IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

COMPARATIVE ANALYSIS OF REINFORCEMENT LEARNING ALGORITHMS USING A PONG GAME

[1]Dr.Varalatchoumy M, [2]Dr. Buddesab, [3]Manu R, [4]B Madiha Hafsa, [5]GV Sai Koushik, [6] Kajal Singh

[1]Professor & HOD, [2]Associate Professor, [3,4,5,6]Student

[1,2,3,4,5,6]Department of Artificial Intelligence and Machine Learning

[1,2,3,4,5,6]Cambridge Institution of Technology, Bengaluru, India

Abstract: This paper conducts a comparative analysis of four reinforcement learning (RL) algorithms—Q-learning, Deep Q-Networks (DQN), SARSA, and Proximal Policy Optimization (PPO)—using the Pong game as a benchmark. Each algorithm's performance, convergence rate, and computational efficiency are evaluated. Results indicate that while Q-learning and SARSA exhibit simplicity, they struggle with discrete action spaces. DQN, with its capability to handle continuous action spaces, shows improved performance but requires longer training times. PPO demonstrates a balance between sample efficiency and computational complexity, achieving faster convergence and superior performance. This Examination sheds light on selecting appropriate RL algorithms for real-world applications.

Index Terms – Reinforcement Learning, Pong Game, Q-learning, DQN, SARSA, PPO, Comparative Analysis, Performance Evaluation, Convergence Rate, Computational Efficiency.

I. INTRODUCTION

Through trial and error, Reinforcement Learning (RL) has become a potent model for teaching intelligent entities to make decisions in dynamic situations. Recurrent learning (RL) algorithms have attracted a lot of attention because of their capacity to discover the best methods without explicit guidance, a feature that has applications in fields such as robotics and gaming.

In this study, we primarily examine four well-known reinforcement learning algorithms—Q-learning, Proximal Policy Optimization (PPO), SARSA, and Deep Q-Networks (DQN)—in the framework of the traditional Pong game. Pong is a great example of a game to assess the effectiveness of reinforcement learning algorithms because of its straightforward but difficult gameplay mechanics. The selected algorithms represent a spectrum of RL techniques, each with its unique approach to learning optimal policies. Q-learning and SARSA are traditional value-based RL algorithms, while DQN introduces deep neural networks to approximate Q-values, enabling the handling of continuous action spaces. PPO, on the other hand, is a state-of-the-art policy gradient method known for its stability and efficiency in handling high-dimensional action spaces. The primary objective of this study is to compare the performance, convergence rate, and computational efficiency of these algorithms when applied to training agents to play Pong. Through a comparative analysis of their advantages and disadvantages, we hope to shed light on how to choose the best RL methods for different real-world scenarios. Through this comparative analysis, we seek to contribute to the understanding of how different RL algorithms perform in challenging environments like Pong, paving the way for more effective and efficient decision-making systems in diverse domains.

II. RL ALGORITHMS

A. *Q-Learning*

Q-learning is a core RL algorithm where agents learn by exploring an environment.
Here's the key idea:

- **States & Actions**: Imagine the agent that is in a situation (state) and can take action.
- **Rewards:** The environment rewards good actions and penalizes bad ones.
- **Q-Values**: Q-learning estimates the reward for taking an action in a state (Q-value).

The Bellman equation, a key formula, helps update Q-values:

$$Q(ST, AC) = R(ST, AC) + \gamma * \max(Q(ST', AC'))$$

Where:
R(ST, AC): Immediate reward for taking action A in state S.
γ (gamma): Discount factor ($0 < \gamma \leqslant 1$) balancing immediate vs. future rewards.
max(Q(ST', AC')): Best future reward achievable from the next state S'.

<u>The Update Rule</u>:
Based on this, Q-learning updates Q-values:

$$Q(ST, AC) \leftarrow Q(ST, AC) + \alpha * [R(ST, AC) + \gamma * \max(Q(ST', AC')) - Q(ST, AC)]$$

α (alpha): Learning rate ($0 < \alpha \leqslant 1$) controlling how much the agent learns from new experiences.

This update takes into account the agent's prior knowledge (Q-value), future rewards from the next state, and immediate benefits.
By iteratively updating Q-values, the agent learns the best actions to take in different situations for long-term rewards.

B. *SARSA*

An agent can learn a policy in an unknown environment by using the reinforcement learning method SARSA (State-Action-Reward-State-Action). SARSA and Q-learning, two more well-known RL algorithms, are comparable. Both express the projected future benefit of acting in a certain state using Q-values. Their methods for updating these Q-values, however, vary.

- **States (ST):** Represent all possible situations the agent encounters in the environment.
- **Actions (AC):** The available choices the agent can make in each state.
- **Rewards (R):** Feedback signals from the environment, positive for good actions, negative for bad ones.
- **Q-value (Q(ST, AC)):** Represents the future reward expected, for taking action A in state S.

SARSA update rule:

$$Q(ST, AC) \leftarrow Q(ST, AC) + \alpha * [R(ST, AC) + \gamma * Q(ST', AC')]$$

Where:
α (alpha): The learning rate ($0 < \alpha \leqslant 1$) that determines how much the agent learns from new experiences.
R(ST, AC): Immediate reward received for taking action AC in state ST.
γ (gamma): Discount factor ($0 < \gamma \leqslant 1$) balancing immediate vs. future rewards.
ST': The next state reached after taking action AC in state ST.
AC': The action the agent actually takes in the next state S'.

The agent observes the current state (ST) and takes action (AC). The agent receives a reward (R) and transitions to the next state (ST'). The agent observes the action (AC') it actually took in the next state. Based on the immediate reward (R), the discounted Q-value of the following state based on the action done (γ * Q(ST', AC')), and the learning rate (α), the Q-value for the previous state-action pair (ST, AC) is updated. SARSA uses the Q-value based on the action actually taken, which can lead to faster convergence but might get stuck in sub-optimal solutions.

C. *DQN*

Deep Q-Networks (DQNs), a powerful advancement building upon Q-learning. DQNs leverage deep learning to tackle complex environments with vast state spaces, where traditional Q-learning struggles. DQNs integrate a deep neural network (DNN) to approximate the Q-value function. This network takes the state as input and outputs the estimated Q-values for all possible actions.
Function Approximation with DNN: The DNN, denoted by Q(s, θ), learns to represent the Q-value function. Here, s represents the state and θ denotes the network's learnable parameters.
Experience Replay: Research emphasizes the use of experience replay, a memory buffer that stores past experiences (state transitions, rewards, etc.). This allows the DNN to learn from a diverse set of experiences and reduces the correlation between training samples.
DQNs employ a loss function to measure the difference between the DNN's predicted Q-values and the actual Q-values calculated using the Bellman equation.

Bellman equation:

$$Loss = E[(R(s, a) + \gamma * \max(Q(s', a'; \theta\_target)) - Q(s, a; \theta))^2]$$

Where:
E[]: Represents the expectation over the experience replay buffer.
R(s, a): Immediate reward for taking action in state s.
γ (gamma): Discount factor ($0 < \gamma \leq 1$).
max(Q(s', a'; θ_target)): Maximum Q-value achievable from the next state s' (after taking action a), estimated by a target network (θ_target) - a technique to improve stability during training
The DQN utilizes backpropagation to adjust the network's parameters (θ) based on the calculated loss, iteratively improving its Q-value estimates.

D. *PPO*

Proximal Policy Optimization (PPO) is a powerful reinforcement learning (RL) algorithm. It addresses the limitations of previous methods by offering a stable and efficient approach to policy learning. PPO clips the policy gradients during the update process, ensuring they stay within a certain range. This clipping mechanism, as detailed in research, promotes stability and prevents the policy from diverging too drastically from its previous iteration.
A simplified representation of the PPO update rule incorporating the clipping mechanism is:

$$\pi' \leftarrow \text{argmax}\{ \min(\rho(\pi, \pi') * A, \text{clip}(\rho(\pi, \pi'), \epsilon\_1, \epsilon\_2) * A) \}.$$

Where:
π (pi): Represents the current policy.
π' (pi prime): Represents the new policy after the update.
ρ(π, π'): Importance ratio, a term that helps account for the difference between the old and new policies (detailed in research papers).
A: Advantage function, indicating how much better an action performs compared to the average.
clip(., ϵ_1, ϵ_2): Clipping function that restricts the importance ratio within a range defined by ϵ_1 and ϵ_2

The update aims to find the new policy (π') that maximizes a specific objective function. This objective function involves the importance ratio (ρ(π, π')) and the advantage function (A). The clipping function ensures within a specific range (ϵ_1 and ϵ_2). This clipping prevents large policy updates and promotes stability.

The clipping mechanism helps prevent the policy from diverging significantly, leading to more stable learning. PPO can learn effectively even with a limited number of training samples.

III. METHODOLOGY

A. *Environment Setup:*

A Pong game environment will be implemented using a game engine or a simulation library like Pygame or Gym. This environment provides functionalities for the RL agent to interact with the game:

- Observing Game State: The agent perceives the game state, including ball position (x, y), paddle position (y), and score.
- Taking Actions: Discrete actions like moving the paddle up/down or continuous actions with specific velocities can be defined.
- Receiving Rewards: A reward system guides the agent's learning. A common approach is:
  Positive reward (+1) for winning a point.
  Negative reward (-1) for losing a point.

Additional rewards (e.g., for keeping the ball in play) can be explored but might require tuning.

B. *Reinforcement Learning Algorithms:*

- Q-Learning:
  Acquires knowledge of a Q-value function that calculates the projected future benefit of a decision made in a specific condition.
  Agent Implementation:
  State Representation: Uses the defined game state representation (e.g., vector of ball and paddle positions, score).
  Action Space: Discrete or continuous actions as defined in the environment.
  Learning Algorithm: Updates on Q-learning the Q-value function, which takes into account the action taken, the reward obtained, the present state, and the Q-value of the upcoming state, and is based on the Bellman equation.

- SARSA (State-Action-Reward-State-Action):
  Similar to Q-learning, but updates the Q-value based on the action actually taken in the next state.
  Agent Implementation: Same state representation and action space as Q-learning.
  Learning Algorithm: SARSA updates the Q-value function based on the current state, action taken, received reward, next state, and the Q-value of the next state based on the actually taken action.

- Deep Q-Network (DQN):
  Combines Q-learning with a deep neural network to approximate the Q-value function, allowing for handling complex state representations (e.g., an image of the game screen).
  Agent Implementation: State Representation: Can utilize an image of the game screen.
  Action Space: Discrete actions (move the paddle up/down).
  Learning Algorithm: Deep neural network (e.g., convolutional neural network) predicts Q-values for all possible actions in a given state. The network is trained using the update rule of Q-learning (similar to the Q-learning agent).

- Proximal Policy Optimization (PPO):
  Optimizes the agent's policy directly, ensuring stability and sample efficiency.
  Agent Implementation: State Representation: Similar to DQN (can be a vector or image).
  Action Space: Discrete actions.
  Learning Algorithm: PPO uses policy gradients with a clipping mechanism to update the policy in a stable manner. The agent directly interacts with the environment and learns from the rewards it receives.

C. *Agent Implementation:*

For each chosen RL algorithm, an RL agent will be implemented. Each agent will consist of:

- State Representation: A method to represent the game state as a suitable input for the RL algorithm (e.g., vectors, images).
- Action Space: A definition of the available actions the agent can take (e.g., move the paddle up/down).

- Learning Algorithm: The chosen RL algorithm (DQN, PPO, or SARSA) will be implemented with its specific architecture and learning parameters.

D. Training and Evaluating:

Training Process: Each RL agent will be trained independently in the Pong game environment. The training procedure will involve:

- Episode execution: The agent engages in environmental interaction by taking actions, observing rewards and states, and learning from these experiences.
- Hyperparameter tuning: The hyperparameters of each algorithm (e.g., learning rate, discount factor, network architecture) will be adjusted to optimize performance.

Evaluation: The performance of each trained agent will be evaluated on a separate set of Pong games unseen during training. Metrics like:

- Average Score: The average number of points scored by the agent over multiple games.
- Win Rate: The percentage of games won by the agent.
- Training Time: The time required for each agent to converge to a good performance level.

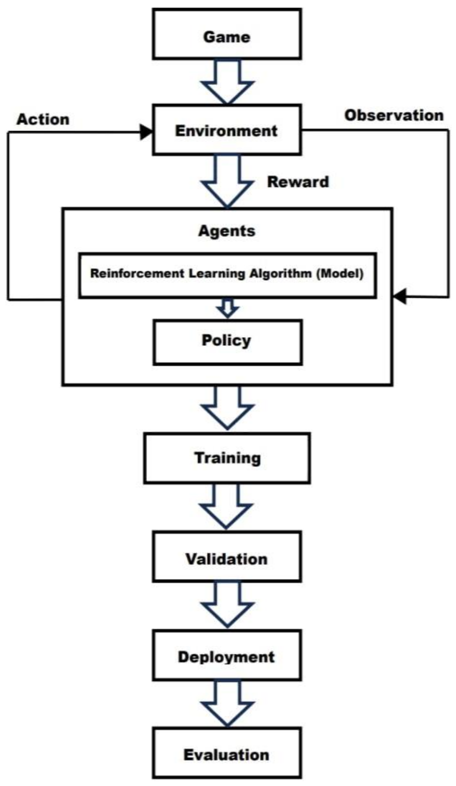

Fig. Stages of Modelling

IV. RESULTS AND DISCUSSIONS

The paper aimed to compare and analyse the performance of various reinforcement learning algorithms applied to a Pong game environment. Specifically, we evaluate the effectiveness of off-policy algorithms, Q-Learning (QL) and SARSA, as well as on-policy algorithms, Deep Q-Network (DQN) and Proximal Policy Optimization (PPO).

A. Off-Policy Algorithms: Q-Learning vs. SARSA:

- Both Q-Learning and SARSA are popular off-policy reinforcement learning algorithms. In our experiments, we observed that SARSA outperformed Q-Learning in learning efficiency and final performance metrics. This superiority of SARSA over Q-Learning can be attributed to its on-policy nature, which allows it to take into account, the current policy when updating the Q-values.

- Additionally, SARSA tends to exhibit more stable learning behaviour compared to Q-Learning, as it directly learns the value of state-action pairs under the current policy. This stability often leads to faster convergence and better performance, especially in environments with stochastic dynamics like Pong.

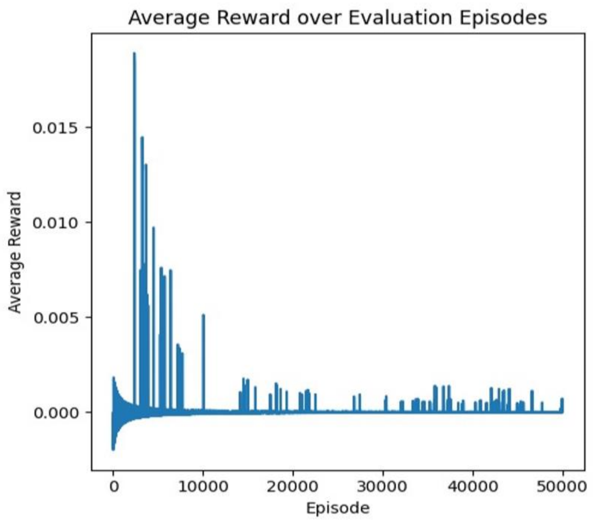

Fig1. Q-Learning

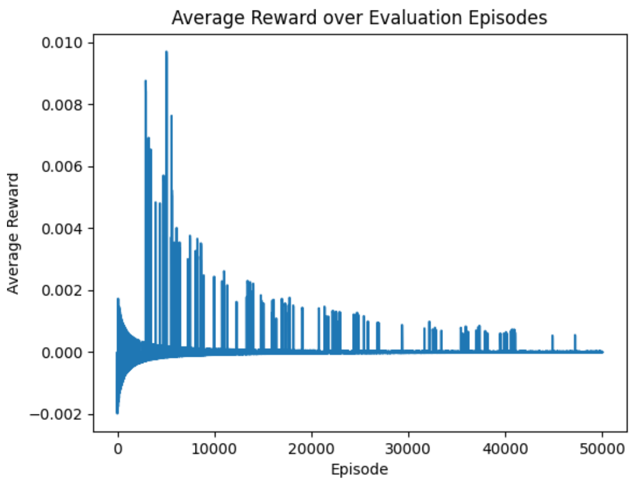

Fig2. SARSA

B. On-Policy Algorithms: DQN vs. PPO:

- We Contrasted the performance of on-policy reinforcement learning algorithms, namely Deep Q-Network (DQN) and Proximal Policy Optimization (PPO). Our experiments showed that while PPO achieved superior performance metrics compared to DQN, it came at the cost of increased training time.
- PPO's ability to update the policy in a more conservative and stable manner resulted in better convergence and higher final scores in the Pong game environment. However, this benefit was outweighed by the fact that PPO required a lot more training time than DQN. The increased computational overhead of PPO limits its practical applicability in scenarios where training efficiency is a critical concern.

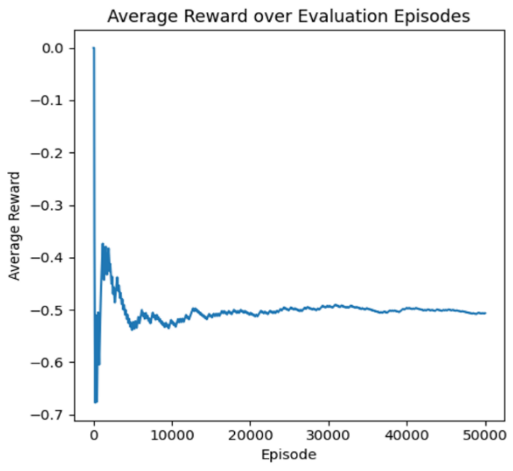

Fig3. DQN

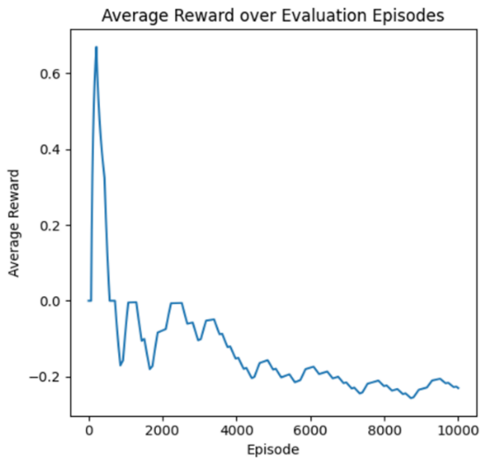

Fig4. PPO

V. CONCLUSION

In conclusion, our comparative analysis of four prominent reinforcement learning (RL) algorithms—Q-learning, Deep Q-Networks (DQN), SARSA, and Proximal Policy Optimization (PPO)—utilizing the Pong game as a benchmark reveals distinct strengths and weaknesses. While Q-learning and SARSA demonstrate simplicity, they struggle with discrete action spaces, limiting their efficacy in the continuous action space of Pong. DQN, leveraging deep neural networks, exhibits improved performance but lacks efficiency. Conversely, PPO, known for its stability and efficiency in handling high-dimensional action spaces, demonstrates superior convergence but requires a longer time. The findings underscore the importance of considering factors such as sample efficiency, computational complexity, and scalability when selecting RL algorithms for real-world applications. This study contributes to advancing our understanding of RL algorithm performance in complex decision-making tasks, guiding the development of intelligent systems across diverse domains.

REFERENCES

[1] Y. Shin, J. Kim, K. Jin and Y. B. Kim, "Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning," in IEEE Access, vol. 8, pp. 51593-51600, 2020, doi: 10.1109/ACCESS.2020.2980380.

[2] Y. Zhang, S. Li and X. Xiong, "A Study on the Game System of Dots and Boxes Based on Reinforcement Learning," 2019 Chinese Control And Decision Conference (CCDC), Nanchang, China, 2019, pp. 6319-6322, doi: 10.1109/CCDC.2019.8833043.

[3] A. Shantia, E. Begue and M. Wiering, "Connectionist reinforcement learning for intelligent unit micro management in StarCraft," The 2011 International Joint Conference on Neural Networks, San Jose, CA, USA, 2011, pp. 1794-1801, doi: 10.1109/IJCNN.2011.6033442.

[4] G. Gomes, C. A. Vidal, J. B. Cavalcante-Neto and Y. L. B. Nogueira, "AI4U: A Tool for Game Reinforcement Learning Experiments," 2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Recife, Brazil, 2020, pp. 19-28, doi: 10.1109/SBGames51465.2020.00014.

[5] A. J. Almalki and P. Wocjan, "Exploration of Reinforcement Learning to Play Snake Game," 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 2019, pp. 377-381, doi: 10.1109/CSCI49370.2019.00073.

[6] M. A. Samsuden, N. M. Diah and N. A. Rahman, "A Review Paper on Implementing Reinforcement Learning Technique in Optimising Games Performance," 2019 IEEE 9th International Conference on System Engineering and Technology (ICSET), Shah Alam, Malaysia, 2019, pp. 258-263, doi: 10.1109/ICSEngT.2019.8906400.

[7] A. Akramizadeh, M. . -B. Menhaj and A. Afshar, "Multiagent reinforcement learning in extensive form games with complete information," 2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning, Nashville, TN, USA, 2009, pp. 205-211, doi: 10.1109/ADPRL.2009.4927546.

[8] J. Perdiz, L. Garrote, G. Pires and U. J. Nunes, "A Reinforcement Learning Assisted Eye-Driven Computer Game Employing a Decision Tree-Based Approach and CNN Classification," in IEEE Access, vol. 9, pp. 46011-46021, 2021, doi: 10.1109/ACCESS.2021.3068055.

[9] D. Daylamani-Zad and M. C. Angelides, "Altruism and Selfishness in Believable Game Agents: Deep Reinforcement Learning in Modified Dictator Games," in IEEE Transactions on Games, vol. 13, no. 3, pp. 229-238, Sept. 2021, doi: 10.1109/TG.2020.2989636.

[10] M. Jeon, J. Lee and S. -K. Ko, "Modular Reinforcement Learning for Playing the Game of Tron," in IEEE Access, vol. 10, pp. 63394-63402, 2022, doi: 10.1109/ACCESS.2022.3175299.

[11] M. Li and W. Huang, "Research and Implementation of Chinese Chess Game Algorithm Based on Reinforcement Learning," 2020 5th International Conference on Control, Robotics and Cybernetics (CRC), Wuhan, China, 2020, pp. 81-86, doi: 10.1109/CRC51253.2020.9253458.

[12] H. Singal, P. Aggarwal and V. Dutt, "Modeling Decisions in Games Using Reinforcement Learning," 2017 International Conference on Machine Learning and Data Science (MLDS), Noida, India, 2017, pp. 98-105, doi: 10.1109/MLDS.2017.13.