Comparative Study of Reinforcement Learning Methods: DQN vs PPO in a Game

Abhinav Wasukar[1], Sumitra Jakhete[2]

[1]Student, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India, abhinavwasukar01@gmail.com1
[2]Assistant Professor, SCTR's Pune Institute of Computer Technology, (IT), Pune, Maharashtra, India, sajakhete@pict.edu2

Abstract

RL a branch of machine-learning, focuses on guiding intelligent agents to maximize cumulative rewards through interactions with their environment. This paper compares two widely utilized RL algorithms: DQN and PPO. The comparison primarily evaluates their performance in game environments based on critical parameters, including learning speed, stability during training, and overall efficiency. By reviewing prior research and practical applications of these algorithms, we analyze how DQN and PPO differ in their ability to learn, maintain consistent performance, and achieve optimal outcomes. This paper sheds light on their applicability to various challenges within artificial intelligence and highlights potential areas for future exploration.

Keywords: RL, DQN, PPO, Game AI

1. Introduction

RL has become a cornerstone of artificial intelligence, significantly influencing domains such as gaming, robotics, and autonomous systems. By allowing agents to cooperate with an surrounding and study from response in the form of compensations and fines, RL provides a dynamic framework for solving complex problems. The agent develops strategies to optimize long-term rewards, making it highly applicable in scenarios requiring sequential decision-making. DRL takes RL a step further by combining it with deep neural networks[12]. This integration empowers agents to handle high-dimensional state spaces, making DRL an effective tool for solving advanced challenges. Two widely adopted DRL algorithms are DQN and PPO. While DQN a value-based system that evaluates state-action pairs to determine optimal actions, PPO employs a policy-gradient approach to directly optimize the mapping of states to actions, ensuring stable updates. In gaming, these algorithms have demonstrated significant capabilities. DQN excels in environments with discrete action spaces, where decisions are limited to a predefined set of actions. PPO, on the other hand, performs well in continuous or mixed-action environments, where actions are more flexible. This paper provides an in-depth exploration of their methodologies, highlights their comparative strengths, and evaluates their potential in diverse real-world applications.

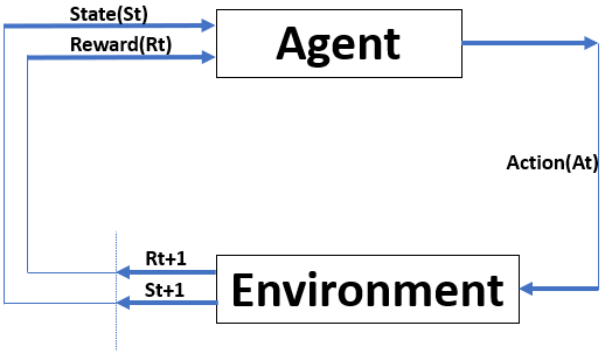

Fig. 1. Proposed Research of RL

Figure 1 illustrates the basic interaction in RL. The **agent** observes the **environment's state** by taking an **action**. The surroundings then offer a **return** and moves to a new **state**, which the mediator observes again, and the cycle continues [7][10].

Table 1. Literature survey of the researched work

Paper	Focus	Purpose	Key Points
[3]Liu et al. (2024)	Imperfect-Information Games	Solve games using a policy-gradient approach	• Iterative convergence methods • Efficient exploration in games with • hidden information
[4]Yang et al. (2024)	Spatial Prisoner's Dilemma Game	Explore Sarsa algorithm effectiveness	• Analysis of spatial dynamics • Non-stationary environment; comparison with other algorithms
[5]Bergdahl et al. (2024)	Dynamic AI for strategic decision-making in tower defense games	Enhance gameplay experience through adaptive AI using reinforcement learning	• Reward function incentivizes successful defense and resource management • RL framework outperformed traditional heuristic approaches • Hybrid RL and heuristic approach achieved a 57.12% success rate
[6]Zhang et al. 2024	Development of DRL agents	Enhance agent exercising, incorporation, and configuration with individual players	• Introduces the Shūkai system for DRL agents in fighting games • Addresses challenges of agent competence vs. player experience • Implements HELT (Heterogeneous League Training) for efficient training • Achieves robust generalizability across multiple characters • Demonstrates improved player retention rates (5% max, 4% avg)
[7]Chen et al. (2023)	Gaming Applica-tions	Examine RL application in the Breakout game	• Case study of Breakout game • Effectiveness of reinforcement learning strategies

[8]Yingyu Xu (2022)	Deep RL and Imitation Learning	Investigate RL and imitation learning in VizDoom	Advanced techniques in complex environments
[9]Albaba and Yildiz(2021)	Application of deep reinforcement learning in driver behavior modeling	Enhance understanding and prediction of driver behavior in various scenarios	• Integrates deep reinforcement learning with behavioral game theory principles • Focuses on modeling complex interactions between drivers and their environment • Utilizes simulations to evaluate driver decision-making processes • Aims to improve safety and efficiency in transportation systems
[10]Alonso et al. (2020)	Navigation in AAA Video Games	Seminar DL methods for navigation	• Use of deep reinforcement learning • Challenges in realtime navigation
[11]Xiao et al.(2019)	Provide a comprehensive overview of DRL techniques and their evolution in gaming	Analyzes various game genres and platforms used for AI research	• Discusses challenges in high-dimensional state spaces and partial observability • Identifies trends in AI for game testing, design, and creative AI development • Highlights the role of games as testbeds for exploring decision-making and learning • Addresses challenges like exploration-exploitation trade-offs and generalization • Future research directions, including the impact of large language models (LLMs)
[12]Cheng et al.(2019)	Application of reinforcement learning in gaming	Explore the effectiveness of Q-learning combined with deep neural network(DNN) in video game environments	• Proposes a novel approach integrating Q-learning with deep learning techniques • Evaluates the performance of the model in various video game scenarios • Discusses the architecture of the DNN used in the study • Analyzes the training process and the impact of different hyperparameters

			• Highlights the results achieved in terms of game performance and learning efficiency
[13]Justensen et al.(2019)	Application of deep learning techniques in gaming	Review and analyze the use of deep learning in video game playing and AI development	• Provides a comprehensive overview of deep learning methods applied to video games • Discusses various architectures, including CNNs and RNNs. • Analyzes learnings of successful implementations in different game genres • Identifies challenges such as sample efficiency, generalization, and real-time performance
[14]Ulyanov et al.(2021)	Exploration of reinforcement learning in gaming	Investigate and develop algorithms for neural networks using reinforcement learning techniques in gaming contexts	• Discusses the integration of reinforcement learning (RL) with neural networks (NN) for game AI development • Examines various RL algorithms, including Q-learning and policy gradients • Analyzes the effectiveness of actor-critic methods in complex gaming environments
[1] Souchleris et al.(2023)	Application of reinforcement learning in gaming | To provide an overview of RL in the gaming industry, identify future directions, and address challenges	Overview of current applications of RL in gaming contexts	• Discussion of advancements in RL algorithms, including deep reinforcement learning • Identification of challenges such as computational complexity and data requirements • Exploration of future research directions, including integration with other AI techniques
[2]Galway et al.(2008)	Application of ML in digital games	Survey the use of ML techniques in the advancement of game AI	• Highlights limitations of current AI methods, including predictability • Explores the potential of machine learning to create adaptive game agents • Covers various ML techniques, such as NN and RL, • Identifies challenges in implementing learning AI,

			including game balance concerns
[15] ElDahshan et al.(2022)	Application of DRL in video games	Review the advancements and applications of DRL in video games	• Discusses the fundamentals of DRL and its relevance to gaming • Analyzes various DRL algorithms and their effectiveness in game environments • Highlights case studies of successful DRL implementations in popular video games

Several studies provide direct comparisons and insights into value-based RL (like DQN) versus policy-gradient methods (like PPO). For instance, [3] focus on policy-gradient approaches in imperfect-information games, which is particularly relevant as PPO falls under this category. This study aids in understanding how PPO optimizes policies efficiently. Similarly, [4] examine the Sarsa algorithm in a non-stationary environment, offering insights into how different RL methods adapt to changing environments—something that is crucial in evaluating DQN's ability to learn in dynamic game settings.

The paper by [5] explores RL in tower defense games, comparing traditional heuristic approaches with RL-based AI, reinforcing the need for more adaptive and reward-driven learning methods like DQN and PPO. This research validates our study by showing how RL can outperform pre-programmed strategies. [6] introduce a Shūkai system for training DRL agents in fighting games, emphasizing the balance between agent competence and player experience, which aligns with our study's goal of evaluating agent learning efficiency.

[7] specifically investigates RL applications in the Breakout game, a classic discrete-action environment—a perfect setting for DQN evaluation. This study supports our research by demonstrating how RL strategies like DQN and PPO adapt to structured game challenges. Similarly, [8] explores Deep RL and Imitation Learning in VizDoom, an advanced game environment requiring complex decision-making, shedding light on how RL generalizes across different types of games.

Beyond gaming, RL applications extend to driver behavior modeling [9] and navigation in AAA video games[10]. These studies reinforce RL's adaptability and efficiency, which are key factors in comparing DQN and PPO. Moreover, [11] provides a comprehensive review of DRL techniques across game genres, discussing state-space complexity, generalization challenges, and algorithmic trends—helpful in understanding where DQN and PPO fit within the broader spectrum of RL research.

Other notable works, such as [12] and [13], focus on integrating Q-learning with deep learning techniques to improve decision-making in video games, directly relating to DQN's function. [14] explore RL integration with neural networks for game AI, while [1] analyzes advancements and challenges in RL-based game AI development. These works provide a strong theoretical foundation for comparing DQN's Q-learning approach and PPO's policy-optimization method.

2. Proposed Methods

This section delves into the methodologies for both DQN and PPO algorithms, describing the step-by-step process of applying these methods to game environments. Each approach incorporates innovative mechanisms to enhance learning and optimize performance.

A) DQN (Deep Q-Networks)

DQN leverages several key methodologies to solve reinforcement learning problems [7]:

Q-Learning: This model-free algorithm estimates the future reward for every condition-act pair. The Q-value is renewed iteratively applying the expression:

$Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r + \gamma \max(Q(s_{t'}, a_{t'})) - Q(s_t, a_t)]$ [7]

Here, γ is the discount factor, r is the immediate reward, α is the learning rate and $\max(Q(s_t', a_t'))$ represents the largest predicted estimate for the following state.

Deep Neural Networks: The Q-function is approximated using a NN that views the current condition as input-outputs Q-values for every potential actions.

Experience Replay: A buffer collects past interactions, which are then sampled randomly during training. This reduces correlations between consecutive experiences and enhances stability.

Target Network: A secondary network, periodically synchronized with the primary Q-network, is used to calculate target Q-values. This prevents oscillations throughout the drill.

D.Q-L: To moderate overestimation biases, DQN employs two networks—one selects the action, and the other evaluates its Q-value.

General Algorithm for DQN:

1. Start
2. Initialize the Q-network, target network, and replay buffer.
3. For every chapter:
   - Observe the current condition.
   - Select an action built on an ϵ\epsilon$\epsilon$-greedy policy.
   - Implement the action, receive the reward then transition to the following state.
   - Collect the occurrence in the rerun buffer.
   - Trial a tiny-batch from the buffer and renew the Q-network.
   - Periodically synchronize the objective network with the Q-network.
4. Repeat until the algorithm converges [10].
5. End

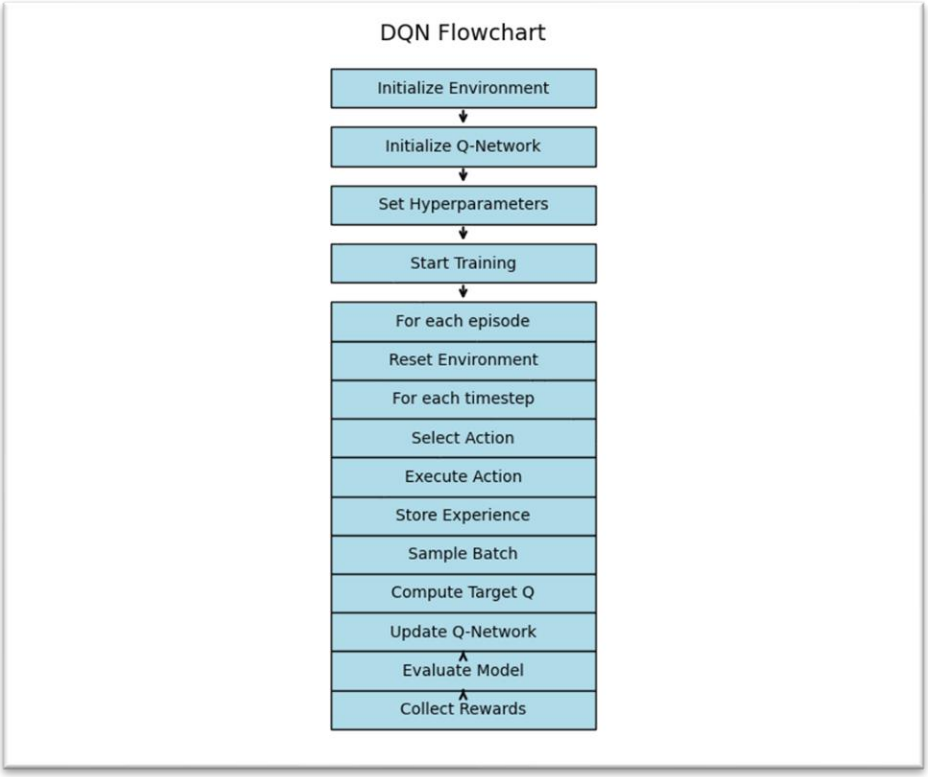

Fig. 2. DQN Flowchat

The DQN training process begins by initializing the environment, Q-network, and hyperparameters. Then, in each episode, the agent cooperates with the surrounding, selecting actions and observing the resulting rewards and next states. These occurrences are deposited in a rerun buffer. Subsequently, a set of experiences is sampled at random from the memory, and the Q-network is guided to lessen the variation between predicted and target Q-values. Periodically, the model's performance is evaluated on a separate set of episodes to track progress and assess its learning.

B) PPO (Proximal Policy Optimization)

PPO uses advanced policy-gradient methods to precisely boost an agent's policy [10]:

Policy Gradient: The agent optimizes the policy by maximizing the expected reward, calculated as:
$J(\theta)=E[\log\pi\theta(a_t|s_t)A(s_t,a_t)]$
Trust Region Optimization: Ensures updates to the policy remain within a safe range, measured by the Kullback-Leibler divergence. This prevents drastic policy changes and maintains stability.
Advantage Function: Evaluates the benefit of taking a specific action compared to the baseline value of the state:
$A(s_t,a_t)=Q(s_t,a_t)-V(s_t)$
Value Function: Predicts the expected reward for being in a given state:
$V(s_t)=E[Q(s_t,a_t)]$

General Algorithm for PPO:

1. Start
2. Prepare policy-value networks.
3. For each iteration:
    - Store paths using the existing policy.
    - Compute rewards and benefit approximations.
    - Renovate the policy by boosting a clipped alternate goal.
    - Adjust the estimate function to minimize prediction errors.
4. Repeat until the policy converges [10].
5. End

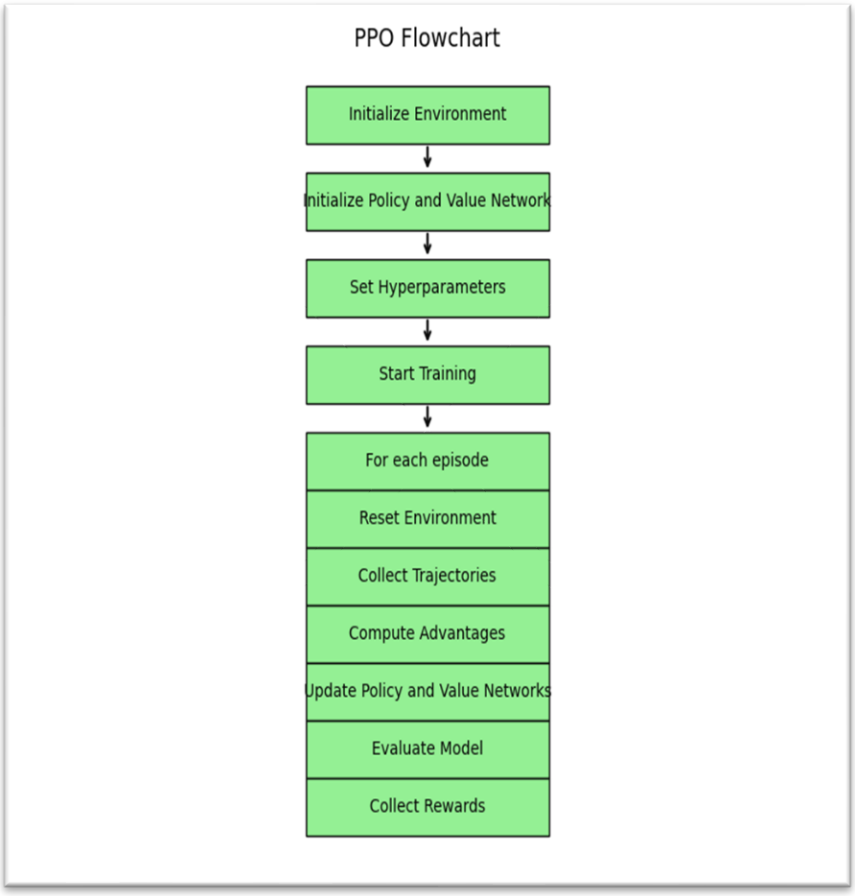

Fig. 3. PPO Flowchart

PPO starts by setting up the environment and initializing the policy and value networks. Then, it interacts with the environment, collecting data and learning from its experiences. The agent's policy is improved while carefully controlling the size of the updates to ensure stability. Finally, the agent is evaluated periodically to track its progress and see how well it's learning.

3. Results & Discussion

3.1 Implementation details

```python
import gym
import numpy as np
import matplotlib.pyplot as plt
import time
from stable_baselines3 import DQN, PPO
from stable_baselines3.common.vec_env import DummyVecEnv
from stable_baselines3.common.evaluation import evaluate_policy

# Create CartPole environment and wrap it in DummyVecEnv for compatibility
env = DummyVecEnv([lambda: gym.make('CartPole-v1')])

# Training parameters
total_timesteps = 10000

# DQN Training
start_time_dqn = time.time()
dqn_model = DQN('MlpPolicy', env, verbose=0)
dqn_model.learn(total_timesteps=total_timesteps)
dqn_training_time = time.time() - start_time_dqn

# PPO Training
start_time_ppo = time.time()
ppo_model = PPO('MlpPolicy', env, verbose=0)
ppo_model.learn(total_timesteps=total_timesteps)
ppo_training_time = time.time() - start_time_ppo

# Evaluation function compatible with VecEnv
def evaluate_model(model, env, num_episodes=100):
    episode_rewards = []
    for episode in range(num_episodes):
        done = False
        obs = env.reset()
        total_reward = 0
        while not done:
            action, _ = model.predict(obs)
            obs, reward, done, _ = env.step(action)
            total_reward += reward
            done = done.any()  # To handle VecEnv
        episode_rewards.append(total_reward)
    return episode_rewards

# Get returns for both DQN and PPO
dqn_rewards = evaluate_model(dqn_model, env)
ppo_rewards = evaluate_model(ppo_model, env)

# Calculate metrics
def calculate_metrics(rewards):
    avg_return = np.mean(rewards)
    cum_return = np.sum(rewards)
    std_dev_return = np.std(rewards)
    return avg_return, cum_return, std_dev_return

dqn_avg_return, dqn_cum_return, dqn_std_dev_return = calculate_metrics(dqn_rewards)
ppo_avg_return, ppo_cum_return, ppo_std_dev_return = calculate_metrics(ppo_rewards)
```

Fig. 4. Implemented Code of DQN vs PPO in a CartPole Environment

The performance of DQN and PPO was evaluated using the CartPole-v1 environment. Both algorithms were trained for 10,000 timesteps, and their efficiency was measured through several metrics: average return, cumulative return, standard deviation of return, and convergence time.

- DQN Results:
  Average Return: 9.54
  Cumulative Return: 954.0
  Standard Deviation of Return: 1.32
  Convergence Time: 5.59 seconds

- PPO Results:
  Average Return: 160.02
  Cumulative Return: 16002.0
  Standard Deviation of Return: 64.60
  Convergence Time: 6.13 seconds

These results highlight the distinct advantages and trade-offs between the two algorithms. While PPO demonstrates higher performance in terms of return metrics, DQN is faster to converge, making it better suited for scenarios requiring quick learning cycles.

3.2 Equations Key Final Metrics

A) Average Return: The mean return over a specified period.
Formula: $R_{avg} = (1/N) * \Sigma(R_t)$

Definition: It measures the average yield from an investment over the specified timeframe.

Average return is the mean reward obtained by the agent over multiple episodes. It measures how well the RL model is maximizing rewards per episode.

Where,

- Rt = Reward obtained in episode t
- N = Total number of episodes

B) Cumulative Return: The total change in investment value over a period, expressed as a percentage.
Formula: $R_{cum} = ((V_f - V_0) / V_0) * 100\%$

Definition: It reflects the overall growth (or decline) of an investment's value from the initial amount V0 to its final value Vf.

Cumulative return represents the total sum of rewards collected by the agent throughout all episodes. This parameter provides a broader view of how efficiently an algorithm accumulates rewards over time.

C) Standard Deviation of Return: A measure of the variability or dispersion of returns.
Formula: $\sigma = \sqrt{[(1 / (N - 1)) * \Sigma(R_t - R_{avg})^2]}$

Definition: Indicates how much the returns deviate from the average return; a higher standard deviation signifies greater volatility.

Standard deviation measures the variability or consistency of the rewards obtained by the agent across different episodes. A low standard deviation means the model produces stable and predictable results, while a high standard deviation indicates fluctuating performance.

Where,

- R_{avg} = Average return
- Rt = Return in episode t
- N = Number of episodes

D) Convergence Time: The amount of repetitions essential to attain a constant result. Formula: $T_{conv} = \min\{k: |V_k - V_{k-1}| < \varepsilon\}$

Definition: It determines how quickly a procedure approaches a final or stable solution based on a predefined threshold $\epsilon\epsilon$.

Convergence time refers to the number of iterations required for the model to reach a stable policy where learning improvements become minimal. It measures how quickly an RL algorithm learns an optimal strategy.

Where,

- Vk = Value function at iteration k
- ϵ = Convergence threshold

3.3 Tables and Figures

Table 2. Average Return DQN vs PPO

Algorithm	Average Return
• DQN	• 9.54
• PPO	• 160.02

Table 3. Cumulative Return DQN vs PPO

Algorithm	Cumulative Return
• DQN	• 954.0
• PPO	• 16002.0

Table 4. Standard Deviation of Return DQN vs PPO

Algorithm	Standard Deviation of Return
• DQN	• 1.32
• PPO	• 64.60

Table 5. Convergence time DQN vs PPO

Algorithm	Convergence Time
• DQN	• 5.59
• PPO	• 6.13

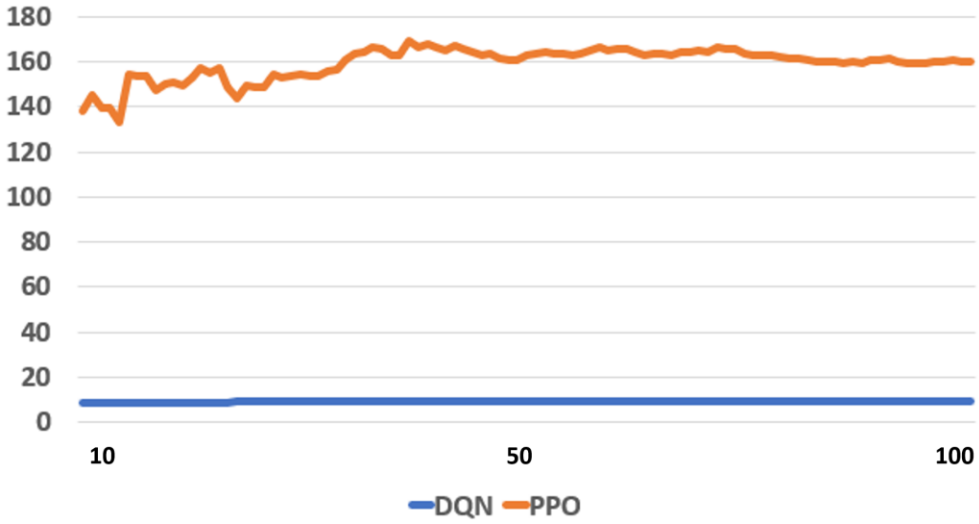

Fig. 5. shows the graph of Average returns of both DQN and PPO over 100 episodes.

The average return graph demonstrates that PPO consistently outperforms DQN in terms of reward accumulation per episode. PPO's higher average return suggests that its policy-gradient approach enables it to learn more efficient strategies and maximize rewards over time. In contrast, DQN,

which relies on Q-learning, shows a more gradual improvement, indicating that it may take longer to reach optimal performance.

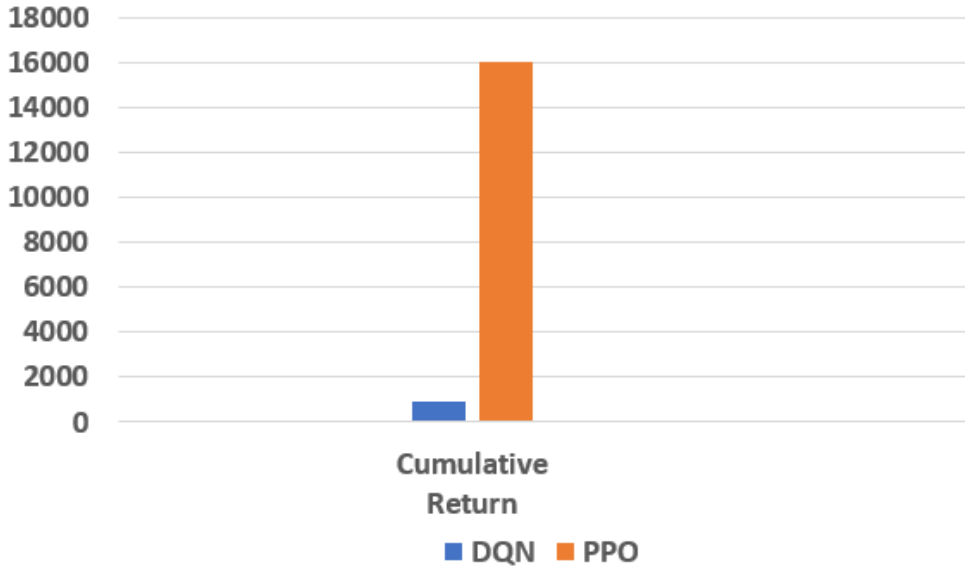

Fig. 6. Cumulative return for DQN and PPO. Here, PPO's return is more than DQN's return over 100 episodes.

The cumulative return graph emphasizes the superiority of PPO in terms of total rewards collected over multiple episodes. The significant difference between PPO's higher cumulative return and DQN's lower total reward suggests that PPO is more effective at maintaining and compounding rewards over extended training sessions.

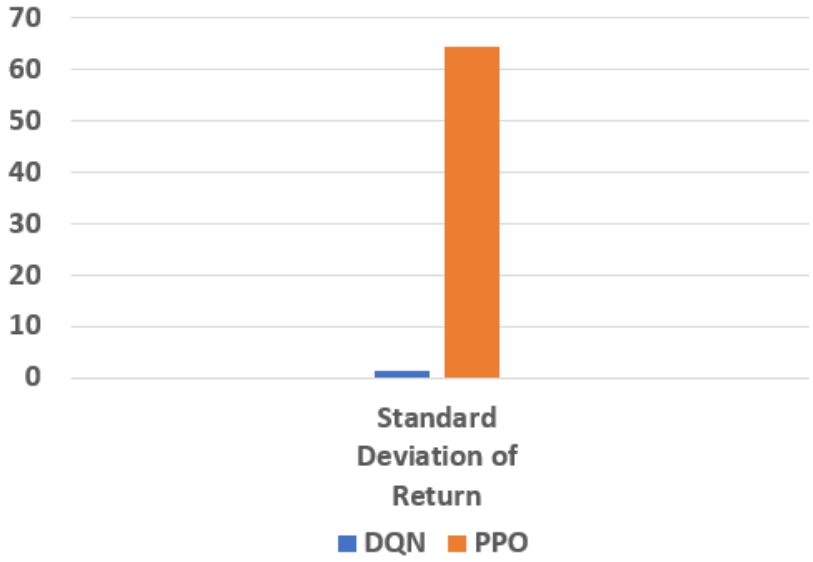

Fig. 7. shows DQN's std. deviation is lower than PPO**'s s**td. deviation over 100 episodes.

The fluctuations in PPO's results indicate that its learning process introduces more variability between episodes, making it less predictable than DQN. In contrast, DQN maintains a lower standard deviation, suggesting that it provides more stable and consistent performance across episodes

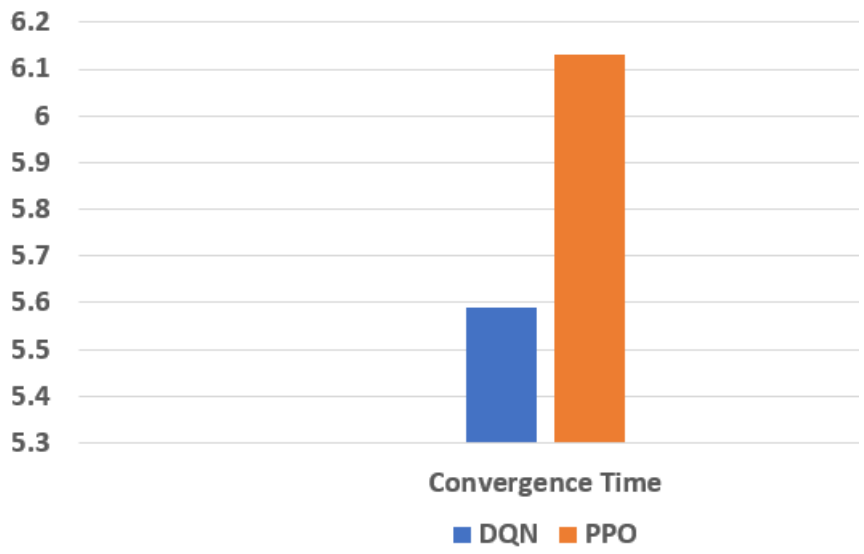

Fig. 8. Illustrates DQN convergence time faster than PPO over 100 episodes.

The convergence time graph provides insights into the computational efficiency of each algorithm. DQN converges faster than PPO, meaning it learns an optimal policy in fewer iterations. PPO, despite its better reward performance, takes longer to stabilize, suggesting that its policy updates require more fine-tuning before achieving reliable outcomes.

4. Conclusion

This study compared two RL algorithms, DQN and PPO, in a simulated game environment.

Key findings include:

Reward Acquisition: PPO significantly outperformed DQN in terms of the rewards earned. PPO achieved a much higher average return of 160.02 and a cumulative return of 16002.0, demonstrating its stronger ability to consistently maximize rewards over time. In contrast, DQN only managed an average return of 9.54 and a cumulative return of 954.0, a substantially lower performance. This result aligns with existing research suggesting that policy-based methods like PPO are better suited for environments requiring dynamic decision-making.

Training Speed: DQN showed a slight edge in terms of training time, converging in approximately 5.59 seconds. PPO took a little longer, around 6.13 seconds to train. This suggests DQN might be preferable when rapid adaptation or learning speed is paramount.

Stability: While PPO achieved considerably higher rewards, it also exhibited greater variability in its performance. PPO had a standard deviation of 64.60, indicating a wider range of outcomes across different runs. DQN, on the other hand, displayed much more consistent performance with a standard deviation of only 1.32. This implies that while DQN might not reach the same high rewards as PPO, its performance is much more predictable and reliable. This result aligns with the fundamental differences between value-based (DQN) and policy-based (PPO) learning—DQN updates discrete Q-values, whereas PPO continuously refines its policy, making training more computationally intensive.

In essence, these results show an exchange between reward maximization and stability. PPO is like a high-risk, high-reward strategy, capable of achieving much greater success but also more prone to fluctuations. DQN is more like a steady, reliable approach, consistently delivering modest results with minimal variation.

Looking ahead, there are several promising avenues for future research:

Real-World Applications: Testing these algorithms in practical scenarios like robotics, autonomous systems, and even financial modeling is crucial to understand their real-world applicability. This would involve adapting the algorithms to handle the complexities and uncertainties inherent in real-world data and environments.

Advanced Techniques: Integrating advanced techniques like meta-learning (learning how to learn) and transfer learning (leveraging knowledge from previous tasks) could further boost the performance and efficiency of both DQN and PPO. These techniques could enable faster learning, better generalization to new situations, and more efficient use of computational resources.

Deeper Analysis: Further investigation into the strengths and limitations of both algorithms is warranted. This includes exploring their performance in diverse environments, identifying potential biases, and understanding the factors that influence their behavior.

References

[1] Sidiropoulos GK, Souchleris K, Papakostas GA. Reinforcement learning in game industry—review, prospects and challenges. Applied Sciences. 2023 Feb 14;13(4):2443.

[2] Charles, D., Galway, L. & Black, M. Machine learning in digital games: a survey. Artif Intell Rev 29, 123–161 (2008). https://doi.org/10.1007/s10462-009-9112-y

[3] Liu M, Farina G, Ozdaglar A. A Policy-Gradient Approach to Solving ImperfectInformation Games with Iterate Convergence. arXiv preprint arXiv:2408.00751. 2024 Aug1.

[4] Yang L, Jiang D, Guo F, Fu M. The State-Action-Reward-State-Action Algorithm in Spatial Prisoner's Dilemma Game. arXiv preprint arXiv:2406.17326. 2024 Jun 25.

[5] Sestini A, Bergdahl J, Gisslen L. Reinforcement Learning for High-Level Strategic Control´ in Tower Defense Games. arXiv preprint arXiv:2406.07980. 2024 Jun 12.

[6] Zhang C, He Q, Yuan Z, Liu ES, Wang H, Zhao J, Wang Y. Advancing DRL Agents in Commercial Fighting Games: Training, Integration, and Agent- Human Alignment. arXiv preprint arXiv:2406.01103. 2024 Jun 3.

[7] Dewan T, Chen A, Trivedi M, Jiang D, Aditya A, Mohammed S. The Use of Reinforcement Learning in Gaming The Breakout Game Case Seminar. Authorea Preprints. 2023 Oct 30.

[8] Yingyu Xu. Deep Reinforcement Learning and Imitation Learning Based on VizDoom. Oct. 2022, doi:10.1145/3573428.3573729.

[9] Albaba BM, Yildiz Y. Driver modeling through deep reinforcement learning and behavioral game theory. IEEE Transactions on Control Systems Technology. 2021 May 5;30(2):88592.

[10] Peter M, Alonso E, Goumard D, Romoff J. Deep reinforcement learning for navigation in AAA video games. arXiv preprint arXiv:2011.04764. 2020 Nov 9.

[11] Tang Z, Shao K, Zhu Y, Li N, Zhao D. A survey of deep reinforcement learning in video games. arXiv preprint arXiv:1912.10944. 2019 Dec 23.

[12] ChengJian Lin, Hsueh-Yi Lin, Jyun-Yu Jhang, Chin-Ling Lee, KuuYoung Young. Using a Reinforcement Q-Learning-Based Deep Neural Network for Playing Video Games electronics. 2019 Oct;8(10):1128–1128

[13] Bontrager P, Justesen N, Togelius J, Risi S. Deep learning for video game playing. IEEE Transactions on Games. 2019 Feb 13;12(1):1-20.

[14] Savelyev D, Ulyanov D. Development and research of learning algorithms for neural networks with reinforcement in the gaming industry. In2021 International Conference on Information Technology and Nanotechnology (ITNT) 2021 Sep 20 (pp. 1-6). IEEE.

[15] Farouk H, ElDahshan KA, Mofreh E. Deep reinforcement learning based video games: A review. In2022 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC) 2022 May 8 (pp. 302-309). IEEE.