CSE 546 – Final Project

Pixel Perfect: Accelerating AI Mastery in Atari with Advanced Convolutional Reinforcement Learning

Abhiroop Ghosh	50541142
Dakshesh Gusain	50540361
Weiyang Lee	50545710

Aim:

The objective of this project is to explore and enhance the efficacy of Convolutional Neural Networks (CNNs) in conjunction with Reinforcement Learning (RL) techniques, such as Deep Q-Networks (DQN), Double DQNs, and Proximal Policy Optimization (PPO), in training autonomous agents to play various Atari games. Starting with the relatively simple environment of Pong and progressing to more complex challenges such as Breakout and Montezuma's Revenge, the project seeks to identify and fine-tune hyperparameters that accelerate the learning process. The ultimate goal is to develop faster, more efficient algorithms capable of quickly training RL agents without resorting to binary frame conversion, thereby pushing the boundaries of how raw pixel data can be leveraged to teach machines to play—and excel at—video games.

Abstract

As artificial intelligence progresses, teaching machines to understand and interact with complex environments has profound implications, not only for gaming but for broader applications in robotics, autonomous navigation, and decision-making systems. This project focuses on applying advanced CNNs and RL techniques to the task of training agents on Atari games, using the games as benchmarks for developing faster and more efficient learning algorithms.

We begin by employing DQN and Double DQN methodologies, integrating them with PPO for enhanced sample utilization efficiency. Our agents start by learning from the raw pixel data of the game Pong, using CNNs to interpret the visual input and make decisions to maximize in-game rewards. As the project progresses, we scale the complexity to more challenging games like Breakout and Montezuma's Revenge, which present intricate scenarios and require advanced strategy learning.

The research aims to demonstrate that through careful tuning of network architectures and learning parameters, agents can learn to play directly from high-dimensional sensory inputs in a more efficient and expedited manner. We postulate that deeper convolutional layers can capture the complex hierarchies of visual data essential for nuanced gameplay. This exploration is pivotal, as it could signify a leap forward in our understanding of visual perception and decision-making in artificial agents, thereby contributing significantly to the fields of artificial intelligence and machine learning. The successful implementation of such

CNN-RL systems holds the potential for far-reaching impacts, from improving computer vision applications to advancing the capabilities of autonomous systems in real-world tasks.

How this is interesting?

What elevates the intrigue of this project is its ambition to break new ground in the realm of artificial intelligence by fusing the strengths of Convolutional Neural Networks and Reinforcement Learning to unlock a higher echelon of autonomous strategic gameplay. The project is compelling due to its methodical approach from the pixelated simplicity of Pong to the sophisticated labyrinth of Montezuma's Revenge, each game serving as a progressively intricate puzzle that challenges the agent to evolve. It is the translation of raw visual input into strategic mastery, all without diluting the richness of the visual data into binary simplicity, that promises not just an evolution in how machines play games but in how they perceive and interact with the world.

The challenges presented by the complex environments in games like Montezuma's Revenge are analogous to navigating through the unpredictable nature of real-world scenarios. The cognitive processes involved in discerning patterns, adapting strategies, and learning from visual cues within game settings mirror those required in dynamic real-life contexts. Training agents on rich, unprocessed visual inputs in gaming environments fosters a sophisticated level of pattern recognition that is directly applicable to real-world applications. For instance, the same principles that enable an agent to navigate a virtual labyrinth can be applied to autonomous vehicles maneuvering through traffic or robotic systems performing tasks in variable and unstructured environments. By leveraging the deep learning and analytical skills honed in these game-based puzzles, AI agents can be better equipped to interpret complex imagery and make informed decisions in diverse, real-world settings, thereby enhancing the efficacy and reliability of AI solutions across various sectors.

Objectives

1. Integrate and optimize CNNs with RL techniques: Evaluate and refine the integration of Convolutional Neural Networks with Reinforcement Learning algorithms such as DQN, Double DQN, and PPO to train agents on Atari games, focusing on hyperparameter optimization for efficient learning.
2. Scale learning from simple to complex tasks: Progress agent training from the basic game of Pong to the intricate environments of Breakout and Montezuma's Revenge, aiming to enhance the agents' ability to learn and strategize from complex visual inputs.
3. To investigate the transferability of game-learned skills: Explore the potential for strategies and recognition patterns learned in game environments to be transferred to real-world applications, such as autonomous navigation and computer vision tasks.

Algorithms applied

Deep Q-network

Definition

An approach in reinforcement learning that combines Q-learning, which is an off-policy reinforcement learning algorithm and is model-free, with deep neural networks. DQNs were developed to deal with high-dimensional state and action spaces, enabling the application of Q-learning. This was done to overcome the curse of dimensionality. The network is used to approximate the Q-value function, which would ordinarily require a Q-table with a very large size. The introduction of deep learning with Q-learning, helps to generalize over similar states, hence making the learning more efficient and scalable.

Algorithm

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Merits

1. Uses experience replay, which learns from all past policies. This improves sample efficiency and stabilizes the learning process by reducing correlations in the observation sequence.
2. Freezes the target Q-network, which avoids oscillations and breaks correlations between Q-network and target.
3. Clips rewards or normalizes the network adaptive to sensible range.
4. DQNs can handle large and more complex state spaces that traditional Q-learning, thanks to function approximation capabilities of neural networks.

Demerits

1. The non-linear function approximators like a neural network, can lead to instability and divergence in the learning process. Techniques like fixed target networks and reward clipping would be necessary to mitigate these issues.
2. Training DQNs require significant computational resources and time, especially as the complexity of the environment increases.
3. The performance of DQNs can be highly sensitive to the choice of hyperparameters, such as learning rate, size of replay buffer and batch size.

Double Deep Q-network

Definition

An extension of standard DQN, which reducing the overestimation bias of action values that occurs in standard DQN. In traditional DQN, both the selection of best action and the evaluation of that action value is performed using the same network, which led to significant overestimations. Double DQN addresses this issue by performing the selection and evaluation of the action's value using two networks: policy network and target network.

Algorithm

Algorithm 1: Double DQN (Hasset et al. 2015

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau < 1$, C

for *each iteration* **do**

  for *each step* **do**

    Observe state s_t and select $a_t \sim \pi(s_t)$

    Execute action a_t and observe next state s_{t+1} and reward r_t;

    Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$.

  end

  for *each update step* **do**

    Sample minibatch of transitions $(s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$

    Compute target Q value:

$$Q^*(s_t, a_t) = r_t + \gamma Q_{\theta'}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a'))$$

    Perform a gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$ with respect to the primary network parameters θ

    Every C steps update target network parameters:

$$\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$$

  end

end

Merits

1. By decoupling the action selection from the value estimation by using two different networks, this algorithm leads to more accurate value estimates by reducing the overestimation bias.
2. This algorithm leads to better policy performance and more stable learning, particularly in environments where there is noise or deceptive signals.

Demerits

1. Requires maintenance and periodic update of two separate neural networks, which can increase computational overhead and memory usage.
2. The process of updating two networks and managing their interaction adds complexity.
3. Double DQN is sensitive to hyperparameters: learning rate, buffer size, batch size and target update frequency.

Proximal Policy Optimization

Definition

PPO is a policy gradient method for reinforcement learning which optimizes a surrogate objective function via stochastic gradient descent. PPO aims to take the largest possible improvement on a policy while avoiding excessive updates and ensuring a lower variance of policy updates, which leads to improved training stability.

Algorithm

$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$, *where* $a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t)$, for $t \geq 0$

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

  by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \mathrm{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for

Merits

1. PPO is more stable and robust to hyperparameter settings, compared to other policy gradient methods.
2. PPO is easier to implement and tune compared to complex algorithms like TRPO

3. PPO is more sample efficient than on-policy methods like vanilla policy gradient or A2C.

Demerits

1. PPO can still be computationally expensive due to multiple epochs of minibatch updates
2. Being an on-policy algorithm, it may be inefficient in terms of sample reuse, since it requires fresh samples from the policy for each update.

Environments

Pong-v4

'Pong-v4' is a version of the classic Atari 2600 game "Pong" implemented in the OpenAI Gym environment. In this game, players control paddles to hit a ball back and forth across the screen. The objective is to score goals, by passing the ball beyond the opponent's paddle.

PongDeterministic-v4

This is a deterministic setting of the Pong game. In deterministic versions of Atari games, actions taken in the environment lead to a predetermined sequence of states given the same initial conditions and actions, reducing randomness in the environment's responses.

BreakoutNoFrameskip-v4

BreakoutNoFrameskip-v4 is based on the Atari game 'Breakout' where players control a paddle to hit a ball upwards to break bricks at the top of the screen. The goal is to clear all the bricks without letting the ball pass the paddle. The "NoFrameskip" variant of this environment does not skip frames – every frame is rendered and shown to the agent. This provides a more challenging and granular control environment, as skipping frames can sometimes simplify the task.

GravitarDeterministic-v4

This is a deterministic version of the Atari 2600 game "Gravitar". In this game, players must navigate a spaceship through different planetary systems each presenting unique challenges, like gravity, shooting turrets, and fuel actions.

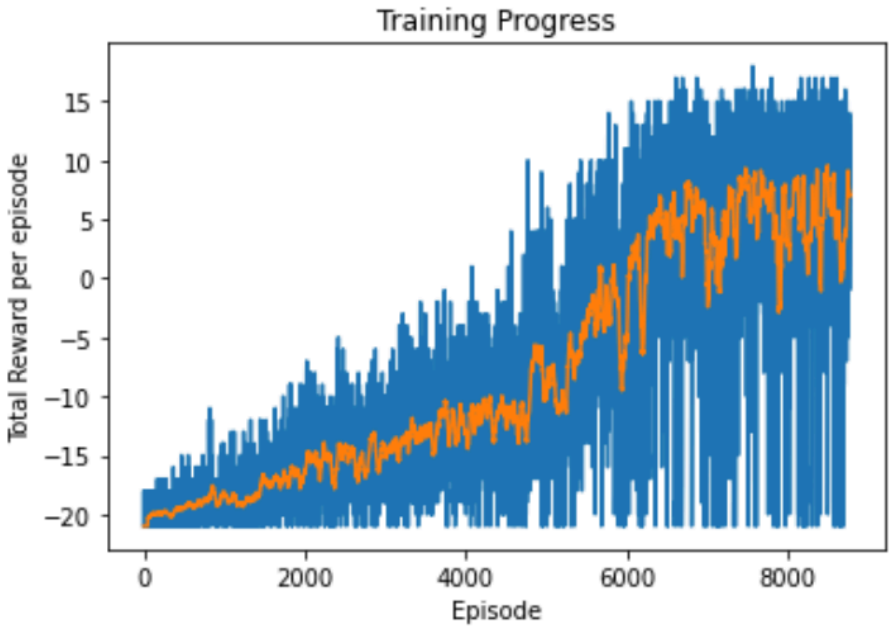
LunarLander-v4

This is an environment simulates a scenario where a lunar lander must be controlled to land safely on the moon. In the LunarLander-v4 environment, the player controls a lander using discrete actions with the objective of landing it between two flag markers on a flat surface. The challenge is to make the landing as smooth as possible without crashing. The environment simulates physics like gravity, inertia, and thrust, making the task non-trivial and requiring strategic control of the lander's thrusters.

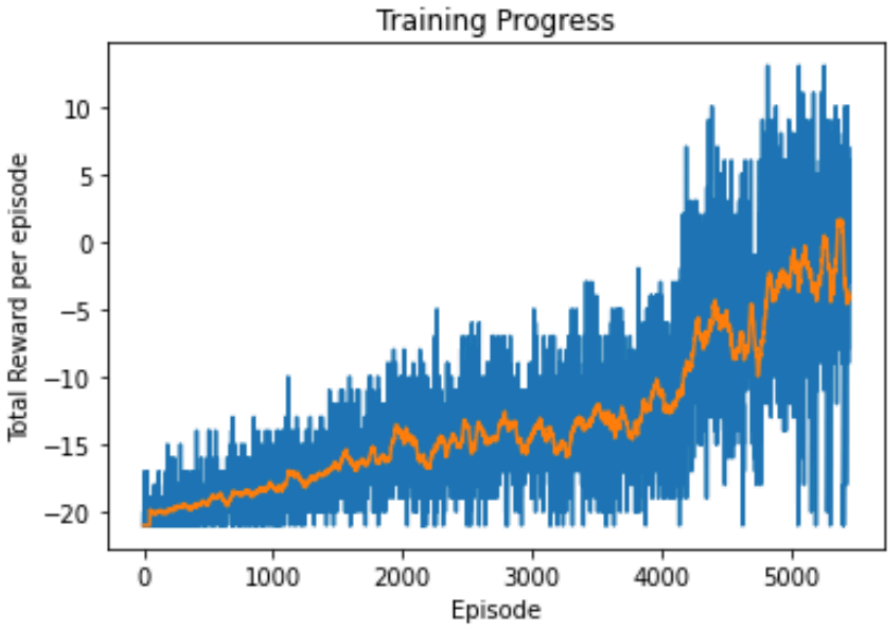
Results:

A dynamic plot function was used to plot the rewards earned per episode, so that we can observe how the model learns during training without waiting for the end of the training. Here are the baseline model results obtained while running on CCR.

Pongv-4 DQN:



Pongv-4 Double DQN



Inferences

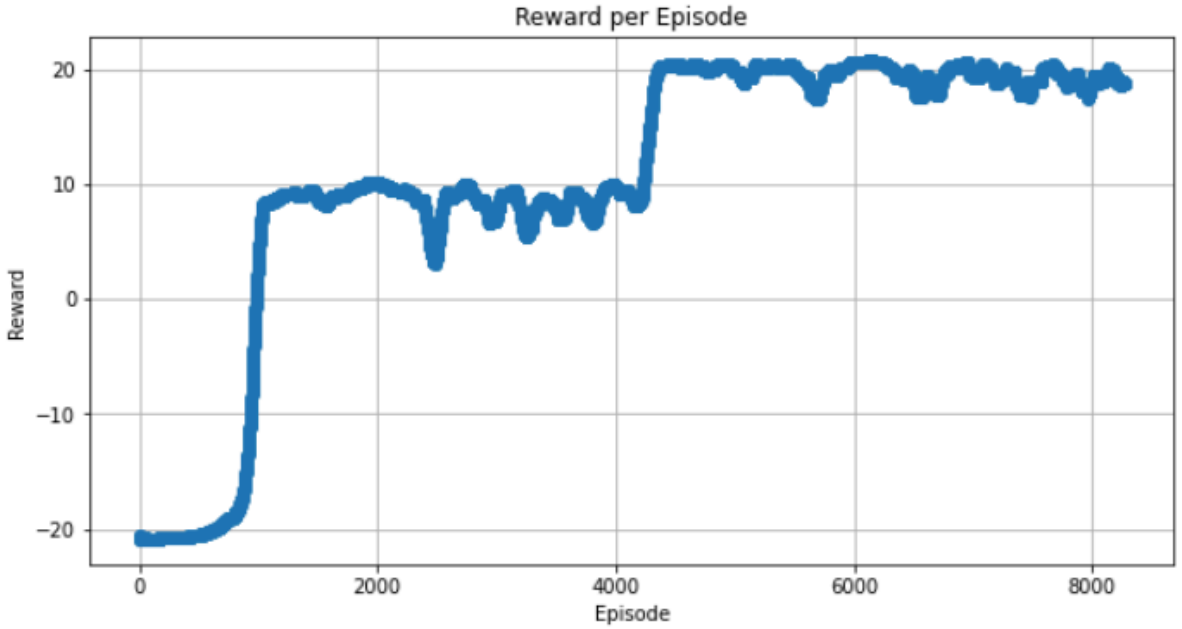
1. For DQN, the agent's performance gradually improves over approximately 9000 episodes. The moving average (orange line) suggests that the agent becomes consistently better at the game, reaching near a reward of 0 but still shows significant variability in individual episode rewards. This variability indicates some instability in the policy or challenges in fully mastering the game environment.
2. For doubleDQN, the agent starts from a similar low reward and shows improvement over 5000 episodes. It seems to achieve slightly better and more stable performance earlier than the standard DQN model, with the moving average window approaching a reward of 5. However, just like the DQN, there is significant fluctuation in the reward, showing that while the performance has improved, there are still episodes where the policy does not perform optimally.

These results weren't that good, so we ignored the evaluation part, made some changes to the neural network and tried a new algorithm: PPO on PongDeterministic-v4 environment.

We hoped this may lead to a more stable learning curve for the agent.

PongDeterministic-v4 PPO:

Training curve



Evaluation curve

```
Environment 1, Episode 1/10: Reward = 21.00
Environment 1, Episode 2/10: Reward = 21.00
Environment 1, Episode 3/10: Reward = 21.00
Environment 1, Episode 4/10: Reward = 21.00
Environment 1, Episode 5/10: Reward = 21.00
Environment 1, Episode 6/10: Reward = 21.00
Environment 1, Episode 7/10: Reward = 21.00
Environment 1, Episode 8/10: Reward = 21.00
Environment 1, Episode 9/10: Reward = 20.00
Environment 1, Episode 10/10: Reward = 20.00
```


```
Evaluation Average Reward: 20.8
```

Inferences

1. The PPO algorithm performs pretty well in the environment. The agent achieves near optimal performance, demonstrating the algorithm's effectiveness at handling the exploration and exploitation tradeoff and optimizing the policy network efficiently.
2. The sharp rise and plateau in the training curve suggests that the PPO efficiently exploits the learning signal from the environment once it starts to understand the game dynamics. This is indicative of PPO's advantage in terms of sample efficiency and stability over other methods like DQN and Double DQN.
3. The consistent high scores during evaluation phase underscore the robustness of the learned policy.

Due to the consistent high scores obtained, we went ahead to test the ability of this algorithm and neural network on another environment. Hence, the algorithm was implemented on another environment. The only difference was the change in the environment name, everything else was kept the same.

Notes

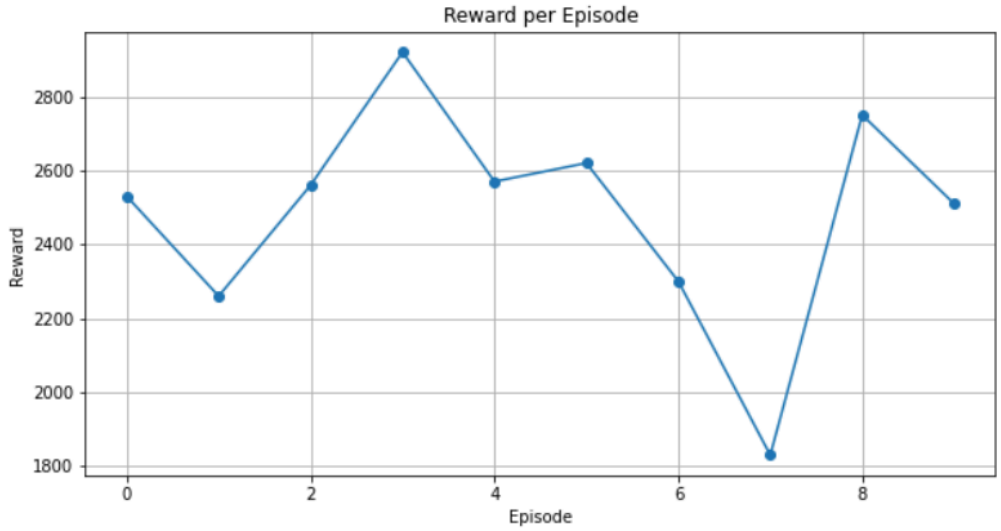
1. The agent was trained on three environments of Pong asynchronously and the rewards per episode was the average reward across the last 100 episodes for all three environments. This helped the agent find a stable learning policy much quicker.

GravitarDeterministic-v4 PPO

Training curve

Evaluation curve

```
Episode:1, Reward:2530.0
Episode:2, Reward:2260.0
Episode:3, Reward:2560.0
Episode:4, Reward:2920.0
Episode:5, Reward:2570.0
Episode:6, Reward:2620.0
Episode:7, Reward:2300.0
Episode:8, Reward:1830.0
Episode:9, Reward:2750.0
Episode:10, Reward:2510.0
```




```
Evaluation Average Reward: 2485.0
```

Inferences:

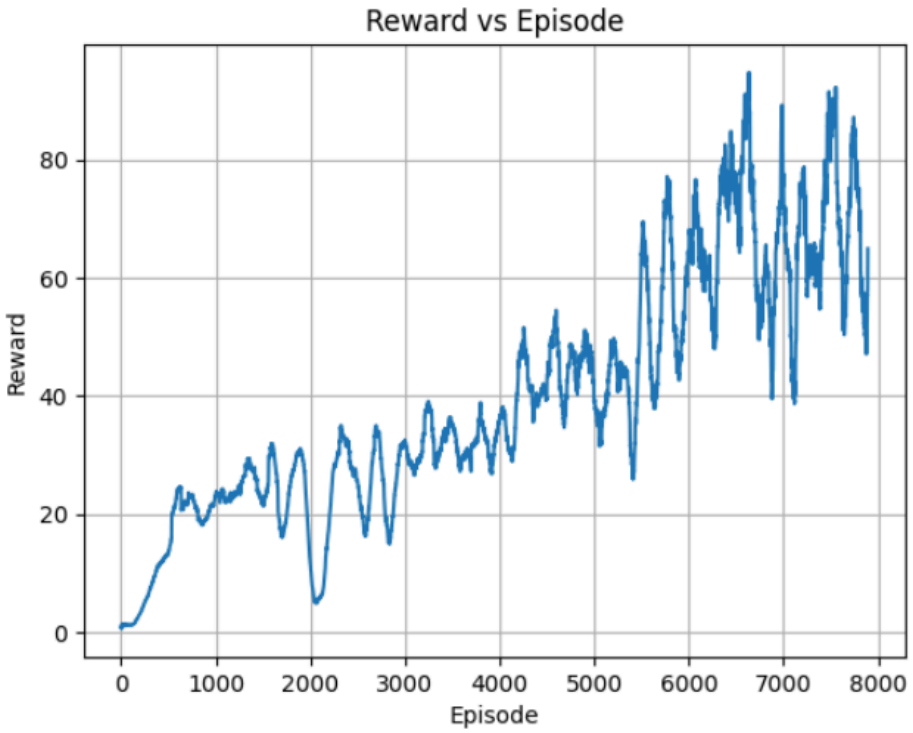
1. The training curve shows a steady increase in rewards during the initial phase ,indicating that the agent is learning effectively from the environment. Around the middle of the training, the rewards began to plateau, suggesting that the agent has achieved a stable understanding of the policy required to maximize rewards in the game.
2. The overall trend is upward until it stabilizes, which is typical for this policy. The major dip and subsequent recovery could be due to the agent encountering and adapting to new states and scenarios. This is true, since there are upto 5 levels in this game.
3. As the agent progresses to a new level, it is bound to fail so that it can learn and adapt to the near environment. However, since it has a relatively good understanding of the environment and its action space, it quickly adapts and picks up the new optimal policy.
4. The varying results in the evaluation curve suggests that while the agent has learned a generally effective strategy for achieving high scores, its performance can still be inconsistent. This may be due to the complexity of the "Gravitar" game, where strategic decision making is crucial and small variations in behavior can lead to significantly different outcomes.

Note:

1. There is no max steps in Gravitar environment. The max number of steps was set to 5000.
2. The agent was trained on three environments of Gravitar asynchronously, and the rewards were obtained from all three and averaged, for each particular episode. This enabled the algorithm to learn the optimal policy much quicker.

BreakoutNoFrameskip-v4 PPO

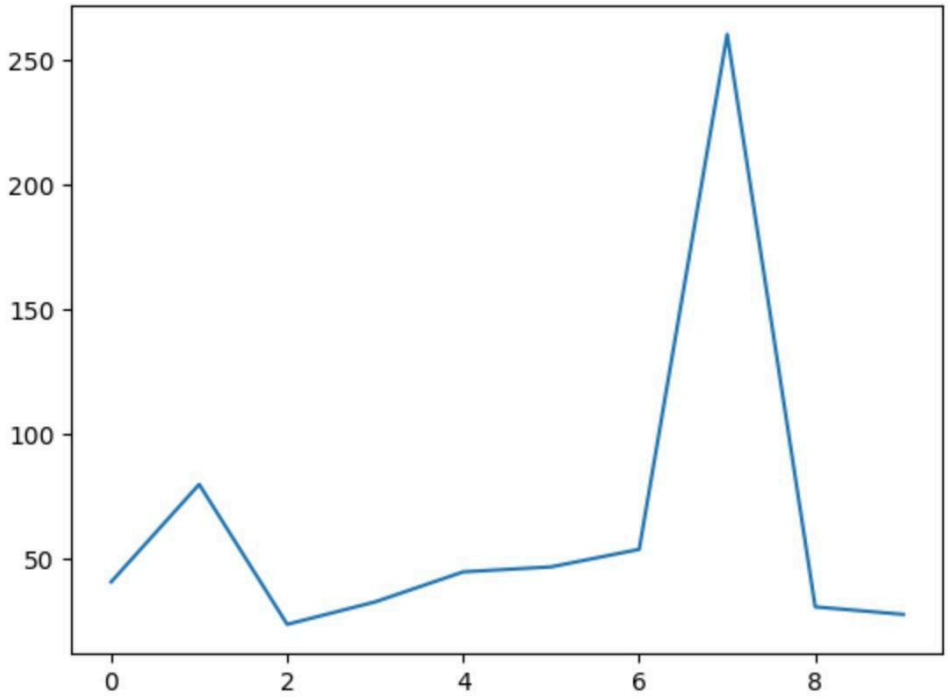
Training curve



Inferences-

1. Initially, the rewards fluctuate significantly, indicating exploration and learning. Initially the agent takes time to learn achieving lesser reward but theres an upward trend which shows that the agent is learning really well. The agent seems to learn an effective policy, as evidenced by the upward trend. If trained for more time then significant rewards can be achieved.
2. The x-axis is the episode number, and the y-axis is the reward. The reward is a numerical score that the agent receives for taking actions in the environment. In Breakout, the agent receives a reward for breaking out blocks and a penalty for losing the ball. The graph shows that the agent's reward increases over time. This suggests that the agent is learning to play the game better.
3. **Early Episodes:** The initial low rewards suggest the agent struggled at first to achieve its goals in the game.
4. **Later Episodes:** The rise in rewards in later episodes suggests the agent gradually learned effective strategies for playing.
5. **Rate of Increase:** The rate of increase in reward over episodes can indicate how quickly the agent is learning. A steeper slope suggests faster learning.
6. **Steady Improvement:** If the line continues its upward trend, it suggests the agent is still learning and may achieve even higher rewards in future episodes.

Evaluation curve

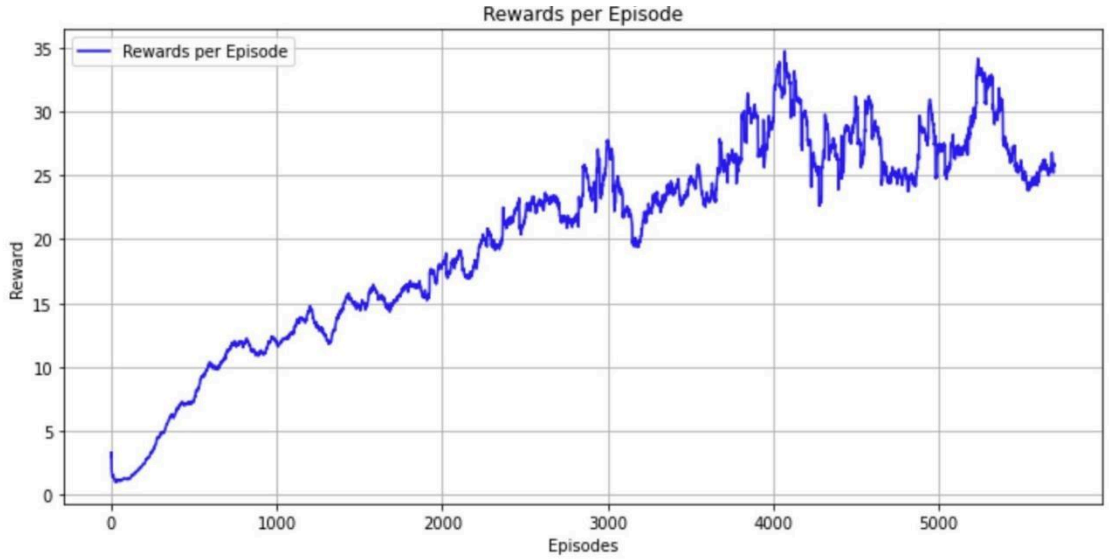


Inferences-

1. **Upward Trend:** The generally increasing slope of the line indicates the agent's performance is improving as the number of episodes (training iterations) increases. This suggests the agent is learning to play Breakout better.
2. **Fluctuations:** The line isn't perfectly smooth, and there are fluctuations in the reward throughout training. This is common in reinforcement learning as the agent explores different strategies and encounters some randomness in the environment.
   Early Episodes: The initial low rewards suggest the agent struggled at first to achieve its goals in the game.
3. **Later Episodes:** The rise in rewards in later episodes suggests the agent gradually learned effective strategies for playing Breakout.
4. **Rate of Increase:** The rate of increase in reward over episodes can indicate how quickly the agent is learning. A steeper slope suggests faster learning.
5. **Steady Improvement:** If the line continues its upward trend, it suggests the agent is still learning and may achieve even higher rewards in future episodes.

Lunar-Lander-v2 PPO (playground)

Training curve



Inferences-

1. Steady Improvement (Episodes 500-2000): The graph between episodes 500 and 2000 shows a steady increase in the average reward, with some fluctuations. This suggests the agent is continuously learning and improving its landing precision.
2. Possible Convergence (Episode 2000): The graph appears to plateau near high reward in the later episodes. This suggests the agent might have converged to a policy that allows it to land the Lunar Lander consistently with high rewards.
3. Overall, the graph suggests that the agent using PPO has successfully learned to land the Lunar Lander within 2000 episodes. The reward trajectory shows steady improvement with some exploration, and the final performance seems stable

References

1. GitHub - bhctsntrk/OpenAIPong-DQN: Solving Atari Pong Game w/ Duel Double DQN in Pytorch
2. Deep Q-Network (DQN)-I. OpenAI Gym Pong and Wrappers | by Jordi TORRES.AI | Towards Data Science
3. OpenAI Baselines: DQN
4. GitHub - GiannisMitr/DQN-Atari-Breakout: A Deep Q-Network trained to play Breakout Atari game on OpenAI Gym environment.
5. Learning Montezuma's Revenge from a single demonstration (openai.com)
6. https://github.com/vwxyzjn/cleanrl/blob/master/cleanrl/ppo.py