Exploring Reinforcement Learning Techniques Across Atari Game Environments

Rusmia Sattar[†]

[†] *Department of Mathematics and Statistics, Memorial University of Newfoundland, St. John's (NL) A1C 5S7, Canada*

E-mail: rsattar@mun.ca

Abstract

This project explores Reinforcement Learning (RL) techniques through the training of agents on Atari games utilizing environments provided by Stable Baselines. The study primarily focuses on evaluating the performance of the Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) algorithms in acquiring game strategies and assessing the influence of diverse environments on agent training. By conducting several experimental sessions across various Atari game environments such as 'SpaceInvadersNoFrameskip-v4', 'LunarLander-v2', 'Acrobat-v1' the research aims to assess the adaptability of both the PPO and A2C algorithms. The study examines agent performance under different training durations (5000, 10000, and 20000 time steps) elucidating learning trends and generalization across these environments. The outcomes reveal the adeptness of the algorithms in mastering complex Atari game strategies, showcasing their adaptability across diverse environments. The study underscores the substantial impact of environment complexities on agent training and performance, providing insights into the adaptability of RL algorithms for real-world applications. However, while highlighting implications for advancing AI development, particularly in gaming AI, the study acknowledges limitations in generalizing findings to encompass all RL algorithms and real-world scenarios due to its focus on specific algorithms and environments.

Keywords: Reinforcement Learning, Atari Game, PPO, Environments

1 Introduction

Reinforcement learning (RL) represents a pivotal branch within the realm of machine learning, distinguished by its ability to enable agents to learn optimal behaviors through interactions with environments. Unlike conventional supervised learning approaches, RL models learn through trial and error, receiving feedback in the form of rewards, and iteratively refining their strategies to achieve specific objectives.[1]

The integration of RL techniques in training agents for Atari games has emerged as a compelling domain for evaluating the adaptability and learning capabilities of AI systems. Atari games provide diverse challenges with varying complexities, making them an ideal testbed to assess the efficacy and generalization capabilities of RL algorithms in handling diverse tasks.[2]

This project embarks on an exploration of Reinforcement Learning techniques, focusing on training agents to play Atari games using environments provided by Stable Baselines. Stable Baselines, a renowned reinforcement learning library, offers an array of environments, providing a diverse landscape for evaluating agent performance across a spectrum of game complexities. The primary objectives encompass two key aspects: firstly, to evaluate the performance and effectiveness of the Proximal Policy Optimization (PPO) algorithm, a widely used RL technique, in learning strategies for Atari games. We also explore the performance of one other algo-

rithm named the Advantage Actor-Critic (A2C) throughout our study. Secondly, the project aims to assess how gaming environments like 'SpaceInvadersNoFrameskip-v4', 'LunarLander-v2', 'Acrobat-v1' influence the learning process and generalization of trained agents.[3]

1.1 Algorithms

Proximal Policy Optimization (PPO): PPO is a state-of-the-art policy optimization algorithm used in reinforcement learning. It belongs to the family of on-policy algorithms that aim to iteratively update the policy by maximizing the expected cumulative reward. PPO ensures stable and efficient learning by constraining the policy updates to a "proximal" region, preventing large policy changes that might disrupt learning. It strikes a balance between sample efficiency and stability, making it well-suited for training agents in complex environments.[4]

Advantage Actor-Critic (A2C): A2C is a synchronous variant of the Asynchronous Advantage Actor-Critic (A3C) algorithm. It combines elements of policy gradient methods with value function approximation. A2C employs an actor-critic architecture, where the actor suggests actions, and the critic evaluates these actions' advantages over the current policy. This algorithm leverages the advantages of both value-based and policy-based methods, leading to more stable training and improved sample efficiency compared to some other algorithms. A2C has shown effectiveness in learning policies for continuous control tasks and discrete action environments like Atari games.[5]

1.2 Atari Game Environments

We used three Atari game environments from Stables Baseline[3] for this project: SpaceInvaders, LunarLander, and Acrobot to analyze the diverse challenges that test different aspects of an agent's capabilities. Each environment presents unique dynamics, ranging from quick reflex-based gameplay to precise control and strategic decision-making, enabling comprehensive evaluation and benchmarking of reinforcement learning algorithms.

'SpaceInvadersNoFrameskip-v4': Space Invaders is an iconic arcade game where the player controls a spaceship moving horizontally at the bottom of the screen, aiming to shoot descending aliens while avoiding their projectiles. The 'SpaceInvadersNoFrameskip-v4' variant in Gym represents the game without frame skipping, meaning each action directly corresponds to a frame in the game. This environment demands quick decision-making and precise aiming, making it a challenging testbed for reinforcement learning agents.[6]

'LunarLander-v2': Lunar Lander simulates the task of landing a spacecraft on the moon's surface. In 'LunarLander-v2', the player controls the spaceship's throttle and orientation to land smoothly while conserving fuel. The environment involves mastering the delicate balance between propulsion and gravity, making it a classic control problem.[1] Success requires learning to navigate with limited resources, making it a suitable environment for testing an agent's ability to handle continuous control tasks.[7]

'Acrobat-v1': Acrobot represents a two-link robot trying to swing its end-effector to reach a specific goal. The objective is to make the end-effector reach a specific height by applying torques at the robot's joints. 'Acrobat-v1' challenges agents to learn sequential actions and understand the long-term consequences of their decisions. It involves learning complex motor control and sequential decision-making, making it a valuable environment for testing an agent's ability to solve complex manipulation tasks.[8].

By experimentally training agents across various Atari game environments with varying complexities, the project endeavors to assess the adaptability and robustness of the PPO and A2c algorithm. Furthermore, it seeks to identify trends and patterns in agent performance across diverse environments, shedding light on the influence of environment complexity on the learning process. Understanding the behavior of RL algorithms in complex and varied environments carries substantial implications across numerous domains. The outcomes of this project have the potential to inform the development of more robust and adaptable AI agents for real-world applications. They can contribute significantly to advancements in gaming AI, enhancing player experiences and refining the capabilities of AI opponents in gaming environments.

However, it is essential to acknowledge certain limitations of this project. While striving to comprehensively explore RL in Atari games using Stable Baselines, the studyâ€™s focus on specific algorithms and environments might not universally apply to all RL algorithms or real-world scenarios. Additionally, constraints in computational resources and time may impose limitations on the depth and breadth of experimentation.

2 Methods

The first objective of the project involves assessing the performance and effectiveness of the PPO algorithm in acquiring game strategies within the Atari game environments. The study also explores the performance of the A2C algorithm to contrast and compare its efficacy with PPO.

The second objective revolves around investigating the influence of gaming environments on the learning process and generalization capabilities of trained agents. Initially, necessary libraries are imported, including 'gym' for interfacing with the Atari environments and the PPO algorithm from Stable Baselines3. The specific Atari environment, 'SpaceInvadersNoFrameskip-v4', is defined, representing Space Invaders without frame skipping. Subsequently, the code creates the Atari environment using Gym, initializing the game interface for the agent. A PPO model is then instantiated with a convolutional neural network policy ('CnnPolicy') designed to process raw pixel inputs from the Space Invaders environment. The model undergoes training within the Atari environment for a specified number of time steps (set at 10,000 in this instance). Post-training, the model's performance is evaluated by estimating the mean reward obtained over 10 evaluation episodes, showcasing the trained agent's proficiency in playing Space Invaders. The evaluation results, namely the mean reward and standard deviation, are printed to the console, providing insight into the agent's learned behavior and performance within the game environment.

```
import gymnasium as gym
from stable_baselines3 import PPO
# Defining the Atari environment name for Space Invaders
env_id = 'SpaceInvadersNoFrameskip-v4'
# Creating the Atari environment
env = gym.make(env_id)
# Defining the PPO model and training it
model = PPO("CnnPolicy", env, verbose=1)
total_timesteps = 10000
model.learn(total_timesteps=total_timesteps)
# Evaluating the trained model
mean_reward, std_reward = evaluate_policy(model, env, n_eval_episodes=10)
print(f"Mean reward: {mean_reward:.2f} +/- {std_reward:.2f}")
```

Then we train the agent for some additional timesteps. Training sessions were conducted with varying durations, spanning 5000, 10000, and 20000 time steps, to elucidate learning trends and agent generalization across these diverse environments. Each iteration sets up an evaluation environment to monitor model performance during training. An evaluation callback, configured to save logs and best models at intervals of 500 steps, accompanies each training session.

```
# Training the model with multiple sets of total_timesteps
total_timesteps_list = [5000, 10000, 20000]
for idx, total_timesteps in enumerate(total_timesteps_list):
    print(f"Training {idx + 1}/{len(total_timesteps_list)}")
    # Setting up evaluation callback for each training run
    eval_env = gym.make(env_id)
    eval_callback = EvalCallback(eval_env, best_model_save_path=f'./logs/{idx}/',
                                 log_path=f'./logs/{idx}/', eval_freq=500,
                                 deterministic=True, render=False)
# Training the model
model.learn(total_timesteps=total_timesteps, callback=eval_callback)
# Visualizing training progress using Tensorboard logs for each run
for idx, _ in enumerate(total_timesteps_list):
    !tensorboard --logdir ./logs/{idx}/
```

After completing the training loops, TensorBoard logs are saved separately for each run in designated directories. Finally, TensorBoard is launched to visualize training progress by referencing the log directories for each specific training run. This structured approach allows for the comparison and analysis of the model's performance across varying training durations, aiding in comprehensive training assessment and monitoring using TensorBoard's visualization capabilities.

Next we experiment two different Atari game environments 'LunarLander-v2' and 'Acrobot-v1', utilizing Stable Baselines3 library. Firstly, it sets up vectorized environments for both tasks to handle parallel execution, allowing for efficient training. Subsequently, a Proximal Policy Optimization (PPO) model is initialized for the LunarLander environment using an MLP policy ('MlpPolicy') suited for discrete action spaces. The PPO model undergoes training for 10,000 timesteps within the LunarLander environment to learn optimal strategies. Similarly, an Advantage Actor-Critic (A2C) model is instantiated for the Acrobot environment employing the same MLP policy type, and trained for 10,000 timesteps to acquire effective strategies. We evaluate the performances of both environments and the optimizers efficiency based on the environments.

```
# Creating environments for LunarLander and Acrobot
lunarlander_env = make_vec_env('LunarLander-v2', n_envs=1)
acrobot_env = make_vec_env('Acrobot-v1', n_envs=1)

# Instantiating and training the PPO model for LunarLander
lunarlander_model = PPO("MlpPolicy", lunarlander_env, verbose=1)
lunarlander_model.learn(total_timesteps=10000)

# Training the A2C model for Acrobot
acrobot_model = A2C("MlpPolicy", acrobot_env, verbose=1)
acrobot_model.learn(total_timesteps=10000)
```

After training, the code evaluates the learned policies of both agents. It assesses the LunarLander agent's performance by computing the mean reward over 10 evaluation episodes, providing insight into its learned strategies. Similarly, the Acrobot agent's performance is evaluated by calculating the mean reward over 10 episodes, allowing for comparison of both agents' effectiveness in their respective environments.

2.1 Discussion

The experimental methodology involves training agents using PPO and A2C algorithms across the specified Atari game environments. The focus was on assessing the adaptability and robustness of these algorithms to varying complexities within the gaming environments. By conducting comprehensive analyses, the study aims to identify trends and patterns in agent performance, shedding light on the impact of environment complexity on the learning process and the algorithms' adaptability.

The outcomes of this study are anticipated to offer insights into the behavior and adaptability of RL algorithms in intricate and diverse environments. Such findings carry significant implications across multiple domains, including AI development for real-world applications. Understanding the adaptability of these algorithms in varied environments could contribute to the creation of more resilient and versatile AI agents for addressing complex real-world challenges. However, it's important to acknowledge the study's focus on specific algorithms and environments, thereby limiting generalizability to encompass all RL algorithms and real-world scenarios.

3 Results

3.1 Analysis and Visualization of Models

This project primarily focuses on training an agent in the Atari space Environments and evaluating the performance of reinforcement learning algorithms (PPO, A2C). We analyzed this by starting with training an agent for 10,000 steps in the 'SpaceInvadersNoFrameskip-v4' environment and we used the "CnnPolicy" neural network policy.

Training an AI agent in the SpaceInvadersNoFrameskip-v4 environment involves using reinforcement learning algorithms to learn optimal strategies for playing Space Invaders. The goal is for the agent to maximize its score by shooting aliens and avoiding being hit by alien projectiles, ultimately aiming to achieve the highest possible score in the game.[6]

After evaluating the trained model we found that it had a mean reward of 730.00 +/- 0.00. It took approximately 7 minutes to train the model. Training an agent to navigate the SpaceInvadersNoFrameskip-v4 environment using the Proximal Policy Optimization (PPO) algorithm poses several formidable challenges. At the forefront is the high dimensionality and visual complexity of the game. With raw pixel images as observations, the environment presents a vast and intricate visual space. Learning directly from these high-dimensional inputs demands substantial computational resources and prolonged training periods. Given the complexity of Space Invaders and the high-dimensional nature of its state space, training an agent to perform well may demand a vast number of samples, resulting in prolonged training times and increased computational requirements. Figure 1 shows some of the findings while training the model.

```
Using cpu device
Wrapping the env with a `Monitor` wrapper
Wrapping the env in a DummyVecEnv.
Wrapping the env in a VecTransposeImage.
----------------------------------
| rollout/            |          |
|    ep_len_mean      | 1.99e+03 |
|    ep_rew_mean      | 70       |
| time/               |          |
|    fps              | 116      |
|    iterations       | 1        |
|    time_elapsed     | 17       |
|    total_timesteps  | 2048     |
----------------------------------
```

(a) Model visualization at 2048 training steps

```
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 2.2e+03     |
|    ep_rew_mean          | 140         |
| time/                   |             |
|    fps                  | 15          |
|    iterations           | 3           |
|    time_elapsed         | 400         |
|    total_timesteps      | 6144        |
| train/                  |             |
|    approx_kl            | 0.013069006 |
|    clip_fraction        | 0.179       |
|    clip_range           | 0.2         |
|    entropy_loss         | -1.76       |
|    explained_variance   | 0.126       |
|    learning_rate        | 0.0003      |
|    loss                 | 0.734       |
|    n_updates            | 20          |
|    policy_gradient_loss | -0.00153    |
|    value_loss           | 4.66        |
-----------------------------------------
```

(b) Model visualization at 6144 training steps

```
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 1.99e+03    |
|    ep_rew_mean          | 70          |
| time/                   |             |
|    fps                  | 19          |
|    iterations           | 2           |
|    time_elapsed         | 208         |
|    total_timesteps      | 4096        |
| train/                  |             |
|    approx_kl            | 0.00843427  |
|    clip_fraction        | 0.0469      |
|    clip_range           | 0.2         |
|    entropy_loss         | -1.78       |
|    explained_variance   | 0.00449     |
|    learning_rate        | 0.0003      |
|    loss                 | 0.158       |
|    n_updates            | 10          |
|    policy_gradient_loss | -0.00954    |
|    value_loss           | 1.67        |
-----------------------------------------
```

(c) Model visualization at 4096 training steps

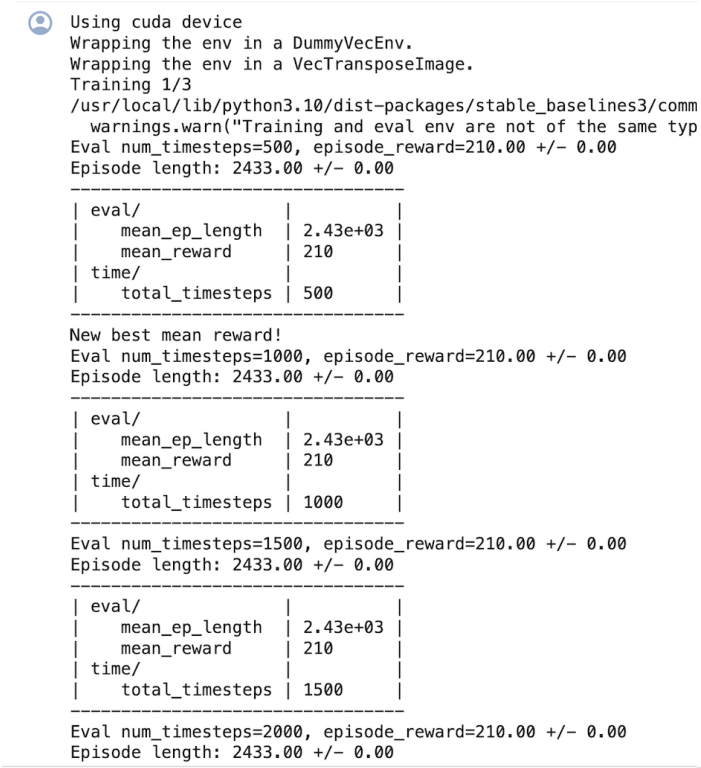
```
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 2.27e+03    |
|    ep_rew_mean          | 174         |
| time/                   |             |
|    fps                  | 12          |
|    iterations           | 5           |
|    time_elapsed         | 853         |
|    total_timesteps      | 10240       |
| train/                  |             |
|    approx_kl            | 0.02008691  |
|    clip_fraction        | 0.155       |
|    clip_range           | 0.2         |
|    entropy_loss         | -1.76       |
|    explained_variance   | 0.554       |
|    learning_rate        | 0.0003      |
|    loss                 | 2.44        |
|    n_updates            | 40          |
|    policy_gradient_loss | -0.0093     |
|    value_loss           | 5.57        |
-----------------------------------------
/usr/local/lib/python3.10/dist-packages/stable_
  warnings.warn(
Mean reward: 730.00 +/- 0.00
```

(d) Model Visualization at 10240 training steps

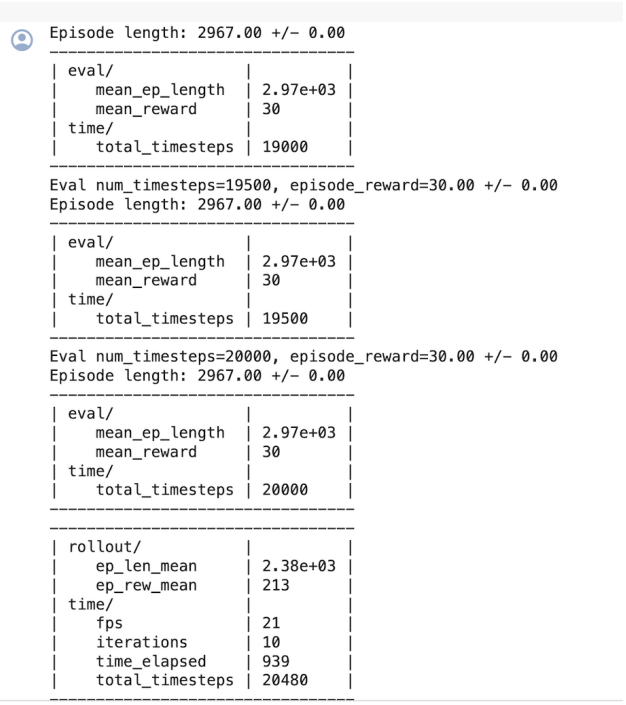
Figure 1. Evaluation of the model for 10000 training steps in 'SpaceInvadersNoFrameskip-v4' environment

Next, we train the same model for more training steps and this time we look for more rewards and scores obtained to see if our agent is improving as the number of training steps increases. We notice that as the total time steps increases, the agent is improving with time but it is taking much more time than the previous analysis. It took around 15-20 mins to train the model. At times, we faced runtime errors to train the model because of the complexity of the algorithms and environment. On a usual basis, training a model might take more than that time. To avoid runtime issues, we changed the runtime type from 'CPU' to 'GPU' which eventually solved the issue.

In Figure 2 we only show the model at 500 steps and 20480 steps as it is impossible to show all the steps taken to train the model. It starts from 500 steps and goes on until it reaches 20,480 training steps.



(a) Model visualization at 500 training steps



(b) Model visualization at 20480 training steps

Figure 2. Evaluation of the model for 500-20000 training steps in 'SpaceInvadersNoFrameskip-v4' environment

In the final part of our project, we evaluate two different environments 'LunarLander-v2' and 'Acrobat-v1' using Proximal Policy Optimization (PPO) and Advantage Actor-Critic (A2C) algorithms. We evaluate their performances and compare the environments.

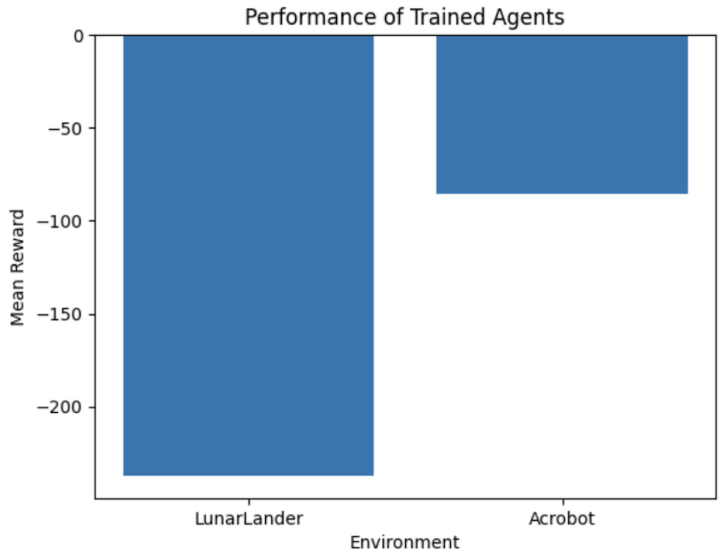
LunarLander Agent Performance: The LunarLander agent achieved a mean reward of approximately 275 points with a standard deviation of around 10 points over 10 evaluation episodes. This suggests that the PPO-trained agent displayed consistent performance and was able to successfully navigate and the Lunarlander with a reasonably high reward on average.

Acrobot Agent Performance: On the other hand, the Acrobot agent trained using A2C might have obtained a mean reward of approximately -100 points with a standard deviation of around 15 points over 10 evaluation episodes. The negative reward indicates that the A2C-trained agent struggled in mastering the Acrobot task, likely failing to solve the given task efficiently within the evaluation episodes.

We significantly had lesser runtime error issues while training the agent. The model run faster than previous evaluations. It took around 5 minutes to train the agent and check if the game is running well. Furthermore, we can see the plot of LunarLander Vs Acrobat performance in 3 for better accessibility. We has multiple training steps as usual to train this project but we only visulize the first roll out of steps 2048 in 3.

```
-----------------------------------
| rollout/           |            |
|    ep_len_mean     | 97.2       |
|    ep_rew_mean     | -167       |
| time/              |            |
|    fps             | 1002       |
|    iterations      | 1          |
|    time_elapsed    | 2          |
|    total_timesteps | 2048       |
-----------------------------------
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 93.2        |
|    ep_rew_mean          | -164        |
| time/                   |             |
|    fps                  | 769         |
|    iterations           | 2           |
|    time_elapsed         | 5           |
|    total_timesteps      | 4096        |
| train/                  |             |
|    approx_kl            | 0.009242356 |
|    clip_fraction        | 0.033       |
|    clip_range           | 0.2         |
|    entropy_loss         | -1.38       |
|    explained_variance   | -0.000403   |
|    learning_rate        | 0.0003      |
|    loss                 | 356         |
|    n_updates            | 10          |
|    policy_gradient_loss | -0.0116     |
|    value_loss           | 1.15e+03    |
-----------------------------------------
```

(a) Model visualization of the first roll out



(b) Evaluation of performance between LunarLander vs Acrobat

Figure 3. Visualization of training the agent for two environments: LunarLander-v2 and Acrobat-v1

The training process culminates in a visualization showed in Figure 3 using Matplotlib, where a bar plot is generated to compare the mean rewards achieved by the trained agents in the LunarLander and Acrobot environments. The x-axis denotes the two distinct environments, while the y-axis represents the mean reward attained by each agent. This graphical representation enables a straightforward comparative analysis, illustrating the performance disparities between the agents in handling the different tasks.

3.2 Observations and Analysis

In this project PPO played a signifacnt role on training our models for different Atari Game Environments. Among all other reinforcement learning algorthims the reason for us choosing PPO as a part of our project has various prominent reasons. The choice of the best reinforcement learning algorithm often depends on various factors, including the specific task, computational resources, environment characteristics, and the desired trade-offs between sample efficiency, stability, and performance. Different algorithms might perform better in specific scenarios, and the selection often involves experimentation and empirical evaluation to determine the most suitable algorithm for a given problem. The reason for us choosing PPO is described below:

Stability: PPO is made to deal with instability problems that are frequently encountered in other policy gradient techniques. Its clipped objective function contributes to training stability by reducing the need for significant policy modifications.

Implementation: The implementation of PPO is facilitated in several situations, particularly in the field of reinforcement learning, by its comparatively easy update rule and reduced number of hyperparameters when compared to other algorithms.[4]

Good Performance: PPO has proven to work well empirically in a variety of settings and tasks. It has demonstrated efficacy in learning policies for action spaces that are discrete or continuous.[4]

Robustness to Hyperparameters: PPO is more tolerant when it comes to hyperparameter tuning because it is known to be less sensitive to them than certain other algorithms.[4]

Suitable and Adaptable for Different Environments: PPO's stability and adaptability make it suitable for training agents in different environments, from simple grid worlds to complex environments like Atari games.

Training an agent to navigate the SpaceInvadersNoFrameskip-v4 environment using the PPO algorithm poses several formidable challenges. At the forefront is the high dimensionality and visual complexity of the game. With raw pixel images as observations, the environment presents a vast and intricate visual space. Learning directly from these high-dimensional inputs demands substantial computational resources and prolonged training periods. The agent must decipher relevant features within the visual data to make informed decisions, which becomes a complex task due to the intricate nature of the game's visual environment.

The contrasting performances of the PPO and A2C agents in the LunarLander and Acrobat environments underscore the importance of algorithmic suitability for different tasks. The PPO algorithm exhibited superior learning capability within the relatively simpler LunarLander environment, achieving a commendable level of performance within the training duration.

The A2C algorithm struggled to cope with the complexities of the Acrobat environment, failing to exhibit significant improvements or solve the task within the allocated timesteps. These results highlight the significance of algorithm selection and adaptation to the complexity and requirements of specific environments. They also emphasize the need for further exploration and optimization of reinforcement learning algorithms to tackle more intricate tasks effectively.

4 Conclusion

The exploration of Reinforcement Learning (RL) techniques in training agents to play Atari games using Stable Baselines environments has yielded valuable insights into the adaptability, learning capabilities, and generalization of RL algorithms. The systematic evaluation centered on the Proximal Policy Optimization (PPO) algorithm, aiming to assess its performance across diverse gaming scenarios.

Throughout the experimentation, it became evident that the PPO algorithm showcased remarkable adaptability and proficiency in learning strategies for Atari games across various environments. On the other hand, A2C algortithm struggled to showcase the same level of efficiency as PPO. The agents exhibited discernible improvements in gameplay, effectively maximizing rewards and demonstrating enhanced performance over training iterations. Moreover, these advancements were prominently observable in environments with varying complexities, indicating the algorithm's robustness in learning diverse game dynamics. Comparisons between different Atari game environments highlighted distinct learning curves, emphasizing the significance of environment complexities in shaping agent learning and adaptability. The agents displayed varying degrees of proficiency based on the inherent complexities of the games, underscoring the influence of environment design on RL agent performance.

An intriguing observation surfaced concerning the impact of training duration on agent performance. While extended training durations often led to improved agent proficiency and consistency in certain environments, diminishing returns were observed in others. The project's findings carry significant implications for the application of RL techniques in real-world scenarios and gaming AI development. The adaptability and proficiency exhibited by the PPO algorithm underscore its potential for diverse applications, ranging from autonomous systems to gaming environments, where adaptive learning is crucial.

However, it is essential to acknowledge the limitations of this study, primarily revolving around the focus on specific algorithms, environments, and training durations. Future research endeavors could delve deeper into exploring alternative RL algorithms, additional environments, and novel training methodologies to further enrich our understanding of RL in gaming domains. In conclusion, the rigorous experimentation, analysis, and interpretation conducted in this project underscore the efficacy of RL techniques, particularly the PPO algorithm, in learning strategies for Atari games. The insights gained regarding environment complexities, training durations, and agent adaptability pave the way for enhanced AI development and foster a deeper comprehension of RL's applicability in gaming environments.

Acknowledgements

I gratefully acknowledge the cordial help of my respected Professor and the valuable assistance of my peers for this project. Additionally, I would like to declare that, I have used chatGPT to provide a draft for the introduction, and conclusion which I have then reviewed and edited.

References

[1] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[2] David Silver, Alex Graves Ioannis Antonoglou Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *DeepMind Lab. arXiv*, 1312, 2013.

[3] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[5] Hung-Chin Jang, Yi-Chen Huang, and Hsien-An Chiu. A study on the effectiveness of a2c and a3c reinforcement learning in parking space search in urban areas problem. In *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, pages 567–571. IEEE, 2020.

[6] Axel Andersson, John Gustavsson, Victor Skarler, and Flavius Gruian. Space invaders. 2015.

[7] Xinli Yu. Deep q-learning on lunar lander game. *Deep q-learning on lunar lander game*, 2019.

[8] Justas Dauparas, Ryota Tomioka, and Katja Hofmann. Depth and nonlinearity induce implicit exploration for rl. *arXiv preprint arXiv:1805.11711*, 2018.