Comparative Analysis of Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) Algorithms

Kaushik Deo, Hemmanahalli Manjunath Ruthvik

Northeastern University
Boston, MA

Abstract

This project aims to conduct a comparative study of two algorithms: Soft Actor-Critic(SAC) and Proximal Policy Optimization(PPO) across three distinct domains in the OpenAI Gym framework: Taxi-v3, CartPole-v1, and LunarLander-v2. These environments were chosen to represent a range of challenges, including grid-based navigation, real-time balancing, and precision landing, which test the algorithms' capabilities in discrete and continuous action spaces. The primary objective is to evaluate SAC and PPO in terms of their ability to maximize cumulative rewards efficiently while maintaining training stability and computational feasibility. By analyzing performance metrics such as cumulative rewards, training time, and stability, this study seeks to highlight the strengths and limitations of algorithms in various RL domains.

Introduction

Reinforcement learning (RL) has become a cornerstone of modern artificial intelligence, enabling agents to learn optimal strategies for decision-making in complex and dynamic environments. The versatility of RL algorithms has led to breakthroughs in various fields, from robotics to gaming and autonomous systems. However, the effectiveness of an RL algorithm often depends on its adaptability to the specific characteristics of the task at hand, such as the complexity of the environment, the action space, and the nature of the rewards. Although many RL algorithms excel in specific tasks, few have been extensively analyzed for their ability to generalize across multiple environments with varying requirements.

Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) are two popular RL algorithms with distinct strengths. SAC is an off-policy algorithm designed for continuous control tasks that uses an entropy term to encourage exploration. PPO, on the other hand, is an on-policy algorithm that controls the magnitude of policy updates to ensure stability and reliability in both discrete and continuous domains. Despite their differences, both algorithms have shown promising results in solving RL problems.

In real-world applications, RL systems often have to deal with a variety of challenges, such as discrete decision making, real-time balancing, and precision control. This calls for a thorough examination of how algorithms like SAC and PPO perform across tasks with different action spaces, state representations, and reward structures. Furthermore, while SAC's exploration mechanisms and the stability of PPO have been evaluated individually, there is limited research directly comparing their performance in environments with varying complexities. Addressing this gap can provide valuable insights into the strengths and trade-offs of these algorithms.

This project seeks to evaluate SAC and PPO across three OpenAI Gym environments—Taxi-v3, CartPole-v1, and LunarLander-v2—representing discrete, continuous, and hybrid action spaces, respectively. These environments test the algorithms on a range of tasks, including grid-based navigation, dynamic balancing, and goal-oriented control.

To tackle the problem, this study adopts the following approach:

- Develop and train agents using SAC and PPO in each of the three environments.
- Measure and compare the algorithms' performance based on key metrics such as cumulative rewards, training time, stability, and adaptability.
- Investigate algorithm-specific behaviors, such as SAC's balance between exploration and exploitation and PPO's ability to maintain stability during training.

The objectives of this project are:

- To analyze the performance of SAC and PPO across discrete, continuous, and hybrid environments.
- To identify the specific strengths and weaknesses of each algorithm in handling navigation, balancing, and control tasks.
- To provide actionable insights for selecting RL algorithms based on task requirements and environment characteristics.

Environments

The OpenAI Gym framework provides a suite of simulated environments that are commonly used for benchmarking reinforcement learning (RL) algorithms. For this project, we utilize three distinct environments—Taxi-v3, CartPole-v1, and LunarLander-v2—each representing unique challenges in terms of state-action spaces and task objectives.

CartPole-v1

- A pole is connected to a cart via a passive joint, allowing it to move freely. The cart moves along a frictionless track, with the pole initially positioned upright. The objective is to balance the pole by applying forces to the cart, either to the left or right.
- **Actions** are `ndarray` with shape `(1,)` that can take values `{0, 1}` representing the direction of the force the cart is pushed with.
  - 0: Push the cart to left
  - 1: Push the cart to right
- **Observations** are `ndarray` with shape `(4,)` with the values corresponding to the following positions and velocities:

Observation	Min	Max
Cart Position	-4.8	4.8
Cart Velocity	-Inf	Inf
Pole Angle	-0.418 rad (-24°)	-0.418 rad (24°)
Pole Angular Velocity	-Inf	Inf

Table 1: CartPole-v1 Observation Space

- x-position (0) can be take values between `(-4.8, 4.8)`, but the episode ends if the cart leaves the `(-2.4, 2.4)` range.
- The pole angle is observed between `(-0.418, 0.418)` radians (or ±24°), but the episode ends if the pole angle is not in the range `(-0.2095, 0.2095)` (or ±12°)
- A **reward** `+1` is given for every step taken, including the termination step. The default reward limit is 500 due to the time limit on the environment.
- The **episode ends** if any one of the following occurs:
  - Pole Angle is greater than ±12°
  - Cart Position is greater than ±2.4 (center of the cart reaches the edge of the display)
  - Episode length is greater than 500

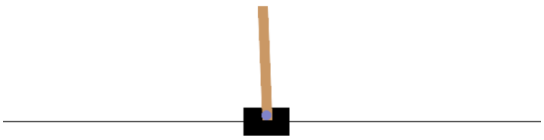

Figure 1: CartPole-v1 Environment

LunarLander-v2

- This is a classic rocket trajectory optimization problem.
- There are four discrete **actions** available:
  - 0: do nothing
  - 1: fire left orientation engine
  - 2: fire main engine
  - 3: fire right orientation engine
- The **observation** is an 8-dimensional vector: the coordinates of the lander in `x` & `y`, its linear velocities in `x` & `y`, its angle, its angular velocity, and two booleans that represent whether each leg is in contact with the ground or not.
- For each step, the:
  - Reward increases or decreases linearly based on proximity of lander to the landing pad.
  - Reward increases or decreases based on the magnitude of lander's velocity, rewarding slower movements.
  - Reward decreases linearly based on the absolute value of the lander's tilt .
  - 10 points for each leg of the lander in contact with the ground.
  - A penalty of 0.03 points for each frame in which a side engine is firing.
  - A penalty of 0.3 points for each frame in which the main engine is firing.
  - An extra reward of -100 points for crashing and +100 points for landing safely is given at the end of the episode. The episode is considered solved when the total score reaches at least 200 points.
- The **episode ends** if:
  - the lander crashes (the lander body gets in contact with the moon)
  - the lander gets outside of the viewport (`x` coordinate is greater than 1)

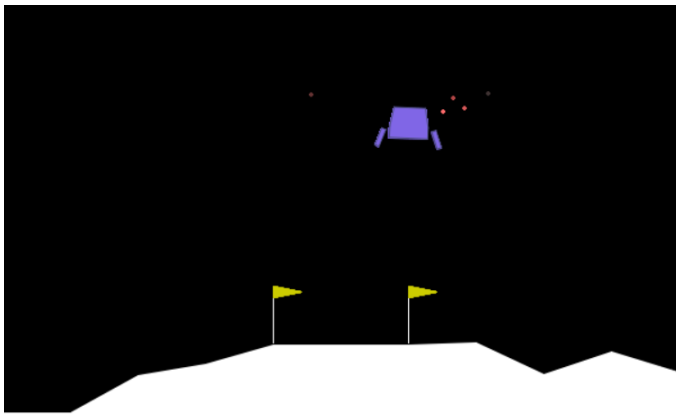

Figure 2: LunarLander-v2 Environment

Taxi-v3

- In the 5x5 grid world, there are four specific locations for picking up and dropping off: Red, Green, Yellow, and Blue. The taxi can be in any one of these places at the start, and similarly, the passenger can be in any of the pickup locations.

  In each episode, the goal is to steer the taxi to the passenger, to then pick them up, transport to their destination and drop-off in this location. An episode finishes itself once the passenger has finally gotten off at the correct end. An outline that involves negative and positive rewards is necessary:

  Of course rewarded steps will be dropping off at correct positions while incorrect pickup or dropping attempt in each single taken steps yields an assigned but more strongly "punitive" negative payoffs.
- **Action** shape is `(1,)` in the range `{0, 5}` showing which direction to move the taxi or to pickup/drop off passengers.
  - 0: Down
  - 1: Up
  - 2: Right
  - 3: Left
  - 4: Pickup
  - 5: Drop off
- The environment consists of 500 discrete \textbf{states}, derived from the 25 possible taxi positions, 5 passenger locations, and 4 destination options. The destinations on the map are identified by the first letter of their respective colors.
  Locations:
  - 0: Red
  - 1: Green
  - 2: Yellow
  - 3: Blue
  - 4: In Taxi

  Destinations:
  - 0: Red
  - 1: Green
  - 2: Yellow
  - 3: Blue
- An observation is returned as `int()` that encodes the corresponding state, calculated by ((taxi_row * 5 + taxi_col) * 5 + passenger_location) * 4 + destination.
- **Rewards**
  - -1 unless other reward is triggered.
  - +20 delivering passenger.
  - -10 Illegaly executing "pickup" and "drop-off" actions.
- The **episode ends** if :
  - The taxi drops off the passenger.
  - The length of the episode is 200.

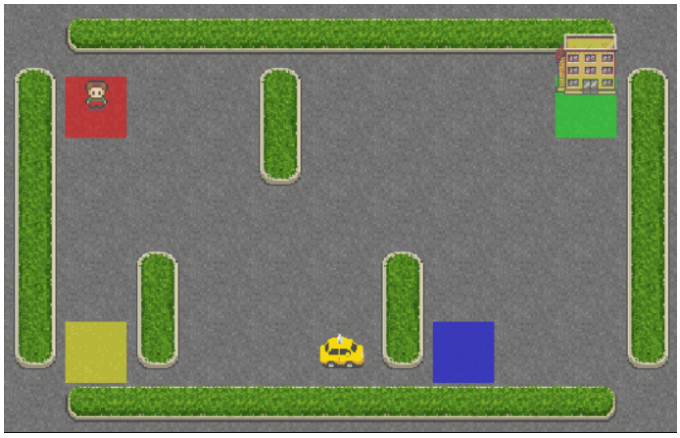

Figure 3: Taxi-v3 Environment

Algorithms

Soft Actor-Critic (SAC)

The Soft Actor-Critic (SAC) algorithm is an advanced reinforcement learning (RL) method that combines the advantages of actor-critic frameworks with entropy maximization for efficient learning in continuous action spaces. It is an *off-policy* algorithm, meaning it uses experiences from a replay buffer rather than requiring new data from the current policy at every step.

Algorithm Explanation

1. **Initialization:**
   - Initialize policy parameters θ, Q-function parameters ϕ_1, ϕ_2, and target Q-function parameters $\phi_{\text{targ},1}$, $\phi_{\text{targ},2}$.
   - Set replay buffer D to store transitions.
2. **Environment Interaction:**
   - Observe the current state s and select an action $a \sim \pi_\theta(\cdot|s)$.
   - Execute the action a in the environment and observe the next state s', reward r, and done signal d.
3. **Replay Buffer:**
   - Store the transition (s, a, r, s', d) in the replay buffer D.
4. **Updating (if required):**
   - **Q-function updates:**
     - Compute the target value:

$$y(r, s', d) = r + \gamma(1-d)(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a') - \alpha \log \pi_\theta(a'|s'))$$

$$, \quad a' \sim \pi_\theta(\cdot|s').$$

     - Update both Q-functions Q_{ϕ_1} and Q_{ϕ_2} by minimizing the squared error:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2, \text{for } i = 1, 2.$$

   - **Policy update:**

– Adjust the policy π_θ using the re-parameterization trick to maximize:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \right),$$

where $\tilde{a}_\theta(s)$ is sampled from $\pi_\theta(\cdot|s)$.

- **Target network update:**

– Slowly update the target Q-function parameters using a soft update rule:

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i, \quad \text{for } i = 1, 2.$$

5. **Repeat Until Convergence:**
   - Continue collecting data and updating the networks until the policy converges.

Key Features of SAC

- **Entropy Regularization:** SAC encourages exploration by maximizing not only the expected reward but also the entropy of the policy. Higher entropy means more randomness in the agent's actions, which avoids premature convergence to suboptimal policies.
- **Stochastic Policies:** The policy outputs a probability distribution over actions, enabling better exploration.
- **Two Q-Networks:** SAC employs two Q-value networks to mitigate overestimation bias, a common issue in RL algorithms.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy reinforcement learning algorithm designed to improve upon traditional policy gradient methods by addressing issues of stability and efficiency. The PPO-Clip variant ensures stable updates by restricting the size of policy changes, which prevents the agent from taking excessively large policy updates that might degrade performance.

Algorithm Explanation

1. **Initialization:**
   - Start with initial policy parameters θ_0 and value function parameters ϕ_0.
2. **Collect Trajectories:**
   - Run the current policy $\pi_k = \pi(\theta_k)$ in the environment to collect a batch of trajectories $\mathcal{D}_k = \{\tau_i\}$, where each trajectory τ is a sequence of (s_t, a_t, r_t).
3. **Compute Rewards-to-Go:**
   - For each trajectory, calculate the cumulative discounted rewards from time t onward:

$$\hat{R}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

4. **Compute Advantage Estimates:**
   - Estimate the advantage function $\hat{A}_t$, which measures how much better an action is compared to the average expected value for a state. Common methods include Generalized Advantage Estimation (GAE).

Algorithm 1: Soft Actor-Critic

(5)

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters: $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  if s' is terminal **then**
    Reset environment state
  end if
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets for the Q-functions:

$$y(r, s', d) = r + \gamma(1-d)$$

$$\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

      Update Q-functions by one step of gradient descent using:

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2$$

for i = 1, 2

      Update policy by one step of gradient ascent using:

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \right)$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable with respect to θ via the reparameterization trick.
      Update target networks with:

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1-\rho)\phi_i \quad \text{for } i = 1, 2$$

    end for
  end if
until convergence

5. **Policy Update:**
   - Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \hat{A}_t,$$

$$g(\epsilon, \hat{A}_t)),$$

   where $g(\epsilon, \hat{A}_t)$ is the clipped advantage function:

$$g(\epsilon, \hat{A}_t) = \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t.$$

6. **Value Function Update:**
   - Fit the value function by minimizing the mean-squared error between the predicted value $V_\phi(s_t)$ and the rewards-to-go:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2.$$

7. **Repeat:**
   - Continue collecting trajectories and updating θ and ϕ until convergence.

Algorithm 2: PPO-Clip

(4)

1: **Input:** initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
6: Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}\right.$$

$$\left.\frac{\pi_{\theta_k}(s_t,a_t)}{t}, g(\epsilon, \hat{A}_t^{\pi_{\theta_k}(s_t,a_t)})\right),$$

typically via stochastic gradient ascent with Adam.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2,$$

typically via some gradient descent algorithm.
8: **end for**

Key Features of PPO-Clip

- **Clipping Mechanism:** The core idea is to limit the update to the policy by clipping the probability ratio between the new and old policy. This ensures the update stays within a predefined trust region.
- **Stability and Simplicity:** Unlike Trust Region Policy Optimization (TRPO), PPO avoids complex second-order optimization steps, making it simpler and computationally efficient.
- **Suitability Across Tasks:** PPO works well in both discrete and continuous action spaces, making it versatile for a variety of environments.

Experiment 1

Soft Actor-Critic Algorithm on CartPole-v1 Environment

Hyperparameters

- Discount Factor (γ): 0.99
- Soft Target Update Factor (τ): 0.005
- Learning Rate (LR): 1×10^{-3}
- Entropy Temperature (α): 0.2 (adjusted dynamically)
- Replay Buffer Size: 100,000
- Batch Size: 32
- Target Entropy: $-\log(1/\text{action\_dim}) \times 0.95 = -\log(0.5) \times 0.95 \approx 0.658$
- Maximum Episodes: 2000
- Evaluation Frequency: Metrics logged after every episode.

Neural Network Architecture

- **Actor Network**:
  - Input: 4-dimensional state vector.
  - Hidden Layers: Two fully connected layers with 256 neurons each, using ReLU activations.
  - Output: Logits for 2 actions (left and right), converted to probabilities using a softmax function.
- **Critic Network**:
  - Q1 and Q2 Networks: Two separate Q-networks, each configured as follows:
    * Input: 4-dimensional state vector.
    * Hidden Layers: Two fully connected layers with 256 neurons each, using ReLU activations.
    * Output: Q-values for 2 discrete actions.

Proximal Policy Algorithm on CartPole-v1 Environment

Hyperparameters

- Discount Factor (γ): 0.99
- GAE Lambda (λ): 0.95 (used for Generalized Advantage Estimation)
- Policy Clip (ϵ): 0.2 (controls the clipping range for probability ratios)

- Learning Rate (α): 3×10^{-4}
- Batch Size: 32
- Number of Epochs per Update: 4
- Maximum Steps per Episode: 500
- Number of Games (Episodes): 1000
- Reward Clipping: Rewards are clipped to the range $[-1, 1]$ to stabilize training.

Neural Network Architecture

- **Actor Network**:
  - Input: 4-dimensional state vector (CartPole observation space).
  - Hidden Layers: Two fully connected layers with 256 neurons each, using ReLU activations.
  - Output: Action probabilities (softmax distribution over the discrete action space of size 2).
- **Critic Network**:
  - Input: 4-dimensional state vector.
  - Hidden Layers: Two fully connected layers with 256 neurons each, using ReLU activations.
  - Output: Single scalar value representing the state-value function $V(s)$.

Results 1

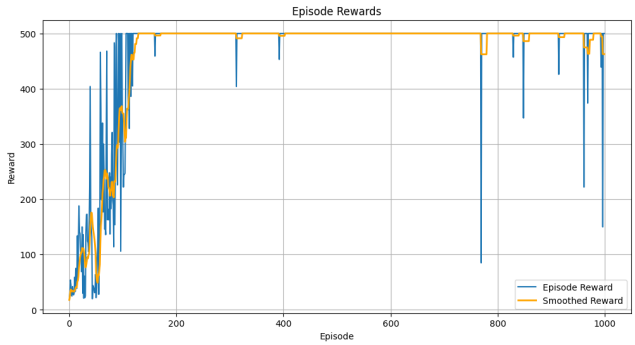

Figure 4: PPO on Cartpole-v1 Environment

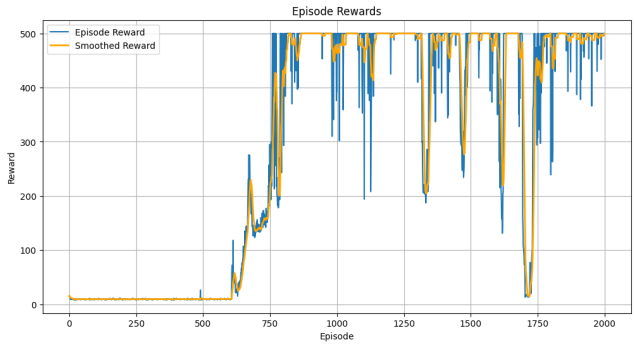

Figure 5: SAC on Cartpole-v1 Environment

1. Learning Speed:

- **PPO:** In the first graph (PPO), the agent quickly reaches a high reward around 500 within the first 200 episodes. This indicates that PPO adapts to the task quickly, showing a faster convergence in the early training phase.
- **SAC:** In the second graph (SAC), the learning is slower initially. It takes approximately 700 episodes before the rewards stabilize near the maximum. SAC, therefore, has a slower learning curve compared to PPO in this task.

2. Stability of Rewards:

- **PPO:** The PPO graph shows relatively stable rewards after convergence. Once the reward stabilizes near the maximum value (500), there are minimal fluctuations. This indicates that PPO maintains consistent performance after learning the optimal policy.
- **SAC:** The SAC plot, while achieving maximum rewards after convergence, exhibits higher fluctuations throughout the training process. Even after reaching the maximum reward, there are frequent drops, indicating less stability compared to PPO.

3. Exploration Behavior:

- **PPO:** The PPO algorithm exhibits rapid exploration and exploitation. The smooth transition in the rewards suggests that the algorithm efficiently balances exploration and exploitation, leading to faster convergence.
- **SAC:** SAC employs entropy-based exploration, which encourages more diverse action choices during training. This is reflected in the slower but steadier rise in rewards during the initial episodes. However, the sustained fluctuations post-convergence suggest that SAC's policy exploration remain longer than necessary in this environment.

4. Suitability for CartPole:

- **PPO:** CartPole is a relatively simple environment with a discrete action space, which suits PPO well. PPO's policy-gradient method with clipping stabilizes the learning process, making it highly effective in this domain.
- **SAC:** SAC, designed for continuous control tasks, can still perform well in discrete environments, but it requires more tuning to achieve optimal results. The additional complexity of SAC's entropy-based exploration may not be fully necessary for this relatively straightforward task.

Experiment 2

Soft Actor-Critic Algorithm on LunarLander-v2 Environment

Hyperparameters

- Discount Factor (γ): 0.99
- Soft Target Update Factor (τ): 0.001
- Learning Rate (LR): 5×10^{-4}
- Entropy Temperature (α): 0.2 (adjusted dynamically)
- Replay Buffer Size: 100,000
- Batch Size: 256
- Maximum Episodes: 1000
- Evaluation Frequency: Metrics logged after every episode.

Neural Network Architecture

- **Actor Network**
  - Input: state_size, action_size
  - Hidden Layer 1: Linear(state_size, hidden_size) → ReLU
  - Hidden Layer 2: Linear(hidden_size, hidden_size) → ReLU
  - Output Layer: Linear(hidden_size, action_size) → Softmax
- **Critic Network**
  - Input: state_size, action_size
  - Hidden Layer 1: Linear(state_size, hidden_size) → ReLU
  - Hidden Layer 2: Linear(hidden_size, hidden_size) → ReLU
  - Output Layer: Linear(hidden_size, action_size)

Proximal Policy Optimization Algorithm on LunarLander-v2 Environment

Hyperparameters

- Discount Factor (γ): 0.99
- GAE Lambda (λ): 0.95 (used for Generalized Advantage Estimation)
- Policy Clip (ϵ): 0.2 (controls the clipping range for probability ratios)
- Learning Rate (α): 1×10^{-3}
- Number of Epochs per Update: 5
- Number of Games (Episodes): 1000
- Reward Clipping: Rewards are clipped to the range $[-1, 1]$ to stabilize training.

Neural Network Architecture

- **Actor Network**:
  - Input: 4-dimensional state vector (LunarLander observation space).
  - Hidden Layers: Two fully connected layers with 64 neurons each, using ReLU activations.
  - Output: Action probabilities (softmax distribution over the discrete action space of size 2).
- **Critic Network**:
  - Input: 4-dimensional state vector.
  - Hidden Layers: Two fully connected layers with 64 neurons each, using ReLU activations.
  - Output: Single scalar value representing the state-value function $V(s)$.

Results 2

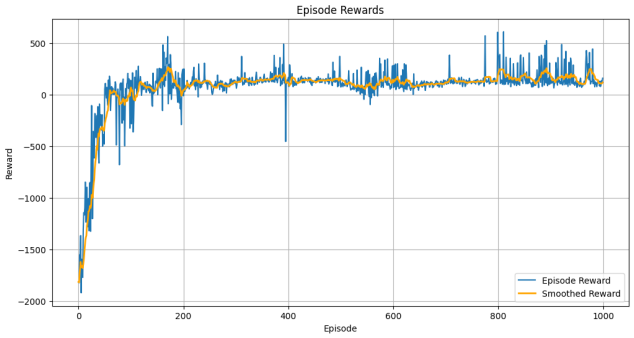

Figure 6: PPO on LunarLander-v2 Environment

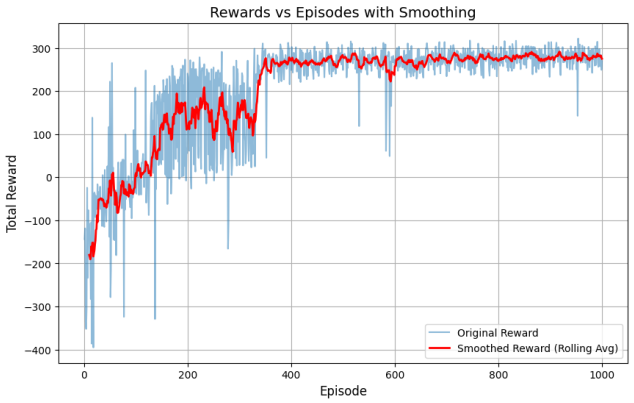

Figure 7: SAC on LunarLander-v2 Environment

1. Learning Speed:

- **PPO:**
  - **Observations:**
    - PPO demonstrates rapid initial learning, with rewards improving significantly during the first 200 episodes. The reward trajectory rises sharply from around -2000 to above 0 during this phase.
    - However, after this initial improvement, PPO stabilizes slowly, with a notable plateau in performance. This indicates slower convergence to optimal behavior.
  - **Influence of Hyperparameters:**
    - PPO's higher learning rate ($\alpha = 110^3$) accelerates the initial learning phase, allowing the agent to quickly adapt to the environment.
    - The policy clip ($\epsilon = 0.2$) stabilizes policy updates, ensuring the rapid learning phase does not lead to overly large updates. However, this clipping limits the refinement of policies in the later stages.
- **SAC:**
  - **Observations:**

* SAC also exhibits quick initial learning but takes longer to stabilize compared to PPO. It achieves near-optimal performance (rewards around 200–300) by around 400 episodes.
* SAC continues fine-tuning its policy even beyond 400 episodes, demonstrating sustained learning.

– **Influence of Hyperparameters**:
* SAC's lower learning rate ($\alpha = 510^4$) promotes gradual and stable updates, leading to slower convergence but more consistent performance.
* The entropy temperature (0.2) ensures exploration early on, allowing the agent to discover high-reward strategies even after the initial learning phase.

2. Stability of Rewards:

- **PPO:**
  – **Observations:**
  * After the early learning phase, PPO rewards stabilize but show significant variance even after convergence (episodes 400–1000). The episode rewards fluctuate frequently, with occasional sharp drops below zero.
  * This suggests that while PPO can achieve optimal performance, its policy updates may not always maintain stability over time in this environment.
  – **Influence of Hyperparameters:**
  * Policy clip ($\epsilon = 0.2$)prevents large policy updates, ensuring some stability in training. However, it limits the agent's ability to fully adjust to complex dynamics, leading to variability.
  * The higher learning rate ($\alpha = 110^3$) contributes to sharper reward fluctuations by amplifying the impact of noisy gradient updates.
- **SAC:**
  – **Observations:** SAC displays smoother reward progression after convergence, with fewer large fluctuations. The smoothed reward curve indicates consistent policy improvement and better stability than PPO in the later training stages.
  – **Influence of Hyperparameters:**
  * Entropy-based exploration balances exploration and exploitation, preventing sharp drops in rewards by avoiding early convergence to suboptimal policies.
  * The soft target update factor ($\tau = 0.001$) ensures gradual updates to the target Q-network, reducing changes in Q-value predictions and contributing to stability.

3. Exploration Behavior:

- **PPO:**
  – **Observations:**
  * PPO employs a clipping mechanism in policy updates, which balances exploration and exploitation well. However, its exploration is not as robust as SAC's entropy-driven strategy.
  * This explains the sharper variability in PPO's episode rewards, as the algorithm occasionally struggles to refine its policy in complex scenarios.
  – **Influence of Hyperparameters:**
  * The policy clip ($\epsilon = 0.2$) limits large deviations in policy updates, which constrains exploration. This can lead to insufficient exploration of better strategies in complex scenarios like LunarLander-v2.
- **SAC:**
  – **Observations:** SAC's entropy term encourages more consistent exploration, which prevents premature convergence to suboptimal policies. This allows SAC to maintain stable performance and explore alternate strategies even after achieving high rewards.

4. Convergence Quality:

- **PPO:**
  – **Observations:** PPO converges to suboptimal behavior in some episodes (evidenced by reward dips and spikes even in later episodes). While the smoothed reward hovers around 0–100, it struggles to maintain performance consistently at higher reward levels.
- **SAC:**
  – **Observations:** SAC converges to higher-quality policies, with rewards stabilizing around 200–300 in the later episodes. This indicates SAC's superior capability in optimizing long-term rewards in the LunarLander environment.

5. Suitability for LunarLander:

- **PPO:**
  – **Observations:** PPO performs well in the initial stages of learning but suffers from instability in policy refinement. Its simpler policy-gradient approach may be less suited for the nuanced control required in LunarLander.
- **SAC:**
  – **Observations:** SAC outperforms PPO in overall performance, stability, and convergence. The continuous control aspects of SAC make it particularly effective for the complex dynamics of LunarLander, where precise control over actions is critical.

Experiment 3

Proximal Policy Optimization on Taxi-v3 Environment

Hyperparameters

- Discount Factor (γ): 0.99
- GAE Lambda (λ): 0.95 (used for Generalized Advantage Estimation)
- Policy Clip (ϵ): 0.1 (controls the clipping range for probability ratios)
- Learning Rate (α): 1×10^{-4}
- Number of Epochs per Update: 10
- Number of Episodes: 1000

Neural Network Architecture

- **Actor Network**:
  - Input: 4-dimensional state vector (Taxi-v3 observation space).
  - Hidden Layers: Two fully connected layers with 256 neurons each, using ReLU activations.
  - Output: Action probabilities (softmax distribution over the discrete action space).
- **Critic Network**:
  - Input: 4-dimensional state vector.
  - Hidden Layers: Two fully connected layers with 256 neurons each, using ReLU activations.
  - Output: Single scalar value representing the state-value function $V(s)$.

Results 3

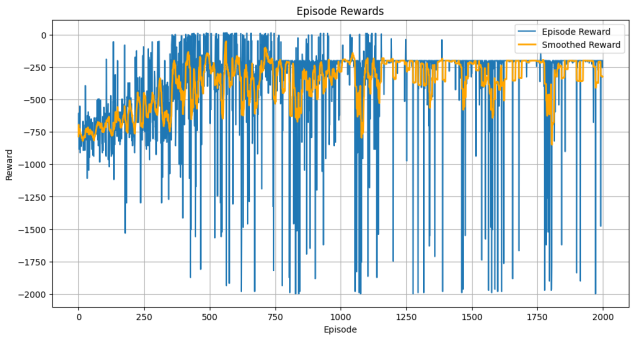

Figure 8: PPO on Taxi-v3 environment

1. Learning Speed

- **Observations:**
  - PPO demonstrates relatively slow initial learning, as the rewards improve gradually over the first 500 episodes. Starting with rewards below -1000, the agent steadily progresses toward better performance.
  - Beyond episode 500, the learning rate appears to slow further, with smaller incremental improvements.
  - By the end of training (around 2000 episodes), the smoothed rewards hover around -250, indicating suboptimal convergence.
- **Impact of Hyperparameters:**
  - Learning Rate (110^4): The low learning rate contributes to slow but stable policy updates. This gradual learning helps avoid overshooting optimal solutions but slows the rate of convergence.
  - Policy Clip ($\epsilon = 0.1$): A smaller clipping range restricts large changes in the policy, ensuring stability but also limiting the ability to make large corrections early in training. This contributes to slower improvement.

2. Stability of Rewards

- **Observations:**
  - The rewards show significant variability throughout training, with frequent sharp drops. Even in the later episodes (1500–2000), the episode rewards fluctuate heavily, occasionally dropping below -500.
  - While the smoothed rewards suggest some stabilization, the fluctuations indicate that PPO struggles to consistently refine a stable policy in this environment.
- **Impact of Hyperparameters:**
  - GAE Lambda ($\lambda = 0.95$): Generalized Advantage Estimation helps balance bias and variance in the advantage function, contributing to stability. However, this balance may not be sufficient to counter the noisy rewards in Taxi-v3.
  - Discount Factor ($\gamma = 0.99$): The high discount factor places strong emphasis on long-term rewards, which is appropriate for Taxi-v3. However, it may increase instability when immediate rewards vary greatly.

3. Exploration Behavior

- **Observations:**
  - PPO shows evidence of suboptimal exploration, as indicated by the frequent sharp drops in episode rewards. These drops suggest that the agent occasionally reverts to less effective policies or fails to adequately explore better strategies.
  - The lack of sustained upward reward trends in the smoothed curve highlights the challenges PPO faces in refining its policy.

4. Convergence Quality

- **Observations:**
  - PPO converges to a suboptimal policy, with smoothed rewards stabilizing around -250. While this reflects an improvement over the initial rewards (around -1000), it is far from the optimal behavior required for Taxi-v3.
  - The large fluctuations even after 1000 episodes indicate that PPO struggles to fully adapt to the environment's challenges, such as the need for efficient passenger pickup and drop-off.

Conclusion

This paper performs a comparative analysis of SAC and PPO in the OpenAI Gym on three environments: Taxi-v3, CartPole-v1, and LunarLander-v2. The results show that SAC, having entropy-based exploration and stability mechanisms, performs better in complex environments such as Lunar Lander-v2 by yielding higher returns and better convergence quality. On the other hand, PPO exhibits faster learning and is more stable on simpler discrete environments such as CartPole-v1, hence more suitable for tasks with fewer complexities in controls. While SAC's greater difficulty in learning and higher computation might restrict its application to simple tasks, its robustness for complex control scenarios signifies the algorithm's versatility. In contrast,

PPO's simplicity for both discrete and continuous domains make it a competitive choice for tasks in the area of moderate complexity. This comparison gives actionable insights into selecting reinforcement learning algorithms based on task requirements and environment characteristics. Future work could extend this study by evaluating the performance of these algorithms in more diverse or real-world environments, with the inclusion of extra metrics such as computational efficiency, robustness to hyperparameter variations, and scalability to higher-dimensional tasks.

References

[1] Farama Foundation. (n.d.). Lunar Lander. *Gymnasium Documentation*. Retrieved December 10, 2024, from https://gymnasium.farama.org/environments/box2d/lunar_lander/

[2] Farama Foundation. (n.d.). CartPole. *Gymnasium Documentation*. Retrieved December 10, 2024, from https://gymnasium.farama.org/environments/classic_control/cart_pole/

[3] Farama Foundation. (n.d.). Taxi. *Gymnasium Documentation*. Retrieved December 10, 2024, from https://gymnasium.farama.org/environments/toy_text/taxi/

[4] OpenAI. (n.d.). Proximal Policy Optimization (PPO): Pseudocode. *Spinning Up Documentation*. Retrieved December 10, 2024, from https://spinningup.openai.com/en/latest/algorithms/ppo.html#pseudocode

[5] OpenAI. (n.d.). Soft Actor-Critic (SAC): Pseudocode. *Spinning Up Documentation*. Retrieved December 10, 2024, from https://spinningup.openai.com/en/latest/algorithms/sac.html#pseudocode

[6] Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. Proceedings of the 35th International Conference on Machine Learning (ICML).

[7] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, Sergey Levine. (2019). *Soft Actor-Critic Algorithms and Applications.* arXiv preprint.

[8] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, Sergey Levine. (2019). *Learning to Walk via Deep Reinforcement Learning.* arXiv preprint.

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms*. arXiv preprint.

[10] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, Pieter Abbeel. (2018). *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. arXiv preprint.

[11] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

[12] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym*. arXiv preprint.

[13] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). *Continuous Control with Deep Reinforcement Learning*. arXiv preprint.

[14] Williams, R. J. (1992). *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*. Machine Learning, 8(3–4), 229–256.

Please find the code used for generating the results https://github.com/kaushik884/RLProject.