is updated using a mean squared error (MSE) loss between its state value predictions and the computed returns.

3.3 Training Procedure

I collected experience into a **Buffer** object that stores states, actions, rewards, estimated values, log probabilities, and done flags. After each episode:

Advantage Estimation: Generalized Advantage Estimation (GAE) was used to compute the advantages, leveraging both the immediate rewards and the value estimates. The advantage at time step t is computed as

$$A_t = \sum_{l=0}^{T-t-1} (\gamma\lambda)^l \delta_{t+l}, \quad (2)$$

where the temporal-difference error is given by

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t). \quad (3)$$

The computed advantages are then normalized to stabilize the learning process.

Mini-Batch Updates: The **PPOAgent** performs multiple epochs over the collected batch. For each epoch, the training data is shuffled and partitioned into mini-batches (e.g., of size 64). Within each mini-batch, both the policy and value networks were updated using the previously computed losses. Learning rate schedulers adjust the learning rates periodically.

Checkpointing: checkpoints at regular intervals were saved to ensure that training progress is preserved and can be resumed if necessary.

3.4 Action Sampling and Transformation

Actions are sampled from the Beta distribution produced by the policy network. Since the Beta distribution outputs values in the range $[0, 1]$, I apply a linear transformation to map these values into the environment’s continuous action space of $[-1, 1]$:

$$\text{Action} = 2 \times (\text{clamp}(a, 0, 1) - 0.5) \quad (4)$$

This transformation ensures that the agent’s actions remain within the valid bounds required by the Lunar Lander v3 environment.

3.5 Implementation Details

- **Optimization:** Both the policy and value networks were optimized using the Adam optimizer with an initial learning rate of 3×10^{-4} . **StepLR** schedulers were used to decay the learning rate periodically, fine-tuning the training process.
- **Entropy Coefficient Decay:** The coefficient for the entropy bonus decays over time, gradually shifting the balance from exploration towards exploitation as training progresses.
- **Environment Interaction:** The agent interacts with the Lunar Lander v3 environment using episodic rollouts. In each episode, actions were sampled, transformed, and applied to the environment. The resulting experiences were stored in the buffer and used for subsequent updates.

This approach integrates deep reinforcement learning with function approximation and carefully models the continuous action space via a Beta distribution, addressing the challenges associated with exploration and stability. Compared to methods like DDPG, the clipped PPO objective and explicit bounded action modeling provide a robust and stable training process, leading to improved performance in achieving optimal landing strategies.

4 Experimental Setup

4.1 Hyperparameter Tuning

In this project, various hyperparameters were explored to optimize the PPO-Clip algorithm for the Lunar Lander task. The following hyperparameters were tested:

- **Learning Rate (lr):** {0.03, 0.003, 0.0003}. Different values were tested to find the optimal learning rate for stable and efficient training.
- **Clip Parameter:** {0, 0.2, 0.5}. Controls the policy deviation during updates, with higher values limiting changes.
- **Entropy Coefficient:** {0, 0.1, 0.5}. Encourages exploration; tested values aim to balance exploration and exploitation.
- **Discount Factor (gamma):** {0, 0.99}. Affects the importance of future rewards, with 0 focusing on immediate rewards and 0.99 considering long-term benefits.
- **Lambda (λ):** 0.95. This value for Generalized Advantage Estimation (GAE) controls the bias-variance trade-off in advantage calculation.
- **Entropy Decay:** 0.99. Gradually reduces exploration to promote more exploitation as the agent becomes confident.

The goal was to identify the best combination of these hyperparameters to achieve efficient learning and performance on the task.

5 Results and Analysis

5.1 Approach and Policy Distribution

PPO-Clip algorithm was chosen because its clipped objective prevented overly large policy updates, resulting in more stable training compared to standard policy gradient methods. Instead of using a Gaussian distribution for the stochastic policy, the Beta distribution was opted. The Gaussian distribution had infinite support, which was problematic in real-world scenarios with naturally bounded actions and could introduce estimation bias. In the first tries, Gaussian distribution was implemented but the result was unsuccessful due to lack of control on the standard deviation and some action were out of the band despite having the mean value in the band. In contrast, the Beta distribution had finite support, confining actions within the valid range. Recent research supported this choice, demonstrating that a Beta policy yielded superior final rewards, increased stability, and faster convergence—advantages that were critical for the Lunar Lander task.

5.2 Hyperparameter Tuning

To optimize performance, I experimented with several key hyperparameters:

- **Learning Rate (lr):** {0.03, 0.003, 0.0003} Default:0.0003
A high learning rate (0.03) led to instability, while a very low learning rate (0.0003) slowed convergence. Interestingly, in this case, low learning rate of 0.0003 showed faster convergence than the intermediate learning rate value of 0.003.
- **Clip Parameter:** {0, 0.2, 0.5} Default:0.2
A clip value of 0 offered no restriction and resulted in volatile updates, whereas 0.5 overly constrained learning. 0.2 was found to limit drastic changes while still permitting efficient updates.

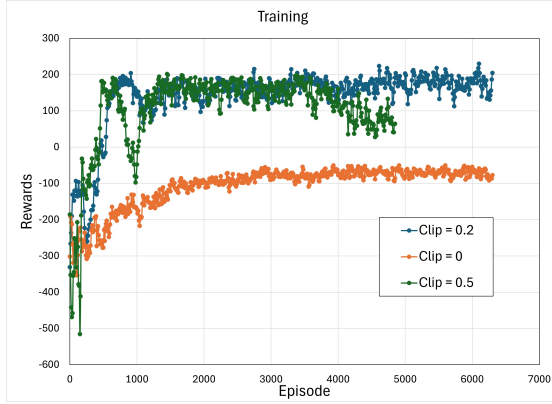


Figure 1: Effect of different clipping values on policy updates in PPO.



Figure 2: Effect of different learning rates on policy updates in PPO.

- **Entropy Coefficient:** {0, 0.1, 0.5} Default:0.0
A higher entropy coefficient (0.5) induced excessive exploration and fluctuation. The value of 0 showed a better performance compared to the value of 0.1 indicating the exploration by sampling from Beta distribution was adequate enough.
- **Discount Factor (gamma):** {0, 0.99} Default:0.99
A gamma of 0 made the agent focus solely on immediate rewards, while 0.99 encouraged long-term planning—an essential requirement for the Lunar Lander task.

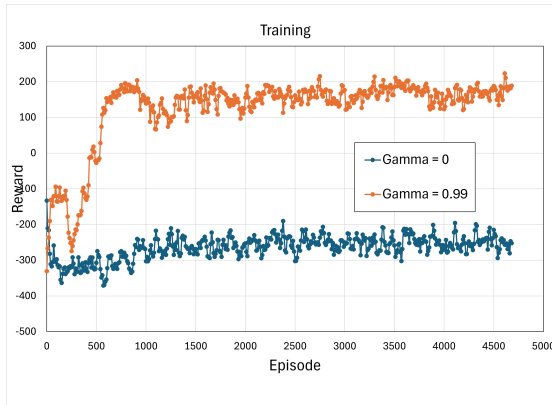


Figure 3: Effect of different Gamma values on policy updates in PPO.

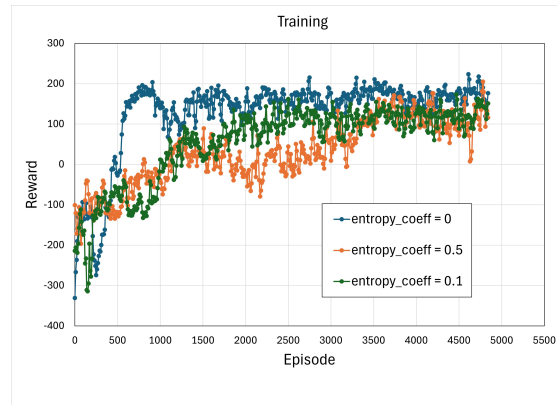


Figure 4: Effect of different entropy coefficient on policy updates in PPO.

- **Lambda (λ) for GAE:** 0.95
This value balanced the bias-variance trade-off in the advantage computation, contributing to smoother training.
- **Entropy Decay:** 0.99
I used this decay rate to gradually reduce exploration as training progressed, allowing the agent to shift focus toward exploitation once it became more confident.

The optimized model with tuned hyper-parameters was trained for 10K episodes and the reward of the trained agent for 100 episodes were plotted. The result showed the final reward of close to 200 indicating the proper landing of the spacecraft on the designated landing area



Figure 5: Reward evaluation of the PPO agent with respect to training episode

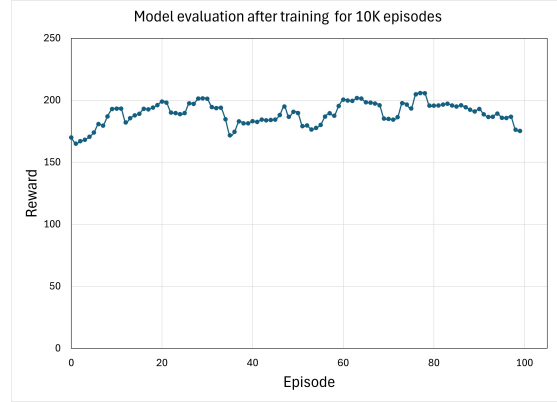


Figure 6: Reward for 100 episodes for trained agent

5.3 Challenges and Future Work

During experimentation, I encountered several challenges:

- **Instability in Early Training:** High variance in initial rewards required careful tuning of the learning rate and entropy coefficient.
- **Balancing Exploration and Exploitation:** Excessive exploration hindered convergence, while insufficient exploration led to premature convergence on suboptimal policies.
- **Hyperparameter Sensitivity:** Small changes in hyperparameter settings significantly impacted performance, necessitating extensive experimentation.

Given more time, I would have explored adaptive entropy regularization, learning rate annealing, and alternative network architectures (e.g., recurrent networks) to better handle partial observability and further enhance performance.

Overall, these design choices and hyperparameter settings were driven by both theoretical insights and empirical evidence, ensuring robust performance on the Lunar Lander task while addressing the inherent challenges of continuous control.

6 Discussion and Conclusion

I implemented the PPO-Clip algorithm with a Beta policy for the Lunar Lander environment, addressing the limitations of Gaussian policies in bounded action spaces. The Beta distribution ensured that actions remained within valid limits, reducing estimation bias and improving stability. Compared to the Gaussian policy, the Beta policy led to more stable training, faster convergence, and better final performance. PPO-Clip further contributed to stability by constraining policy updates, preventing abrupt changes that could hinder learning.

Hyperparameter tuning played a role in optimizing training efficiency. Learning rate, clipping parameter, entropy coefficient, and discount factor were systematically varied to balance stability and performance. The implementation of the Beta policy was notably easier for bounded action spaces compared to the Gaussian policy. With the same workflow, I was unable to train the model effectively using a Gaussian distribution.

Future work could involve adaptive entropy tuning, alternative policy update mechanisms, and extending the approach to more complex continuous control tasks. Investigating how Beta policies generalize across different reinforcement learning environments could further validate their effectiveness.

7 References

1. Petrazzini, I. G. B., & Antonelo, E. A. (2021). Proximal Policy Optimization with Continuous Bounded Action Space via the Beta Distribution. *arXiv preprint arXiv:2111.02202*. <https://arxiv.org/abs/2111.02202>
2. Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*. <https://arxiv.org/abs/1509.02971>
3. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347*. <https://arxiv.org/abs/1707.06347>
4. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). MIT Press.