Multi-Agent Cooperative Environment with Fully Decentralized Approach

Ronald Truong (ronald.truong@mail.mcgill.ca)
Sienna Hsu (shih-ching.hsu@mail.mcgill.ca)

Abstract

In this project, we explore Multi-Agent Reinforcement Learning in a fully cooperative setting with partial observability on the benchmark game Hanabi. We first test the performance of a Single-Agent Deep Q-Network (DQN) method in this setting, then we compare the agent with algorithms using added components: Double-DQN (DDQN) and DDQN with Prioritized Experience Replay. Our results indicate that taking a Single-Agent approach is viable in MARL. Moreover, we confirm that DDQN does indeed perform better than the vanilla DQN algorithm. However, the performance gain of using Prioritized Replay Memory is more nuanced and requires further testing. In this project, Ronald and Sienna worked on researching, coding, training, writing the report, and making the video together.

1 Introduction

In Multi-Agent Reinforcement Learning (MARL), a cooperative environment is where agents share a common reward function and seek to optimize it together (1) (2). Hanabi is a 2 to 5-player cooperative card game where the players work together to score 25 points by playing cards from their hands. However, players cannot see their own cards and must rely on clues from teammates to deduce which cards they hold and play them in a correct order. We chose Hanabi as our project because it features the element of cooperation and partial observability, mirroring real-world scenarios where agents must navigate deduction and limited communication. Furthermore, DeepMind's endorsement of Hanabi as a benchmark for Cooperative MARL research underscores its significance in the field (3).

Within MARL, methods can be categorized into three main learning styles: Fully Centralized, where agents are collectively treated as a single agent and act simultaneously; Centralized Training Decentralized Execution (CTDE), where a centralized critic guides training while agents still act independently during execution; and Fully Decentralized, which applies Single-Agent RL methods directly without additional coordination mechanisms. We opted for the simpler Fully Decentralized approach: using the Single-Agent DQN method (4). We aim to test the performances of the vanilla DQN, DDQN, and DDQN with Prioritized Experience Replay (PDDQN) (5) (6). Due to resource constraints, we limit our experiments to two-player games with homogeneous agents, assuming all agents follow the same policy. This simulates real-world cooperative settings where it could be beneficial to establish a common behavior. Our experiment seeks to answer two key questions:

1. How well does the Single-Agent DQN perform in this multi-agent environment?
2. Are there significant performance improvements when using DDQN or PDDQN compared to vanilla DQN?

2 Related Work

Most other research papers on Hanabi explore Fully Centralized or CTDE methods, with the current state-of-the-art agent achieving an average score of 24.6 with policy-based methods coupled with a search algorithm. Additionally, considerable work has been done in ad-hoc settings, where agents interact with partners they haven't trained with. (7) (8) (9)

3 Background

Hanabi Environment. We used the PettingZoo Hanabi environment (10), which provides a similar API to Gym. The agent's observation is a 658-dimension one-hot vector that includes information such as what cards have been played and what information each player knows about the cards. The action space has 20 actions, which includes playing a card, giving a clue to another player, or discarding. The reward is 1 if the action successfully played a card, and 0 otherwise. The game terminates if 3 cards have been "misplayed", if the deck is empty, or if 25 points are reached. Since we don't prioritize early rewards, we use $\gamma = 1$ and the episode return is exactly the final score of the game. For more details about the rules of the game, see (11). In Hanabi, the true state involves the entire history of the game, not just the current board position, due to the significance of the action order. Partial observability in Hanabi arises from two factors: players' inability to see their own cards and the limited accessible history in the agent's observation.

Applying Single-Agent method. Since the observation vector encodes what information is known to the players and includes some historical game state data, partial observability is addressed in the state feature representation. This allows us to model the problem as a normal Markov Decision Process (MDP) and use Single-Agent RL methods where each agent treats the other as part of the environment. For instance, when Player 1 is in state S and selects action A, it alters the environment (board position); successful card plays increase the reward by 1. Player 2 then responds, further modifying the environment and potentially increasing the reward by 1. When it gets back to Player 1's turn, their observation is their next state S', and the reward $R \in \{0, 1, 2\}$ is the total change in reward since their last turn. Player 2 experiences symmetrically.

4 Methodology

Since we assume homogeneous agents, both agents update and select actions using the same set of networks. Our first goal is to use a baseline model to explore whether this single-agent approach can perform significantly better than a random agent. Our second goal is to conduct an ablation study to improve upon our baseline model.

4.1 Model architecture

Baseline model: DQN. Given the large state space, we decided to use function approximation, specifically the DQN model with Pytorch (12). We used ϵ-greedy policy, where we begin with ϵ_{start} and exponentially decay until ϵ_{end} with decay rate T. We used a policy network (θ) and a target network (θ') to remove moving targets while updating. Both networks begin with identical weights. The policy network selects actions, while the target network computes targets. The policy network's weights are updated using smooth L1 loss and the Adam optimizer. Lastly, the target network's weights are soft-updated at each step: $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$. Both networks consist of two fully connected hidden layers of 512 neurons with ReLu activation, following the paper of Bard et al. (3). The update rule is given by

$$Q(s, a; \theta) \leftarrow Q(s, a; \theta) + \alpha \left[r + \gamma \max_{a'} Q(s', a'; \theta') - Q(s, a; \theta) \right].$$

Batch learning, experience replay, sampling. To increase data efficiency, we use a memory buffer with capacity M to store the transition tuples (s, a, r, s') generated during the episodes. Given a batch size B, at the learning step, we sample B transitions to replay these experiences for updating. Vanilla experience replay samples the transitions uniformly. Proportional Prioritized Experience Replay (PER) assigns a priority, $|\text{TD-error}| + \epsilon_{per}$, to each transition such that those with a larger magnitude of TD-error are more likely to be sampled. The hyperparameters α_{per} controls the level of prioritization while β_{per} controls the strength of importance sampling bias correction (6). We tested both online and offline learning. Online learning samples from the memory B transitions *during* the episode every time after a player acts for updating. Offline learning collects transitions throughout the episode but does not sample and update until an episode is over.

4.2 Performance of baseline DQN

We selected hyperparameter candidates by taking those used in the work of Bard et al (3) and expanding the ranges. Ideally, we would test hyperparameter combinations more thoroughly, but due to hardware limitations, we tuned this model in four stages; each tuning experiment consisted of 2 trials of 10K episodes each unless specified otherwise. Here is a summary of our procedure and tuning results:

Stage 1: Offline vs Online. We tested hyperparamters memory size $M = [1000, 5000]$, batch size $B_{on} = [2, 4, 8]$ for online model and $B_{off} = [32, 64, 128, 256]$ for offline model, learning rate $\alpha = 0.0001$, discount rate $\gamma = 1$, $\epsilon_{start} = 1$, $\epsilon_{end} = 0$, decay rate $T = 10000$, and soft-update rate $\tau = 0.005$ for the online and offline DQN. We found that offline learning performed marginally better than online learning, and thus for the rest of the experiment, we assumed offline learning.

Stage 2: $\boldsymbol{\alpha, M, B}$. We experimented with $\alpha = [0.001, 0.0001, 0.00001]$, $M = [1000, 5000, 10000]$, $B = [32, 64, 128, 256]$. We eliminated $\alpha = 0.001$ as it performed ostensibly worse. Some hyperparameter combinations had mediocre performance, such as $(alpha = 0.00001, B = 32)$. Others, however, showed similar performances, and we chose arbitrarily to proceed with $(\alpha = 0.0001, M = 10000, B = 256)$.

Stage 3: $\boldsymbol{\epsilon_{end}, T}$. We tested $\epsilon_{end} = [0, 0.01, 0.05]$ and $T = [100, 1000, 10000]$. The results were once again comparable, likely due to the only tuning for 10K episodes. We proceeded arbitrarily with $(\epsilon_{end} = 0, T = 10000)$.

Stage 4: $\boldsymbol{\tau}$. Finally, we tested $\tau = [0.001, 0.005]$. For this experiment, we ran for 60K episodes and found that $\tau = 0.005$ performed marginally better.

4.3 Ablation study

Our first ablation study is to use DDQN. The only difference of DDQN from DQN is the update rule, based on Wang et al. (5):

$$Q(s, a; \theta) \leftarrow Q(s, a; \theta) + \alpha \left[r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta') - Q(s, a; \theta) \right].$$

DDQN underwent stages 2, 3, and 4 tuning. Its results did not significantly differ from those of the DQN until stage 4 tuning. Hence we used the hyperparameter combination decided for DQN except for $\tau = 0.001$ for DDQN. Next, we added PER on top of DDQN. We used $\alpha_{per} = 0.6$ and $\beta_{per} = 0.4$ as recommended by Schaul et al. (6). We conducted stages 2, 3, and 4 tuning similarly. In the end, we used the same hyperparameters as DDQN.

5 Experiments and Results

For the final experiments, we ran each of the DQN, DDQN, and PDDQN models for 8 trials, 100K episodes. The hyperparameters used are as described in the previous section. The results and comparison are shown in Figure 1. The average score of the last 100 episodes is 9.17 for DQN, 11.74 for DDQN, and 9.98 for PDDQN. All three models exhibit upward trends and continue to grow by the end of the 100K episodes, showing that a single-agent approach can tackle a multi-agent problem. The scores they achieve are significantly higher than the score achieved by a random agent at 1.25, but much lower than the score achieved by us (human experts) at 17.5 over 10 games. We believe with longer training, the DQN agent and its variants have the potential to reach higher scores.

We note some similarities and differences between the three agents. Firstly, they all plateau starting around the 5000th episode and continue for varying duration. We hypothesize that, initially, the agents may be exploring and discovering new strategies, leading to an increase in rewards. Once they have learned a set of effective strategies, the agents might over-exploit these strategies, causing the score to plateau. However, as the environment changes or the agents discover new, more optimal strategies, the score increases again. This could also be the case for DDQN and PDDQN in the later stages when the scores seem to plateau again. Yet, without further testing, we cannot confirm this hypothesis.

Secondly, it is clear that within 100K episodes, DDQN and PDDQN outperform and have smaller variances compared to DQN. This is expected because DDQN corrects for the overestimation bias occurring in DQN and can achieve faster convergence. PER, however, does not improve the

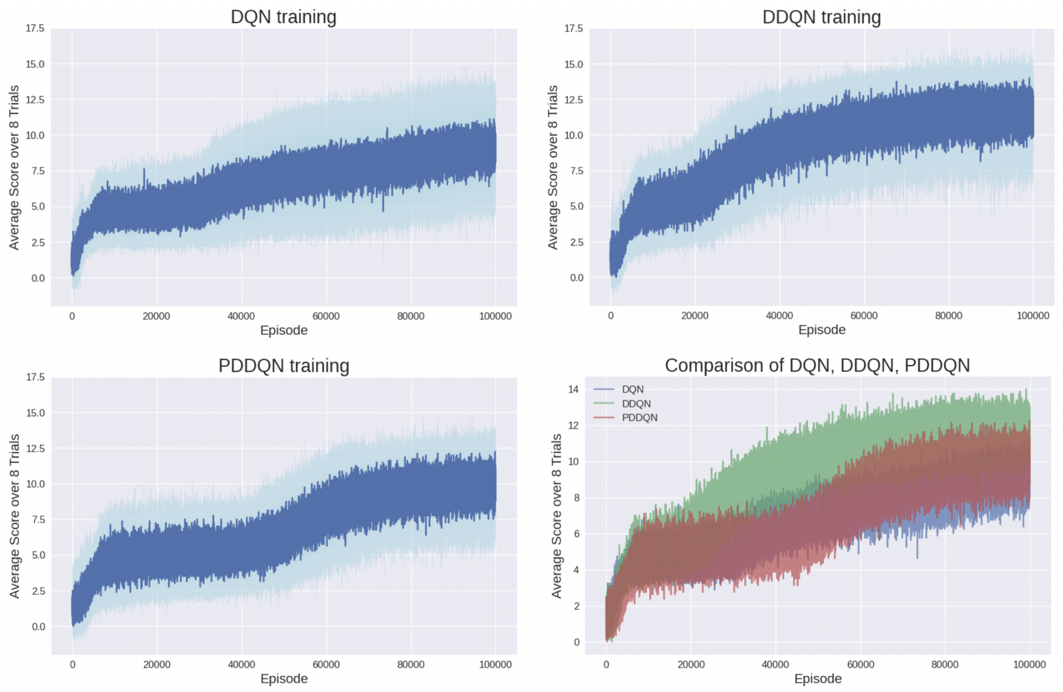

Figure 1: The results and comparison of the DQN, DDQN, and PDDQN models. The shading shows the standard deviation.

performance of DDQN. We suspect this is due to PER prioritizing noisy or misleading experiences. Since agents work in a non-stationary environment, a transition with a high TD error might not be relevant in later episodes but still maintain a high priority. Additionally, during exploration (especially when we set $\epsilon_{start} = 1$), agents might encounter states or actions that are rarely seen, have high TD errors, but are not conducive to scoring. These transitions could be disproportionately prioritized by PER. Still, without further tuning of α_{per} and β_{per} and longer trials, it is hard to conclude on PER's performance.

6 Conclusion and Future Work

The Deepmind's Rainbow agent achieves the score 20.64 after training for seven days using NVIDIA V100 GPU. With our NVIDIA RTX 2080 GPU and around 16-hour training time, our results achieve around the score 11 and show signs of continual improvements. It demonstrates that Single-Agent simplification of Fully Decentralized MARL does provide a solution to solving a multi-agent, cooperative, and partially observable environment. Further, we could clearly see that DDQN improved upon DQN, although PER was tricky to deploy. For future work, we would like to train for longer and for more trials, employ n-step bootstrapping instead of one-step TD, choose features for the state space using domain knowledge, and test the scalability of the method with mroe players. We would also like to integrate history of the gameplay more using techniques such as Recurrent Neural Network or concatenating multiple transitions into a single experience sample. Lastly, partial observability was handled by our models purely by the neural networks detecting patterns in the transitions. We could further explicitly construct agents' beliefs about a partially observable state using belief states (2).

References

[1] Y. Du, J. Z. Leibo, U. Islam, R. Willis, and P. Sunehag, "A review of cooperation in multi-agent learning," 2023.

[2] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2021.

[3] N. Bard, J. N. Foerster, S. Chandar, N. Burch, M. Lanctot, H. F. Song, E. Parisotto, V. Dumoulin, S. Moitra, E. Hughes, I. Dunning, S. Mourad, H. Larochelle, M. G. Bellemare, and M. Bowling, "The hanabi challenge: A new frontier for ai research," *Artificial Intelligence*, vol. 280, p. 103216, Mar. 2020. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2019.103216

[4] S. E. Li, *Introduction to Reinforcement Learning*. Singapore: Springer Nature Singapore, 2023, pp. 1–14. [Online]. Available: https://doi.org/10.1007/978-981-19-7784-8_1

[5] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2016.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2016.

[7] A. Lerer, H. Hu, J. Foerster, and N. Brown, "Improving policies via search in cooperative partially observable games," 2019.

[8] A. Fickinger, H. Hu, B. Amos, S. Russell, and N. Brown, "Scalable online planning via reinforcement learning fine-tuning," 2021.

[9] B. Grooten, J. Wemmenhove, M. Poot, and J. Portegies, "Is vanilla policy gradient overlooked? analyzing deep reinforcement learning for hanabi," 2022.

[10] "Pettingzoo documentation." [Online]. Available: https://pettingzoo.farama.org/environments/classic/hanabi/

[11] "Hanabi rulebook." [Online]. Available: https://cdn.1j1ju.com/medias/b3/a9/0e-hanabi-rulebook.pdf

[12] A. Paszke and M. Towers, "Reinforcement learning (dqn) tutorial — pytorch tutorials 1.8.0 documentation," pytorch.org. [Online]. Available: https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html