Texas Hold'em With DQN Under Different Settings

Yu-Tang Shen, Xiang Liu
Department of Computer Science, San José State University, San Jose, California 95192, USA
Email: yutang.shen@sjsu.edu, xiang.liu@sjsu.edu

***Abstract*—AI has been fairly successful for perfect-information games. However, it struggles to get similar performance in imperfect-information games. Hence researches on leveraging and optimizing different reinforcement learning methods on this scenario becomes crucial. This paper will discuss the experiment results from applying Deep Q-Network to No-limit Texas Hold'em game.s It will also reveal some possible areas for future optimization.**

***Index Terms*—Reinforcement Learning, Deep Q-Network, imperfect games.**

I. Introduction

AI has been fairly successful for traditional board games like chess, Go and shogi. Bots like AlphaGo had already achieved professional level if not superhuman level in these games. These games are known to be perfect-information games meaning with enough computational resources one can predict all future moves at any stage [1]. There is no hidden information for both players. AlphaGo used reinforcement learning with search (Monte Carlo Tree Search) to master the game of Go and managed to defeat the best human player at that time [2]. However, reinforcement learning with search proved to be highly unstable and achieved poor performance in imperfect-information games like poker, where players can't see each other's cards [1]. Thus, it is natural to think about utilizing other reinforcement learning techniques and algorithms in this case. So how do state-of-the-art reinforcement learning algorithms perform in imperfect-information games?

II. Formal Problem

In our project, we transformed No-limit Texas Hold'em into a reinforcement learning problem. In particular, we applied Deep Q-Network (DQN) to No-limit Texas Hold'em and evaluated its performance. We know a reinforcement learning problem typically consists of states, actions and rewards. The DQN agent is trying to find expected Q values for different states by simulating a bunch of hands. In the game setting, the environment will deal two random cards to each player, let them make decisions and observe payoffs at the end. On each episode, the DQN agent adopts the decaying epsilon greedy method to improve its policy and experience in the real environment. For the choice of algorithms, we choose DQN because it is one of the most standard reinforcement learning algorithms nowadays. DQN performs relatively well in some complex games like the Atari games and requires less computation resources [3]. So we want to evaluate its performance and use it as the benchmark for other advanced algorithms. We used RLCard, an open source toolkit which provided us with various card environments and learning algorithms.

III. Methodology

Since RLCard provided us with a built-in game environment and state representations, we decided to start experimenting with some built-in features, where our environment can be pictured as follows:

- 2 players: our agent and an opponent
- Each player had 100 chips in the beginning of each round
- Players can fold, check / call, raise half pot, raise full pot, all-in

We first tried using its default Random Agent as our DQN Agent's opponent, but our agent was not able to learn well in this configuration; we believed that the poor performance was due to our agent not experiencing the full game since the Random Agent randomly decided to fold. Thus, we implemented an agent, Fish Agent, to always call our agent's action; this actually aligned with a lot of amateur players in the real-world where players hope the next public card would match their hands, and it also allowed our agent to successfully learn what hole card combinations were considered strong. With the Fish Agent as our DQN Agent's opponent and trying different hyper-parameters for our DQN Agent, including ϵ decaying time, batch size, Q estimator neural network shape, etc., we were able to achieve some promising results as shown in figure 1. On average the agent can win 10 to 15 chips in each game.

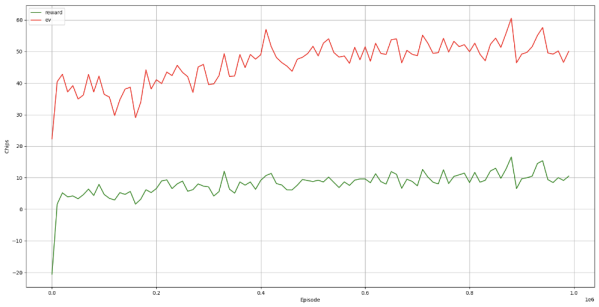

Fig. 1. Fish Agent

With the acceptable results, we wanted our DQN Agent to learn from a stronger opponent when it converged, so we cloned our trained DQN Agent as our opponent when we considered our DQN Agent converged. However, the result was worse in this design as shown in figure 2. In the beginning,

we thought it might be caused by the ϵ being too small when we switched its opponent for our DQN Agent to explore enough against the stronger opponent– we increased the ϵ back to the initial value whenever we switched its opponent. Still, the DQN Agent performed worse than the one under the original setting.

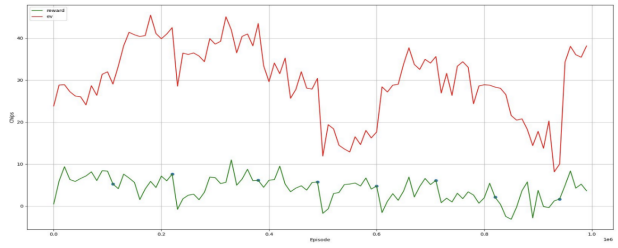

Fig. 2. Alternating Agent (blue dots indicate the opponent had switched)

The default state representation from RLCard was appending an array of 52 card one-hot vector with the chip information, where the 52 card one-hot vector started with Aces and ended with K of each suit [4]. We thought this configuration did not align with our knowledge of Texas Hold'em. First of all, the 52 card one-hot vector mixed the public card and hole cards information, and we believed that hole card information played an important role in this game. Secondly, the one-hot vector started with Aces instead of 2 did not align with the game rule: 2 was the smallest card of all.

Thus, we designed a different state representation to solve these two problems. First of all, we encoded all cards into a number, where four Aces had the largest values [51,48], 2s had the smallest values [3,0], and unrevealed public cards were encoded as -1. We then came up with a new state representation as follows: *(hero hole card 1, hero hole card 2, public card 1, public card 2, public card 3, public card 4, public card 5, hero's chips in the pot, chips in the pot)*.

We implemented this idea to the game environment and tried various hyper-parameter configurations on the DQN Agent, but the performance was not as good as we expected. We believed that there must be some DQN Agent configurations that can make this state representation outperform the original setup, but due to limited computational resources, we could not find this configuration to improve our DQN Agent's performance.

IV. Evaluation Settings And Criteria

In the Texas Hold'em game, there was an existing evaluation statistic to judge how well a player played his hands – All-in EV. The concept of All-in EV was quite straightforward: $probability\ of\ winning \times chips - probability\ of\ losing \times investedchips$. Although the opponent's hands were hidden from our agent, we could get the perfect information of each game to calculate the probability of winning (a.k.a equity). We utilized the Holdem Calculator Library to calculate the equity and further derive the EV value to compare with our actual reward [5]. In RLCard, the rewards for betting games like No-limit Texas Holdem are defined as the average winning big blinds per hand [3].

V. Evaluation Results

This section shows the experiment results for our DQN agent under different settings.

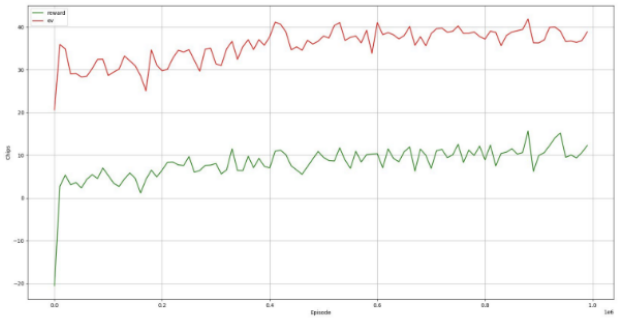

Fig. 3. Default State Representation (state shape: 54)

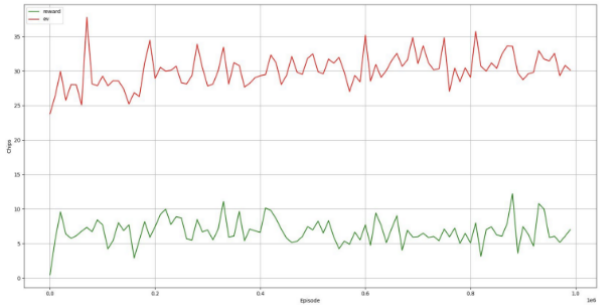

Fig. 4. New State Representation (state shape: 9)

Figure 3 shows the evaluation result from the default state representation, while figure 4 shows the result from our new state representation. Both simulations were conducted with Q estimators of neural networks with one layer of 128 neurons. As mentioned before, we did not get the improvements we expected from the new state representation. The average reward was less and the gap between the EV and the reward was bigger. Furthermore, there were little improvements after the initial surge in the new state representation.

Figure 5 shows the evaluation result from a single layer with 32 neurons Q estimator, while figure 6 is from a single layer with 128 neurons Q estimator. Both simulations were conducted with the Fish agent as the opponent. We improved the performance and were able to get a better Q-value estimator under different configurations. Further results show that the more complex our neural networks become, the better performance we get. However, there is a trade-off between the complexity of the neural network and the execution time. We noticed that it took considerably more time to run the simulation after we changed the Q estimator shape to 128 x 128.

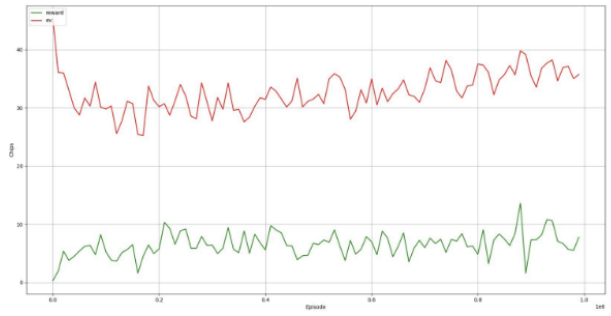

Fig. 5. Single Layer with 32 Neurons

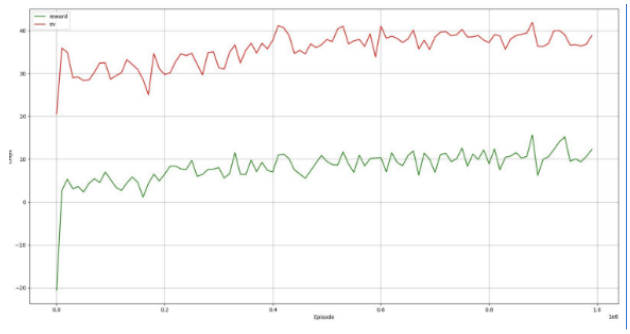

Fig. 6. Single Layer with 128 Neurons

Here in figure 7, we are showing the performance of our DQN agent against the default random agent. The learning curve is fluctuating and the average reward never reaches 10 chips. It also confirms that without a good opponent, it is challenging for the DQN agent to train and learn.

VI. Discussion

We attempted to attain satisfying results using the DQN method to approximate the state action value to further construct a profitable policy in Texas Hold'em games. Despite under most of the settings our agents could either win some chips or at least not lose many chips, we realized that our state representations might still be too naive to actually reflect the game state. For example, one of the most important information in this game was the hero's position – playing in the earlier positions had less advantage. Although this was not crucial in our two-player game setting, this information might be significant in multi-player games. Another information that could play an important role in this game was the action history of the opponent, or in other words, what card combination could the opponent hold that aligned with his action history. In Facebook's RL Texas Hold'em Agent, they described this information as the belief state, and their agent's performance was significantly better than ours [1]. By including these information to our state representation with proper DQN agent parameter settings, we believe that the agent can at least approximate the EV value and play the best action possible with each hand.

Additionally, as we showed in the evaluation result section,

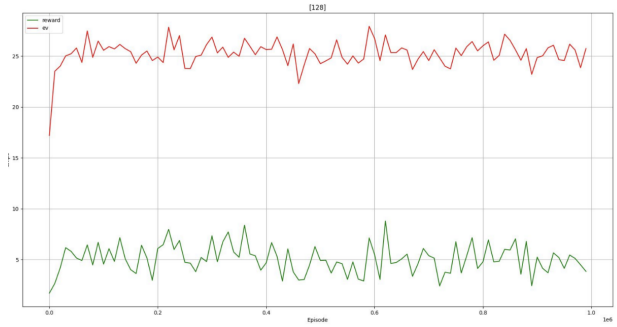

Fig. 7. Random Agent

a reasonable opponent also plays an important role for our agent to obtain a best policy. Therefore, constructing a more sophisticated agent, for example, raise under some advantageous circumstances instead of always calling like the Fish Agent we implemented, could also be beneficial to improving the performance for this problem.

VII. Related Work

There are a few new reinforcement learning techniques that can be adopted to poker games. The most famous one is Counterfactual Regret Minimization (CFR), a regret minimization method designed for imperfect games [3]. Recursive Belief-based Learning (ReBeL), proposed by Facebook, was developed from this CFR algorithm [1]. And it defeated a top human professional just like AlphaGo. However, CFR is still pretty computation-intensive, meaning we won't be able to train the agent on our local computers unlike DQN. Another one is Neural Fictitious Self-Play (NFSP), presented by the father of AlphaGo - David Silver [6]. It is a deep reinforcement learning approach for partially observable games. It showed good results for various betting games. But it needs more in-depth optimizations to make it work and requires a lot of domain knowledge.

References

[1] "Rebel: A general game-playing ai bot that excels at poker and more." https://ai.facebook.com/blog/rebel-a-general-game-playing-ai-bot-that-excels-at-poker-and-more/. Accessed: 2021-12-10.

[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[3] D. Zha, K.-H. Lai, Y. Cao, S. Huang, R. Wei, J. Guo, and X. Hu, "Rlcard: A toolkit for reinforcement learning in card games," *arXiv preprint arXiv:1910.04376*, 2019.

[4] "Games in rlcard," Accessed: 2021-12-10.

[5] "Holdem calculator." https://github.com/ktseng/holdem_calc. Accessed: 2021-12-10.

[6] J. Heinrich, M. Lanctot, and D. Silver, "Fictitious self-play in extensive-form games," in *International conference on machine learning*, pp. 805–813, PMLR, 2015.