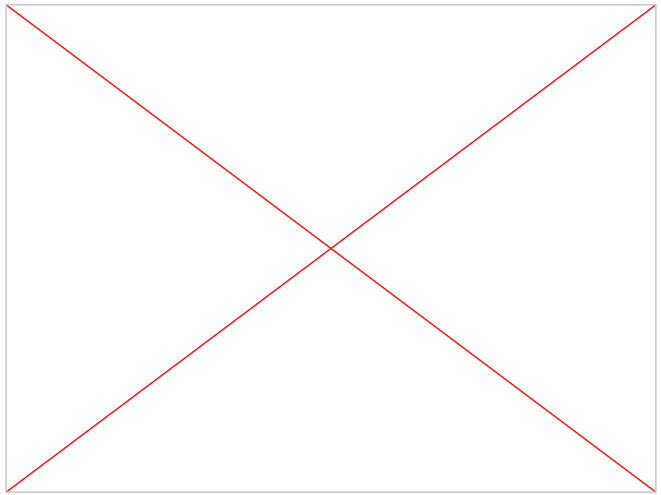
Reinforcement Learning for Lunar Lander using PPO

Aditya Raghuvanshi
Mallika Garg
Yash Bhaskar

Open AI GYM & Lunar Lander

The **Lunar Lander** environment is a classic control problem in OpenAI Gym.

Objective: Land the spacecraft safely on the designated landing pad while optimizing fuel consumption.



Lunar Lander Environment

Key Characteristics:

- **Fully Observable:** All state information is available at every frame.
- **Single-Agent:** No competition or cooperation.
- **Deterministic:** No randomness in action effects or rewards.
- **Episodic:** Reward depends only on the current state and action.
- **Discrete Action Space:**
  - **Thrust** (Up)
  - **Left** (Rotate left)
  - **Right** (Rotate right)
  - **Do Nothing**
- **Finite Horizon:**
  - The episode ends on **successful landing, crash, or 1000 steps**.

Observation Space (8 Features)

- **Continuous Variables:**
  - X Distance from Target
  - Y Distance from Target
  - X Velocity
  - Y Velocity
  - Angle of Ship
  - Angular Velocity of Ship
- **Binary Variables:**
  - Left Leg Grounded (0 or 1)
  - Right Leg Grounded (0 or 1)

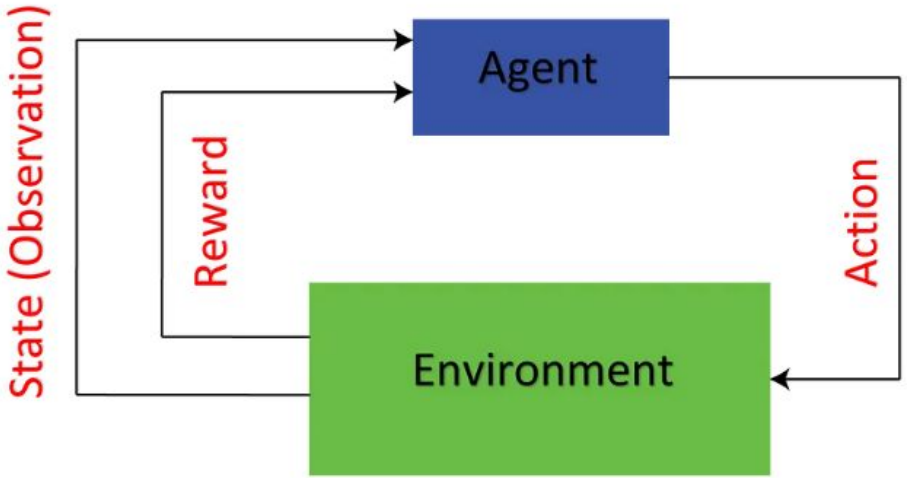
Rewards Breakdown

- **Landing Pad**: +100 to +140 points
- **Crash**: -100 points
- **Leg Contact**: +10 per leg
- **Engine Use**:
  - Main engine: -0.3 per frame
  - Side engines: -0.03 per frame
- **Solved Score**: 200 points

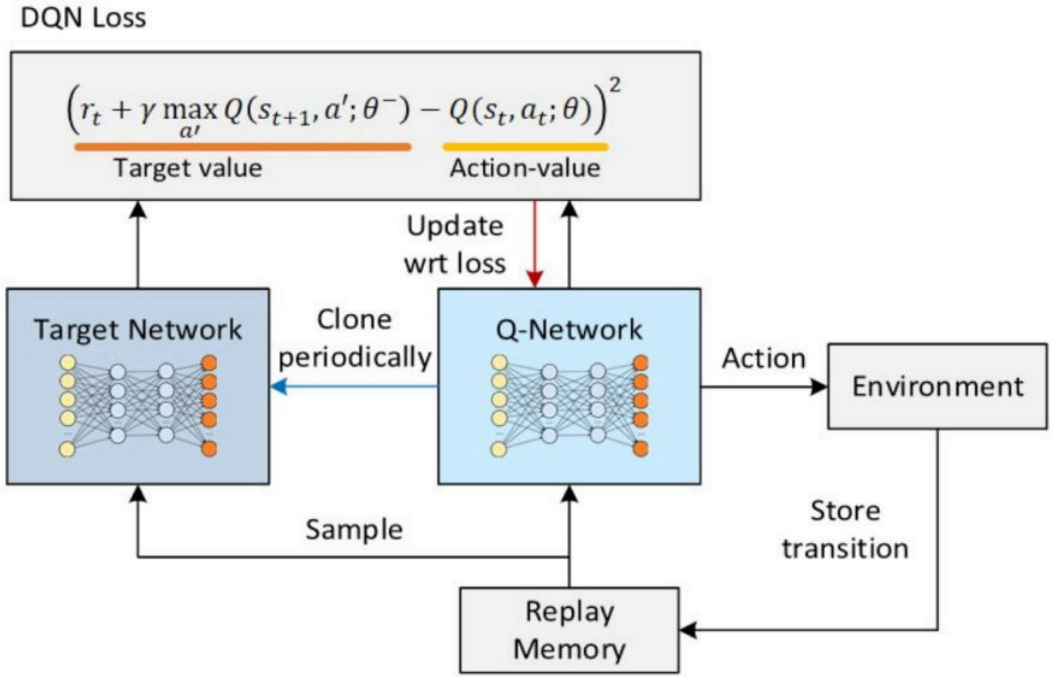
DQN, A2C and PPO

DQN

DQN (Deep Q-Network) is a reinforcement learning algorithm where the agent learns to take actions in an environment to maximize cumulative rewards by estimating Q-values using a neural network.



Agent -> neural network



Target Network: Stability reference

Q-Network: Policy learning

Environment : State generation

Replay Memory: Experience storage

Action: Environment interaction

Sample: Avoid correlations

A2C : Actor Critic Algorithm

Actor-Critic Algorithm is a type of reinforcement learning algorithm that combines aspects of :

1. Policy-based methods (**Actor**)
2. Value-based methods (**Critic**)

In the actor-critic framework, an agent (the "actor") learns a policy to make decisions, and a value function (the "Critic") evaluates the actions taken by the Actor.

Roles of Actor and Critic

Actor: The actor makes decisions by selecting actions based on the current policy. Its responsibility lies in exploring the action space to maximize expected cumulative rewards. By continuously refining the policy, the actor adapts to the dynamic nature of the environment.

The policy, denoted as $\pi(a|s)$, represents the probability of taking action a in state s.

Critic: The critic evaluates the actions taken by the actor. It estimates the value or quality of these actions by providing feedback on their performance. The critic's role is pivotal in guiding the actor towards actions that lead to higher expected returns, contributing to the overall improvement of the learning process.

The value function, denoted as V(s), estimates the expected cumulative reward starting from state s.

How does it A2C works ?

The objective function for the Actor-Critic algorithm is a combination of the policy gradient (for the actor) and the value function (for the critic).

Policy Gradient (Actor)

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=0}^{N} \nabla_\theta \log \pi_\theta(a_i|s_i) \cdot A(s_i, a_i)$$

Here,

- $J(\theta)$ represents the expected return under the policy parameterized by θ
- $\pi_\theta(a \mid s)$ is the policy function
- N is the number of sampled experiences.
- $A(s, a)$ is the advantage function representing the advantage of taking action **a** in state **s**.
- *i* represents the index of the sample

Value Function Update (Critic)

$$\nabla_w J(w) \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_w (V_w(s_i) - Q_w(s_i, a_i))^2$$

Here,

- $\nabla_w J(w)$ is the gradient of the loss function with respect to the critic's parameters **w**.
- N is number of samples
- $V_w(s_i)$ is the critic's estimate of value of state **s** with parameter **w**
- $Q_w(s_i, a_i)$ is the critic's estimate of the action-value of taking action **a**
- *i* represents the index of the sample

Update Rules

The update rules for the actor and critic involve adjusting their respective parameters using gradient ascent (for the actor) and gradient descent (for the critic).

Actor Update

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\theta_t)$$

Here,

- α: learning rate for the actor
- t is the time step within an episode

Critic Update

$$w_t = w_t - \beta \nabla_w J(w_t)$$

Here,

- w represents the parameters of the critic network
- β is the learning rate for the critic

Advantage function

The advantage function, A(s,a), measures the advantage of taking action a in state s over the expected value of the state under the current policy.

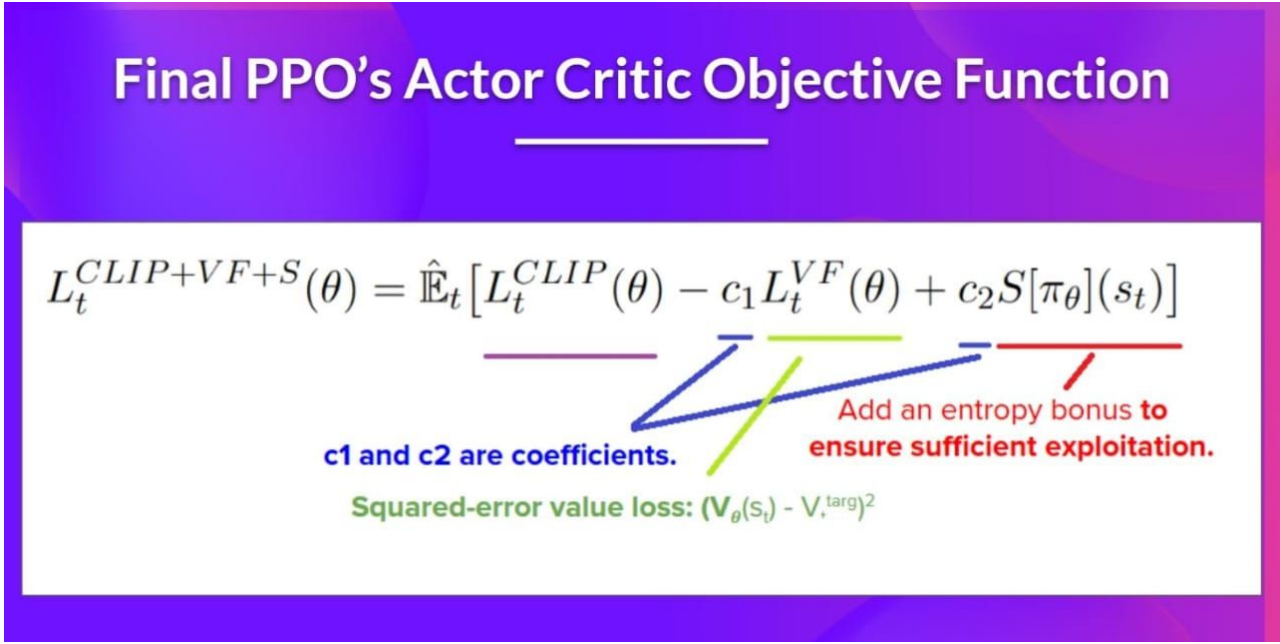
$$A(s,a) = Q(s,a) - V(s)$$

The advantage function, then, provides a measure of how much better or worse an action is compared to the average action.

PPO

Benefits of PPO:

✔ Works for **both discrete & continuous** action spaces.
✔ **Stable training** with clipped policy updates.
✔ **Data-efficient** and simple to implement.



Clipped Surrogate Objective

$$L_{\text{policy}}(\theta) = E\left[\min\left(r_\theta(s)A_t, \text{clip}(r_\theta(s), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

$$r_\theta(s) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$$

The clipping prevents drastic policy updates, improving stability.

If the ratio gets too high, updates are clipped to prevent instability.

$p_t(\theta)\ > 0$	A_t	**Return Value of** min	**Objective is Clipped**	**Sign of Objective**	**Gradient**
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

Source : https://fse.studenttheses.ub.rug.nl/25709/1/mAI_2021_BickD.pdf

Value Critic & Entropy Bonus

Value Critic Loss: Helps stabilize training by providing a better reward estimate.

Entropy Bonus: Encourages exploration early in training.

Over time, **entropy decreases** as the policy converges to a deterministic strategy.

- Value Loss (Squared Error Loss):

$$L_{\text{value}}(\theta) = E\left[(V_\theta(s_t) - R_t)^2\right]$$

- Entropy Bonus:

$$L_{\text{entropy}}(\theta) = -E[H(\pi_\theta)]$$

Targets

- Train an RL agent to achieve stable landings in the Lunar Lander environment.
- Optimize training stability and efficiency using PPO (Proximal Policy Optimization).
- Compare performance against other RL algorithms like A2C and DQN.
- Evaluate model
  - Reward
  - Landing accuracy
  - Fuel efficiency.

Timeline

Week 1:

- Set up Lunar Lander environment.
- Baseline PPO training & hyperparameter tuning.

Week 2:

- Train PPO & compare with A2C and DQN.
- Analyze convergence & stability.

Week 3:

- Optimize PPO (entropy bonus, reward shaping).
- Improve landing accuracy & fuel efficiency.

Week 4:

- Final evaluation & visualization.
- Prepare slides & report.