Experimental Analysis of DQN and DDQN in Multi-Agent Reinforcement Learning for Tic-Tac-Toe and Connect Four

Your Name

Date of Report

Abstract

This report presents an experimental analysis of Deep Q-Network (DQN) and Double DQN (DDQN) algorithms in a multi-agent reinforcement learning setting, specifically applied to Tic-Tac-Toe and Connect Four. The study evaluates algorithms' performance improvement through self-play and shows that this method retains the models' level of generalization

1 Introduction

1.1 Background

The landscape of Artificial Intelligence (AI) has been dramatically reshaped in recent years, particularly through the advancements in Reinforcement Learning (RL). RL's emergence as a pivotal technique in AI has been most prominent in the realm of strategic decision-making and game-playing. A crucial development within RL is Multi-Agent Reinforcement Learning (MARL), where the interaction of multiple decision-makers creates complex dynamic environments.

Evolution of Deep Learning in RL The integration of deep learning with RL, especially through algorithms like Deep Q-Networks (DQN) [4] and its variant Double DQN (DDQN) [7], marks a significant milestone. DQN, introduced by Mnih et al., showcased the potential of deep neural networks in approximating Q-values, revolutionizing the way RL handles high-dimensional state spaces. Its extension, DDQN, further refined this approach by tackling the overestimation bias of Q-values in DQN, leading to more stable learning in complex environments.

Multi-Agent Reinforcement Learning (MARL) MARL [1] extends traditional RL to environments where multiple agents interact simultaneously. This interaction can be either cooperative, competitive, or a mixture of both. In MARL, each agent learns to maximize its own expected reward in the presence of other agents, whose actions also affect the environment's state. Applications of MARL are found in various fields, including robotics, autonomous vehicle coordination, and strategic games.

2 Related Works

DQN and DDQN in Multi-Agent Settings Our project directly builds upon the principles of Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN). By adapting DQN for multi-agent environments, we explore how agents learn and interact within shared spaces, a key aspect of MARL. DDQN's role in addressing Q-learning's overestimation bias is particularly crucial in our project, as it ensures more accurate decision-making and strategy development in the complex dynamics of games like Tic-Tac-Toe and Connect Four.

AlphaZero's Influence on Self-Play Strategies A significant inspiration for our approach is derived from AlphaZero's [5] groundbreaking methodology in learning through self-play. By adopting a similar strategy by first training on an random policy opponent, our project aims to investigate the autonomous strategic evolution of agents in MARL. This adaptation of AlphaZero's self-play methodology allows us to examine the agents' capability to develop sophisticated strategies from scratch in a more simple structured board game environment.

In essence, the methodologies and advancements represented by DQN, DDQN, and AlphaZero's self-play approach form the foundation of our exploration into MARL. Their collective integration into our project provides a nuanced understanding of agent behavior and strategic development within interactive and competitive game settings.

2.1 Objective

The primary objective of this experiment is to explore and compare the effectiveness of Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) within the framework of Multi-Agent Reinforcement Learning (MARL), specifically in the context of self-play in the game of Tic-Tac-Toe and Connect Four. Recognized for its simplicity, Tic-Tac-Toe offers a controlled environment to examine the subtleties of MARL and the distinct behaviors of various RL algorithms. We chose Connect Four for it's similarity to tic-tac-toe but also it's sutble differences in strategy to see if our method can generalize well. This study aims to assess the proficiency of these algorithms in learning and adapting through self-play, a setting where each agent competes against previous versions of itself, thus constantly facing evolving strategies. This approach is intended to contribute to a broader understanding of MARL strategies and their applicability in more complex decision-making scenarios.

A key aspect of this study is to analyze the learning efficiency and strategic depth of DQN and DDQN through iterative self-play. Agents initially trained against a random policy are subsequently engaged in self-play, creating a progressively challenging learning environment. This is essential to determine whether the agents can advance beyond basic reactive strategies to develop anticipatory and complex tactics. The findings are expected to shed light on the practical application of DQN and DDQN in more complex, dynamic environments.

2.2 Motivation

This research is motivated by the need to understand and evaluate the performance of Deep Q-Networks (DQN) and Double Deep Q-Networks (DDQN) in a self-play context within Multi-Agent Reinforcement Learning (MARL) environments. Tic-Tac-Toe and Connect Four, chosen for their clear and manageable structure, serves as an ideal platform for this investigation. The study extends beyond the simplicity of the game, aiming to extrapolate the findings to more intricate and realistic applications. The focus is on the adaptability of DQN and DDQN in self-play scenarios, a crucial element training Reinforcement Learning algorithms with limited compute resources and simulation quality.

3 Methods

3.1 Environment Description: Tic-Tac-Toe

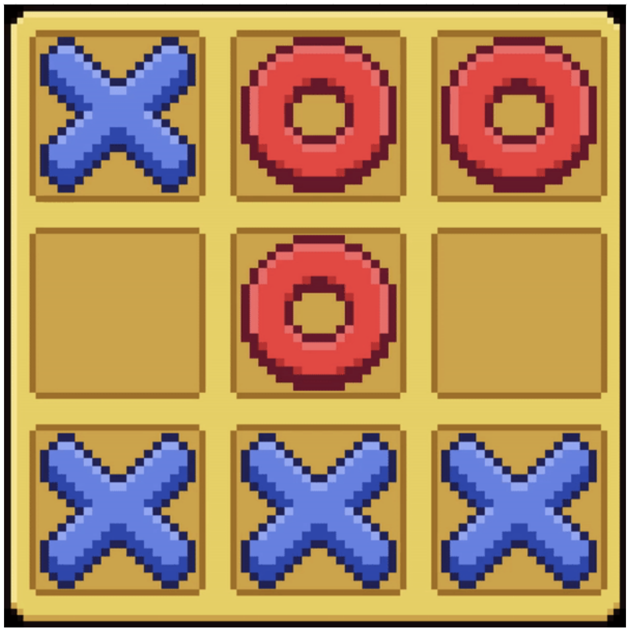

Figure 1: Tic-Tac-Toe PettingZoo Environment

This research utilizes the classic Tic-Tac-Toe game environment from the PettingZoo library, a highly regarded platform for conducting multi-agent reinforcement learning experiments. Tic-Tac-Toe, a straightforward yet strategically intricate game, serves as an ideal testbed for studying the dynamics of multi-agent interactions and the efficacy of algorithms like DQN and DDQN.

Game Dynamics Tic-Tac-Toe is a turn-based strategy game involving two players, labeled X and O. Players alternate marking spaces on a 3x3 grid, with the objective of being the first to align three of their marks horizontally, vertically, or diagonally. The game ends in a win for one player or a draw if neither achieves this goal. It represents a zero-sum game with perfect information, where one player's gain is inherently the other's loss.

Observation Space The PettingZoo implementation of Tic-Tac-Toe provides observations as a dictionary containing two elements: the main board state and a legal actions mask. The board state is represented as two 3x3 planes, where the first plane indicates the positions of Xs and the second plane indicates the positions of Os. Each cell within these planes can either be 0 (no mark) or 1 (mark present). The observation representation varies depending on the agent, ensuring that each agent perceives its marks on the first plane. The legal actions mask, a binary vector, indicates the permissible moves at any given turn, ensuring that agents only make valid moves within the game's rules.

Action Space The action space in this environment is discrete, with nine possible actions corresponding to the nine cells of the Tic-Tac-Toe board. Actions are indexed from 0 to 8, representing the cell positions as follows: top row (0, 3, 6), middle row (1, 4, 7), and bottom row (2, 5, 8). An agent's action is placing its mark (X or O) in one of these cells if it is vacant.

Rewards Structure The reward mechanism is straightforward: a win results in a reward of +1, a loss incurs a -1 penalty, and a draw yields 0 points for both agents. This structure reinforces the learning of winning strategies while penalizing moves that lead to losing outcomes or missed opportunities for victory.

3.2 Environment Description: Connect Four

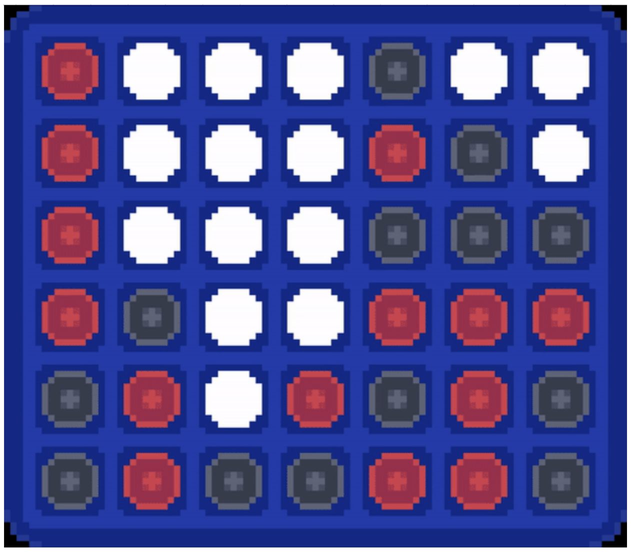
Figure 2: Connect Four PettingZoo Environment

This study utilizes the Connect Four game environment from the PettingZoo library, a prominent framework for multi-agent reinforcement learning experiments. Connect Four is a classic two-player turn-based game that combines strategic depth with straightforward gameplay, making it an excellent choice for studying reinforcement learning algorithms.

Game Dynamics Connect Four is played on a vertical grid of six rows and seven columns. Two players alternate turns dropping colored tokens into the columns. Each token falls to the lowest available space within the selected column. The objective is to be the first to form a horizontal, vertical, or diagonal line of four of one's own tokens. The game ends when either player achieves this goal or when all columns are filled, resulting in a draw. This game represents a challenge in strategic planning and foresight, making it a suitable testbed for reinforcement learning algorithms.

Observation Space The observation space in the Connect Four environment is structured as a dictionary containing two elements: the grid state and a legal actions mask. The grid is represented as two planes in a 6x7 grid, each corresponding to one player's tokens. A value of 1 in a cell indicates the presence of the player's token, while 0 denotes an empty cell or one occupied by the opponent. The observation space provides a comprehensive view of the current game state, crucial for decision-making in reinforcement learning.

Action Space The action space in Connect Four consists of seven discrete actions, numbered from 0 to 6, each representing a column in which a player can drop their token. The choice of column is a critical strategic decision, as it can enable winning formations or block the opponent's advances.

Rewards Structure The reward system in Connect Four is straightforward: successfully connecting four tokens grants the player 1 point, while the opponent receives -1 points. In the event of a draw, both players receive 0 points. This reward structure reinforces the goal of forming a line of four tokens while preventing the opponent from doing the same.

3.3 Libraries used for MARL

PettingZoo PettingZoo [6], a crucial component in our project, stands out as a comprehensive library for Multi-Agent Reinforcement Learning (MARL). Its extensive collection of environments, including classic games like Tic-Tac-Toe and Connect Four, provides a versatile and realistic testing ground for our experiments with DQN and DDQN algorithms. PettingZoo's intuitive API and seamless integration with reinforcement learning libraries like Tianshou enabled us to efficiently implement and evaluate our MARL models.

TianShou In our project, Tianshou [8], an elegant and highly modularized deep reinforcement learning (RL) platform, plays a pivotal role. Tianshou's design philosophy prioritizes flexibility and efficiency, making it an ideal choice for implementing and evaluating our MARL strategies with DQN and DDQN algorithms. Its compatibility with PyTorch allows for seamless integration and customization of neural network models, crucial for tailoring the learning architectures of our agents. Moreover, Tianshou provides a robust set of tools and utilities that streamline the process of training and testing in complex environments like Tic-Tac-Toe and Connect Four. It supports advanced RL techniques, including multi-agent learning environments through the PettingZoo library. Overall, Tianshou provides us with a clean and direct interface, enabling us to focus on the strategic aspects of agent interaction and policy refinement within the specified gaming environments.

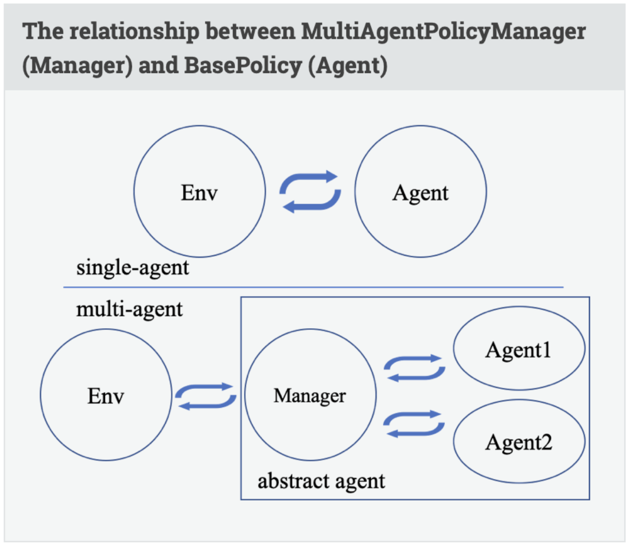
The relationship between MultiAgentPolicyManager (Manager) and BasePolicy (Agent)
Env
Agent
single-agent
multi-agent
Env
Manager
Agent1
Agent2
abstract agent

Figure 3: Enter Caption

MultiAgentPolicyManager The MultiAgentPolicyManager class is implemented in Tianshou to handle scenarios where multiple agents, each with their distinct policy, interact within the same environment. It acts as a central manager that oversees a list of policies corresponding to different agents. When a forward pass is executed, the MultiAgentPolicyManager efficiently dispatches the batch data to the appropriate policy based on the agent's identity, ensuring that each agent's actions and learning processes are aligned with its specific policy. Such an architecture is a simple way to transform regular DQN algorithms into their multi-agent counterparts.

3.4 Model Specification

In this study, two distinct architectures were employed for the agents: the Deep Q-Network (DQN) and the Double Deep Q-Network (DDQN). Both agents are built on the foundation of Q-learning, a value-based reinforcement learning method, but with distinct characteristics tailored to the Tic-Tac-Toe environment.

Deep Q-Network (DQN) The DQN agent utilizes a neural network to approximate the Q-value function. The network comprises multiple fully connected layers, each followed by a rectified linear unit (ReLU) activation function to introduce non-linearity. The input layer size corresponds to the flattened state space of the Tic-Tac-Toe board, while the output layer has a node for each possible action, representing the estimated Q-values. The architecture of the DQN was designed to balance computational efficiency with sufficient capacity to learn the complexities of the game.

Double Deep Q-Network (DDQN) The DDQN agent extends the DQN by decoupling the action selection from the Q-value generation. This modification addresses the overestimation bias observed in standard DQN. The DDQN architecture mirrors that of the DQN in terms of layer structure and activation functions but differs in how it updates its Q-values. Specifically, it employs two networks: one for selecting the best action and another for evaluating the chosen action.

3.5 Problem Statement

Adapting Q-Learning algorithms such as Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) for Multi-Agent Reinforcement Learning (MARL) environments presents unique challenges. One fundamental issue is the non-stationarity inherent in MARL settings, which contradicts the stationary environment assumption central to traditional Q-Learning. This non-stationarity violates the Markov property, as the state-transition probabilities change with the evolving policies of learning agents.

Another challenge faced by training Q-Learning algorithms in MARL through self-play is their susceptibility to catastrophic forgetting [3]. This can lead to instability in training, where agents might overfit on itself or demonstrate performance degradation over time due to the inability to retain effective strategies.

The research aims to address these challenges by exploring a training regime that begins with an agent learning against a stationary policy (random policy agent) and then progresses to self-play against a stationary version of itself. This approach is inspired by successful self-play methodologies in reinforcement learning [1]. By initially training against a naive random strategy and then transitioning to a harder opponent, we aim to develop an agent that not only adapts to better strategies but also retains robustness against previously encountered policies. The objective is to create a learning agent that demonstrates improved performance and policy stability, overcoming the limitations of traditional Q-Learning in MARL environments.

3.6 Training Process

The training of both agents was conducted in two phases: pretraining against a random policy and subsequent training against each other.

Pretraining Phase Initially, agents were pretrained against an opponent following a random policy. This phase aimed to acquaint the agents with the game's basic mechanics and to learn rudimentary strategies.

Self-Play Phase Following pretraining, the agents were set to play against previous versions of themselves. This phase was critical in advancing the agents' strategies from basic reactionary tactics to more sophisticated, anticipatory actions. The self-play setup inherently trains agents to play against opponents of similar skill level.

During training, we used higher learning rates for the pretraining phase, but encouraged more exploration in the self-play phase.

The training regimen employed in this study draws inspiration from methodologies prevalent in self-supervised learning, as detailed in He et al. (2019) [2]. This two-stage training approach is designed to enhance the performance of our agents while maintaining their ability to generalize across various scenarios. We demonstrate that by initially training against random policy and subsequently engaging in self-play, the agents not only achieve performance improvements but also retain a robust level of generalization.

3.7 Evaluation Metrics

To evaluate our agents' effectiveness, we conducted benchmark tests over 1,000 games against a predetermined opponent, with the exploration parameter set to zero. This benchmarking procedure serves as a straightforward and effective method to assess the performance capabilities of the agents in a controlled setting.

4 Results

The results of our experiments in both Tic-Tac-Toe and Connect Four environments provide insightful findings regarding the performance of Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) agents. The data, primarily drawn from a series of simulations, indicates an improvement in the agents' performance due to self-play training.

In the Tic-Tac-Toe environment, as shown in Table Table 1, both DQN and DDQN agents improved their win-rate against a random agent through self-play. Notably, the DQN self-play model exhibited a significant increase in performance, achieving a win-rate of 0.59, compared to 0.4 in with the initial pretrained model. Similarly, in Table 2, the DDQN self-play model demonstrated an improvement, with its win-rate rising from 0.38 to 0.5. This enhancement in performance underscores the efficacy of self-play in developing more robust and adaptable strategies.

Crucially, the results indicate that agents trained through self-play have successfully adapted to competing against earlier versions of themselves, as evidenced by their perfect win-rates in self-

Table 1: Tic-Tac-Toe: DQN vs DDQN self-play effect on win-rate

Model	Random	Self
DDQN self-play	**0.39**	1.0
DQN self-play	**0.59**	1.0
DDQN	0.35	×
DQN	0.4	×

Table 2: Connect Four: DQN vs DDQN self-play effect on win-rate

Model	Random	Self
DDQN self-play	**0.5**	1.0
DQN self-play	**0.44**	1.0
DDQN	0.38	×
DQN	0.42	×

Note: Win-Rate is calculated by *Number of Wins - Number of Losses*. Ties are not counted. Thus a positive win-rate should be seen as a indicator of a better policy.

play scenarios. This outcome suggests that while the pretraining against random agents provided a foundational understanding of the game, it led to the development of relatively weak strategies when facing more sophisticated opponents. The transition to self-play allowed the agents to evolve from these basic tactics to more advanced and effective strategies, highlighting the value of self-play in enhancing the agents' overall strategic capabilities.

In conclusion, the experiments validate the hypothesis that self-play training can significantly enhance the performance of both DQN and DDQN agents. This improvement is not just limited to increased win-rates against random agents but also extends to the agents' ability to compete effectively against their own evolving strategies, thus demonstrating enhanced robustness and adaptability.

5 Conclusion

The investigation of Deep Q-Network (DQN) and Double Deep Q-Network (DDQN) algorithms in the Multi-Agent Reinforcement Learning (MARL) context, specifically within Tic-Tac-Toe and Connect Four environments, has provided valuable insights into the efficacy of self-play training strategies. Our findings demonstrate that self-play enhances the strategic capabilities of these agents, enabling them to evolve from basic tactics to more advanced and effective strategies. This project underscores the potential of self-play in developing sophisticated decision-making processes in MARL setups.

Future research based on this work can delve into more robust ways of training an agent through self-play. There exist methods in self-play where an agent is trained against multiple formal versions of itself to reduce overfitting. Introducing complex environments or introduce variations in game rules can test the limits of these algorithms further. Expanding the scope to include cooperative strategies, where agents work together to achieve shared goals, could also provide intriguing results.

6 References

References

[1] Anthony DiGiovanni and Ethan C. Zell. Survey of Self-Play in Reinforcement Learning, July 2021. arXiv:2107.02850 [cs].

[2] Kaiming He, Ross Girshick, and Piotr Dollar. Rethinking ImageNet Pre-Training. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4917–4926, Seoul, Korea (South), October 2019. IEEE.

[3] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013. arXiv:1312.5602 [cs].

[5] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017. arXiv:1712.01815 [cs].

[6] J. K. Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis Santos, Rodrigo Perez, Caroline Horsch, Clemens Dieffendahl, Niall L. Williams, Yashas Lokesh, and Praveen Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning, October 2021. arXiv:2009.14471 [cs, stat].

[7] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning, December 2015. arXiv:1509.06461 [cs].

[8] Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. Tianshou: a Highly Modularized Deep Reinforcement Learning Library, August 2022. arXiv:2107.14171 [cs].