Deep Q-Network: Multi-Agent Survival Strategy

Andy He
andyhe
andyhe@umich.edu

1. Introduction

We explore the dynamics of self-play in Deep Q-Learning Networks (DQNs), focusing on competitive strategy development through dual-agent interaction. Unlike traditional supervised learning, this approach fosters adaptability by challenging agents to evolve against an unpredictable opponent. Our contributions include the implementation of dual DQN agents in self-play, analysis of emergent strategies, and evaluation of competitive learning dynamics. This work provides insights into multi-agent reinforcement learning for applications such as game testing and strategic simulations.

2. Related Work

Self-play in reinforcement learning has significantly advanced AI in complex games. Notable examples include AlphaGo, achieving superhuman Go performance, and OpenAI's Five, mastering Dota 2 through extensive self-play [1]. While these systems demand sophisticated architectures and high computational resources, simpler implementations like AlphaXos leverage DQNs for strategy learning in board games [2]. Studies such as Huang and Zhu's work on Tic-Tac-Toe demonstrate DQNs' effectiveness in basic competitive scenarios [3]. Tampuu et al. (2015) extended DQNs to multi-agent settings, showing competitive and collaborative behaviors based on reward structures [4].

These findings highlight DQNs' adaptability in multi-agent contexts, forming the basis for our project, which explores DQN-based self-play to understand learning dynamics in competitive environments.

3. Methods

The study "Multiagent Cooperation and Competition with Deep Reinforcement Learning" by Tampuu et al. examined how two agents, controlled by independent Deep Q-Networks (DQNs), interact in Pong under different reward schemes [4]

- **Fully Competitive:** A zero-sum game where one agent's success directly penalizes the other (e.g., left player scores +1, right player scores -1).
- **Fully Cooperative:** Both agents lose a point whenever the ball exits play, encouraging them to keep the ball in play.

While these strategies are conceptually interesting, we found them unsuitable. In the Fully Competitive scheme, rewards are delayed, leading to agents being rewarded for actions unrelated to the final outcome. In the Fully Cooperative scheme, agents are penalized for events beyond their control (e.g., one bot is punished for the other's mistake). To address these limitations, we implemented a "survival strategy" reward scheme focused on individual accountability. Each bot is responsible for defending its own goal: if a bot allows a score, it is penalized (-1), while the opponent receives no penalty (0). Conversely, successful deflections are rewarded (+1). This ensures rewards are directly tied to actions, promoting fairer and more effective learning.

3.1. General Structure

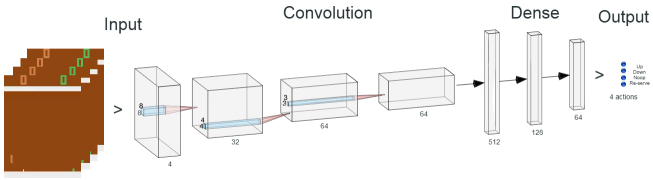

Figure 1. Neural Network Structure For DQN

Algorithm 1. Deep Q-learning with experience replay.

Require:
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize the target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

Ensure:
for episode $= 1$ to M **do**
  initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$;
  for $t = 1$ to T **do**
    With probability ε select a random action a_t;
    Otherwise select $a_t = \arg\max_a Q(\phi(s_t), a; \theta)$;
    Execute action a_t in emulator and observe reward r_t and image x_{t+1};
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$;
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D;
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D;
    Set $y_j = r_j$ if episode terminates at step $j+1$, otherwise $y_j = r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$;
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ with respect to the network parameter θ;
    Every C steps reset $\hat{Q} = Q$, i.e., $\theta^- = \theta$;
  end for
end for

Figure 2. DQN Algorithm Pseudocode [5]

	Left player scores	Right player scores	Left player hits ball	Right player hits ball
Left player reward	0	-1	1	0
Right player reward	-1	0	0	1

Table 1. Survival Strategy

4. Experiments

We implemented a custom DQN model with a unique reward system and compared its performance to the baseline in Tampuu et al. [4]. The study evaluated agent behavior using metrics like paddle hits per point, wall bounces per paddle bounce, and serving time per point. We adapted these metrics, focusing on paddle bounces per point and introducing cumulative points per episode to better capture learning progression under our "survival strategy" reward system.

- **Paddle bounces per point vs. epoch:** Measures how many paddle hits occur before a point is scored.
- **Wall bounces per paddle bounce vs. epoch:** Tracks how frequently the ball hits walls relative to paddle bounces.
- **Serving time per point vs. epoch:** Monitors the time taken to serve the ball after a point is scored.

Unlike the study's use of 250,000-step epochs, we measured performance in episodes (games up to 21 points) for intuitive progress tracking. Wall bounces were excluded as they were less relevant in our setup. Our model trained over 1180 episodes using epsilon decay (0.92, minimum 0.10) to balance exploration and exploitation. The Adam optimizer with a learning rate of 0.00025 and a gamma value of 0.99 was used for stability. While constrained by time and budget, our results demonstrated measurable learning trends, though full replication of the baseline study's results was limited.

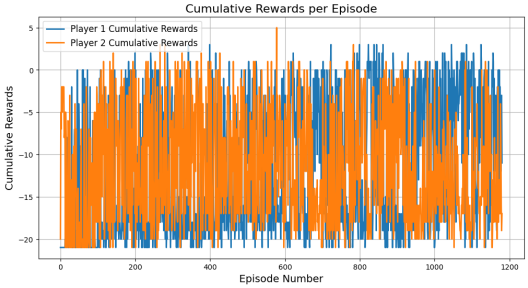

Figure 3. Cumulative Rewards Per Episode

5. Conclusion

This project demonstrated the potential of self-play in enhancing DQN agents' strategic capabilities. Due to time constraints, we can only observe the agents' fluctuating performance as it continues to learn. Consequently, we were unable to fully replicate the study's results. However, based on their findings—where minimal improvement in "paddle bounces per point" occurred until around the 48th epoch in their graph [4], we hypothesize that with greater computational resources, our agents could achieve similar outcomes. Scaling up training time would likely enable our agents to develop more advanced strategies and converge to behaviors observed in the baseline study. However, achieving benchmarks set by studies with longer training times, such as DeepMind's 38-day experiments [6], underscores the importance of extended training.

Our "survival strategy" reward scheme addressed fairness and individual responsibility, offering a viable alternative to fully competitive and cooperative schemes. Future work could explore advanced strategies, training optimizations, and additional techniques like frame skipping or dueling networks.

This study highlights DQN agents' utility in competitive multi-agent environments and the need for further research to maximize their potential.

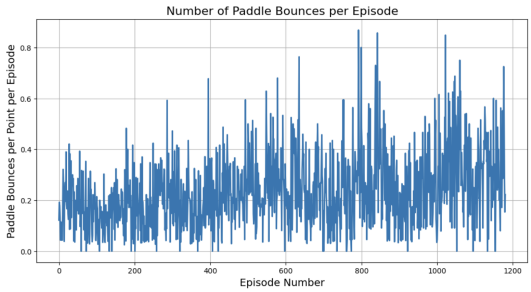

Figure 4. Number of Paddle Bounces Per Episode

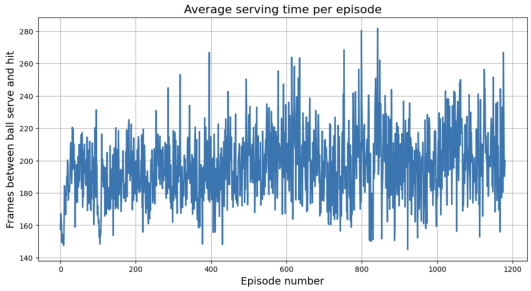

Figure 5. Average Serving Time Per Episode

References

[1] Christopher Berner, Greg Brockman, Brooke Chan, et al. Dota 2 with large scale deep reinforcement learning, 2019.

[2] Robin Ranjit Singh Chauhan. Alphaxos: A multi-agent reinforcement learning framework for board games, 2022.

[3] Zhi-Wen Huang and Wei-Jin Zhu. Tic-tac-toe with deep multi-agent reinforcement learning. *University at Buffalo RL Workshop Proceedings*, 2019.

[4] Ardi Tampuu, Tambet Matiisen, et al. Multiagent cooperation and competition with deep reinforcement learning. *arXiv preprint arXiv:1511.08779*, 2015.

[5] Fuxiao Tan, Pengfei Yan, and Xinping Guan. Deep reinforcement learning: From q-learning to deep q-learning. In Derong Liu, Shengli Xie, Yuanqing Li, Dongbin Zhao, and El-Sayed M. El-Alfy, editors, *Neural Information Processing*, pages 475–483, Cham, 2017. Springer International Publishing.

[6] Keras Team. Deep q-learning (dqn) example for playing breakout. `https://keras.io/examples/rl/deep_q_network_breakout/`. Accessed: 2024-12-11.