[[1]](#footnote-1)

Building Poker Bot with Reinforcement Learning (December 2020)

László Barak, Mónika Farsang, Ádám Szukics

**No-limit Texas Hold’em is one of the most popular cardgames in the world. Leduc Hold’em is a simplified version, where new implementations can be easily tested. We present a poker bot that learns to play Limit-Hold’em and Leduc Hold’em by training a random agent. For this purpose we use the RLCard environment. Our bot uses Deep Q Learning and it’s implemented in PyTorch. We also introduce various strategies that even though are quite simple, yet sometimes effective. In the process we use a replay buffer to store past experiences and another Q-network to aid the learning process. We also take a look at how long-term-planning poker machines perform against our implementation by creating a sample one that uses counterfactual regret minimization. Lastly, we compare the results of each unique strategy to the others, thus giving the reader a better understanding of how poker should be played (in a really simple manner).**

*Impact Statement* — The impact statement should not exceeed 150 words. This section offers an example that is expanded to have only and just 150 words to demonstrate the point. Here is an example on how to write an appropriate impact statement: Chatbots are a popular technology in online interaction. They reduce the load on human support teams and offer continuous 24-7 support to customers. However, recent usability research has demonstrated that 30% of customers are unhappy with current chatbots due to their poor conversational capabilities and inability to emotionally engage customers. The natural language algorithms we introduce in this paper overcame these limitations. With a significant increase in user satisfaction to 92% after adopting our algorithms, the technology is ready to support users in a wide variety of applications including government front shops, automatic tellers, and the gaming industry. It could offer an alternative way of interaction for some physically disable users.

*Index Terms*—poker, AI bot, reinforcement learning, Deep Q‑Network

# INTRODUCTION

T

he current results in reinforcement learning have always been important for humanity. Not so long ago we managed to reach what some considered to be impossible, namely beating the current best Go player by using artificial intelligence [1]. This was a huge milestone, even though perfect-information games were considered to be easier to solve than their imperfect-information counterparts. The reason for that is obvious: imperfect information games require more complex reasoning than similarly sized perfect-information games. We have to balance our strategy so that our opponent does not find out too much about our way of playing, hereby the hidden information we have. To give an example, in poker if we always raise when we have a pair of kings, our opponents will recognize that and punish us for doing it, either by folding their cards or by raising if the current public cards favour them. As we can see from the previous example, poker is intuitively easy to understand and captures the challenges of hidden information effectively and elegantly. For this reason, it has been widely used to illustrate game theory concepts. [2]

The classical solution for games is a Nash equilibrium. This strategy ensures that no player can increase his or her expected utility by altering their strategy. All finite extensive-form games have at least one Nash equilibrium.

# Literature review

In the past few years, the media has been constantly covering breakthroughs in reinforcement learning. For example, in 2019 OpenAI Five [3] became the first AI system to defeat the world champions at an esports game. Some thought it was inevitable that the precision of the computers will outshine their human opponents, and it was bound to happen. Later that year Pluribus shocked the world: a superhuman AI has been made that managed to consistently beat human professional players. Beating humans in poker is not something we have not seen before.

First, in 2015 a paper claimed that Heads-up Limit Hold’em Poker is essentially solved. Cepheus (the name of the supercomputer) was trained for 68 days with CPUs, using a special version of counterfactual regret minimization. Cepheus heavily outperformed Polaris, the poker bot made in 2008. This triggered a snowball effect in the world of poker AIs.

In 2016 another paper [4] argued that their NFSP (Neural Fictitious Self-Play) is the first deep reinforcement learning method known to converge to approximate Nash equilibria in self-play.

In early 2017 DeepStack [5] achieved expert-level artificial intelligence in heads-up no-limit poker. It became the first poker ai that managed to beat professional poker players in heads-up No-Limit Texas Hold’em. DeepStack allows computation to be focused on specific situations that arise when making decisions and the use of automatically trained value functions while using minimal domain knowledge.

To completely isolate these machines from humans, in 2018 Libratus [6] was created without any domain knowledge and also greatly reduced the required training time. It defeated four top human specialist professionals in a 120,000-hand competition in heads-up No-Limit Texas Hold’em. It consisted of a blueprint strategy for the overall strategy, an algorithm that provided more information on the details of the strategy for subgames that were reached during play, and a self-improver algorithm that fixes potential weaknesses that the opponents might identify.

To top it all, in 2019 Pluribus [7] crushed multiple humans simultaneously. It was tested in six-player No-Limit Texas Hold’em poker, the most commonly played poker format in the world.

# Methodology

We used RLCard card environments [8] that are designed for reinforcement learning research. It is an easy-to-use toolkit that provides Limit Hold’em and Leduc Hold’em environment. The latter is a simplified version of Limit Texas Hold’em and it was constructed to have a more tractable game [9].

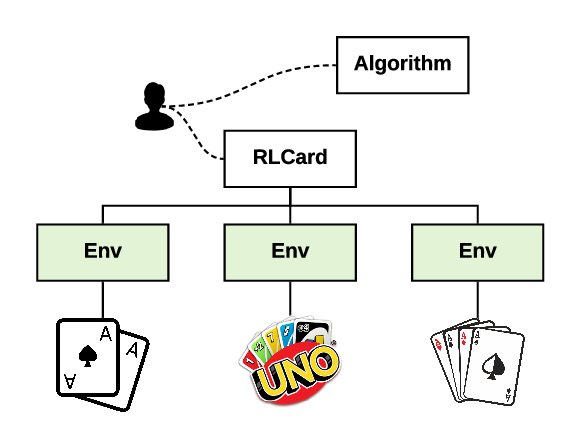


Fig. 1. An overview of the RLCard toolkit [8]

Both types have the same actions: *check, call*, *raise* and *fold*. During *checking* the action passes to the next player without betting. In the case of someone bets, this action is not possible anymore. *Calling* means matching a bet or a raise. If the player chooses to *raise*, he/she increases the size of an existing bet in the round. *Folding* is discarding one’s hand.

The payoff is identical as well in both environments. It is based on the big blinds per hand. The player gets the positive or negative R reward if he/she wins or loses R times the amount of the big blind, respectively.

Limit Hold’em is played with 52 cards. Each player has 2 hole cards and there are 5 community cards with 3 phases, called the *flop*, the *turn* and the *river*. The players have 4 *raise* actions per round each with 4 betting rounds in total. The state representation in this game is a vector of length 72. The first part contains the known cards, namely the hole cards and the already known community cards. The first 13 represents the cards from the Ace of Spade to the King of Spade, followed by the Heart, the Diamond and the Club similarly. The rest of the vector is the number of *raise* actions in each round.

Leduc Hold’em is limited to 6 cards, which are two pairs of King, Queen and Jack. This game is played by 2 players with 2 rounds, where there are 2 *raise* actions in the first one and 4 in the second one. The game is fixed with two-bet and 14 chips maximum.

We implemented a DQN agent in PyTorch. For this, we used the TensorFlow code from RLCard [10] as a base and created a more powerful, more manageable, and easy to use code in PyTorch. This implementation is an advanced Q‑learning agent in two aspects. First, it uses a replay buffer to store past experiences. As we simulate the environment and make an action, we add the state, action, reward and next state to it. When we train our network, we sample from this replay buffer for a more consistent result. Second, to make the training more stable, another Q-network is used as a target network in order to backpropagate through it and train the policy Q-network. These features were first described in [11].

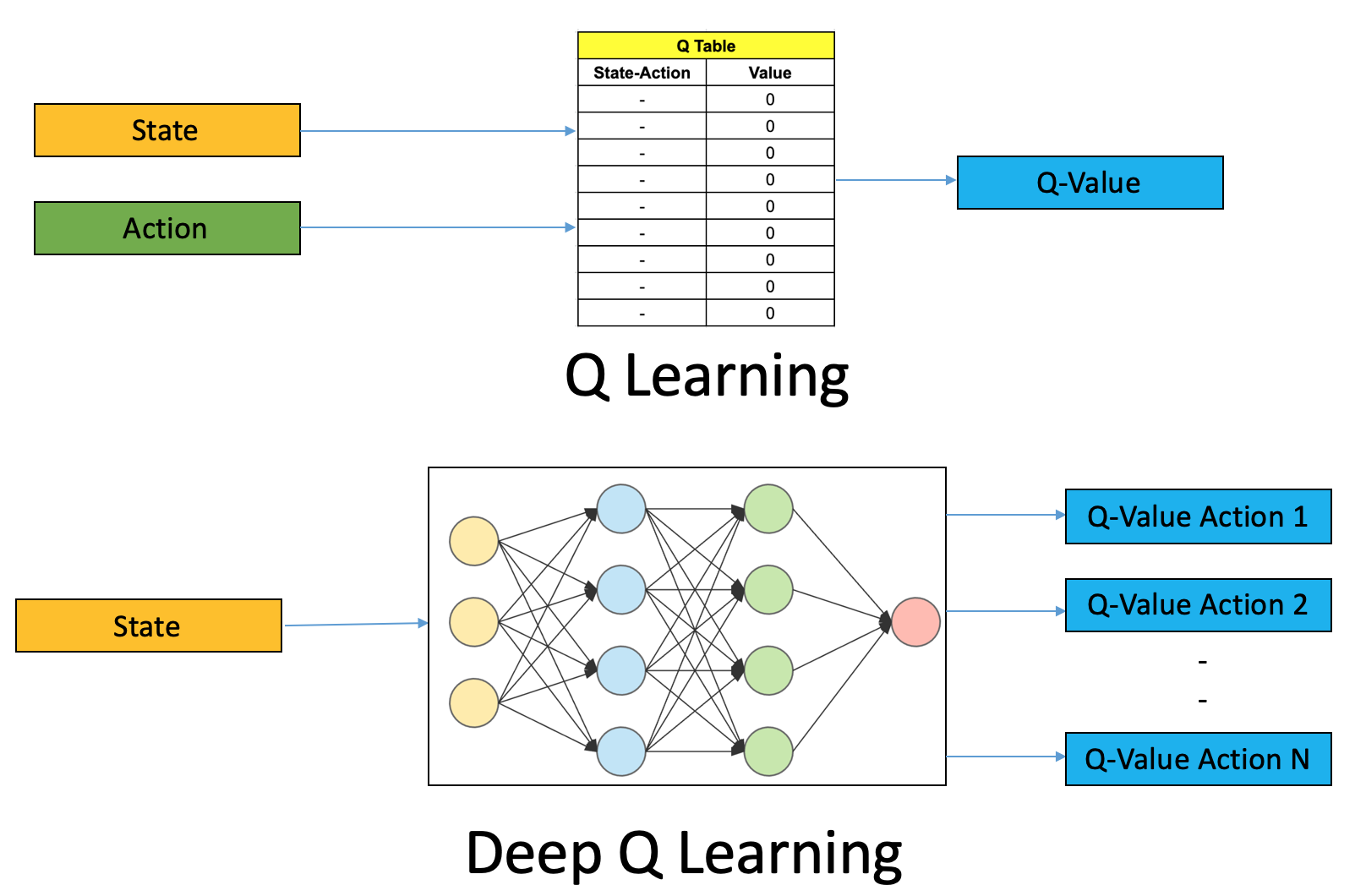


Fig. 2. The architecture of the Deep Q-networks [12]

The purpose of these networks is to estimate a Q-value given the current state, which can be used to determine which action the agent will take. They consist of a simple neural network with the number of states as its input layer and the number of actions as its output layer.

Every step the agent first makes an action based on the epsilon value which is responsible for exploration. If epsilon is high, the agent is more likely to take a random action if it is low, it will use the Q-network to determine the best action. In the early stages of the game, the epsilon starts with a high value “exploring” the environment, and each step its value is reduced by a small amount to the point when it will be near zero.

The agent learns by sampling a minibatch from the replay memory and gets a Q-value for the next state using the policy network and determines the best action for this state. Then it determines the target Q-value using the target network. It calculates the target action using the reward from the replay memory and the target Q-value. Afterward, it backpropagates using this value.

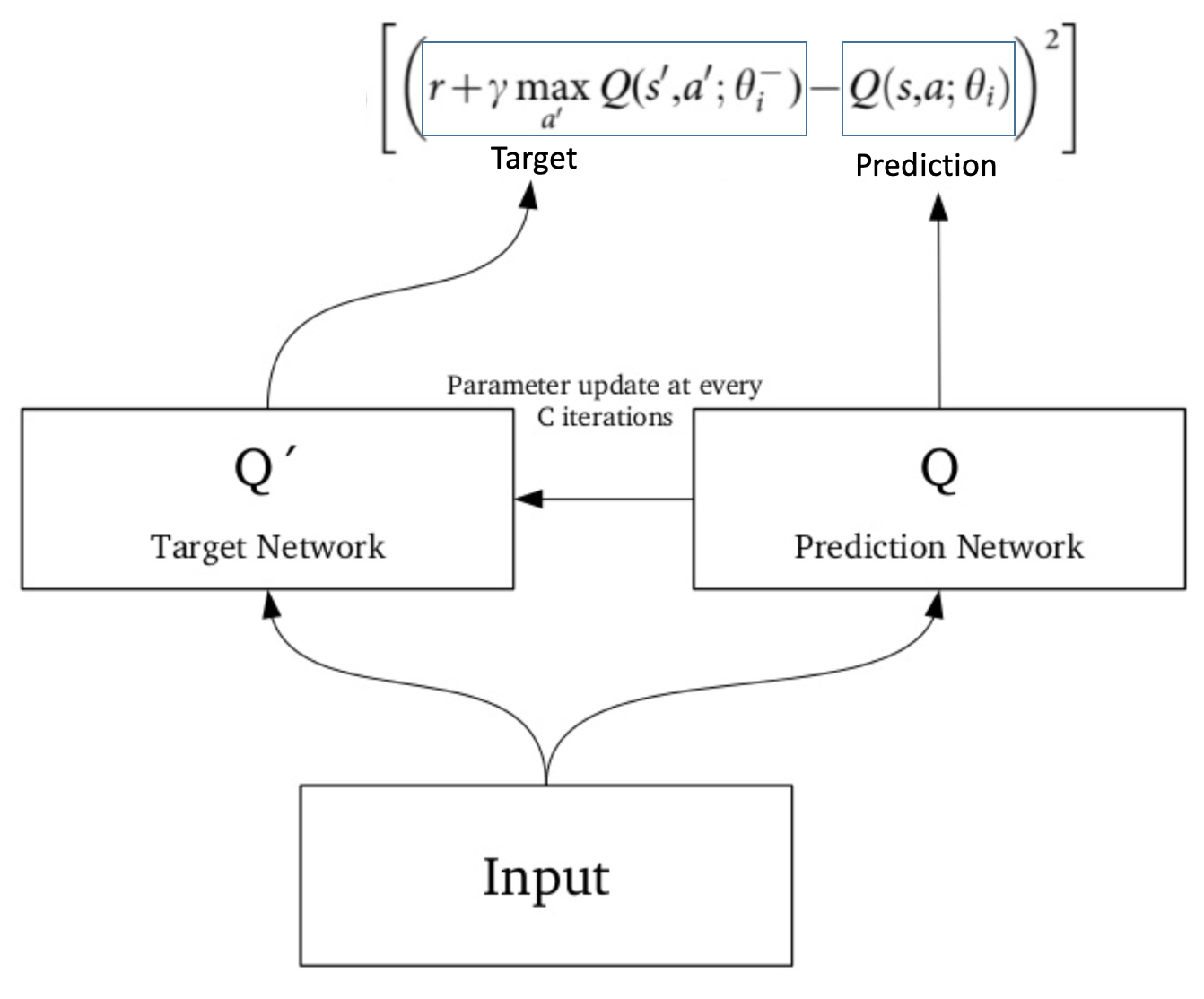


Fig. 3. The inner structure of the DQN networks [12]

First, the agent will “explore” the environment making random actions and getting positive/negative rewards and updating its Q-network accordingly. But as it plays more and more it will take less random actions and has more accurate Q‑values for the given states, playing better and better.

Furthermore, as an extra component, we added the opportunity of a more aggressive playing strategy. In case of the given action has the maximum q-value, the agent chooses the *raise* action as a replacement for it if *raising* is a valid action. Hence, the 3 possible extra settings are to encourage the agent to *raise* instead of *calling*, *checking* and *folding*. We investigate its impact on the performance of the agent.

# Results and Discussion

We trained the DQN agents with 215 different hyperparameter settings against random agents in both environments. During the hyperparameter tuning the number of layers, the replay memory size, the batch size, the discount factor and the learning rate were examined.

The best parameter combinations are shown in *Table I.* It is interesting to observe that 3 parameters, which are the replay memory size, the batch size and the discount factor, lead to the best performance in both environments. The differentiation comes from the parameter of the network layers and the learning rate.

Table I.

Tuned models

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Leduc Hold’em | Limit Hold’em |
| **network layers** | [128, 128, 128] | [128, 128] |
| replay memory | 2000 | 2000 |
| batch size | 64 | 64 |
| discount factor | 0.99 | 0.99 |
| **learning rate** | 0.1 | 0.001 |

Next, we investigate how the proposed agents affect the results in both environments. We trained them 1000 episodes long and in each tenth episode, they were evaluated with 100 games. The reward is calculated from the last 10 evaluations. Table II.and Table III. display the average rewards and their variance in Leduc Hold’em and Limit Hold’em.

The most important finding is that the different versions of the traditional DQN agent have an effect on the performance. Moreover, if the agent prefers *raising* to *calling*, the performance is significantly better.

In the case of the Leduc Hold’em environment, the DQN‑CAR algorithm exceeds the baseline DQN by more than 30% with respect to the achieved average reward. Furthermore, it also performs better than the CFR agent. Meanwhile, the other two methods underperform the classical DQN and CFR as well.

Table II.

Best performance in Leduc Hold'em environment

|  |  |  |
| --- | --- | --- |
|  | mean reward | reward variance |
| CFR | 0.734 | 0.300 |
| DQN | 0.960 | 0.265 |
| **DQN-CAR** | **1.261** | **0.352** |
| DQN-CHR | 0.682 | 0.285 |
| DQN-FR | 0.723 | 0.202 |

Similar results come from the Limit Hold’em environment, where the DQN-CAR outperforms the DQN algorithm by almost 40% improvement.

Table III.

Best performance in Limit Hold'em environment

|  |  |  |  |
| --- | --- | --- | --- |
|  | mean reward | reward variance | |
| DQN | 2.057 | | 0.258 |
| **DQN-CAR** | **2.870** | | **0.465** |
| DQN-CHR | 1.806 | | 0.433 |
| DQN-FR | 1.713 | | 0.276 |

These results with the proposed DQN-CAR algorithm against a random agent shows great performance in both environments. However, it is important to note its limitations. While the aggressive strategy works well against a random agent, the opposite can happen against a stronger opponent. As suggested from [8], the problem of the DQN policy is that it may be highly exploitable since it is easy to find its weaknesses. Indeed, if we train the algorithms against a pre-trained NFSP agent, their performance drops drastically. These evaluation results are shown in Table IV.

Table IV.

Performance against pre-trained agent in Leduc Hold’em

|  |  |  |  |
| --- | --- | --- | --- |
|  | mean reward | reward variance | |
| **CFR** | **0.484** | | **0.288** |
| DQN | -0.049 | | 0.350 |
| DQN-CAR | 0.023 | | 0.350 |
| DQN-CHR | 0.093 | | 0.353 |
| DQN-FR | -0.001 | | 0.3671 |

The 3 new methods slightly perform better than the classical DQN, but the differences are not considerable. This also confirms the fact that the CFR agent can learn better in a difficult scenario. These results with playing against the pre-trained NFSP agent show that the DQN algorithm is not able to adapt in stochastic environments. They also support the findings in recent research [8][11]. It would be worth investigating the same question in the Limit Hold’em environment with a strong pre-trained agent, but such a model is not yet presented for this environment. However, the team of the RLCard proposed to develop more pre-trained agents in the future [8]. Of course, it is possible to train a strong agent with NFSP or CFR by ourself but it was not part of our research.

After comparing the results, we used the DQN-CAR algorithm and created a poker-bot from it in the Leduc Hold’em and Limit Hold’em environment as well. We added a human player and an easy-to-use interface to the game. In this part, we highly relied on the offered elements from the RLCard library [10].

The final poker-bot is available with Docker container and in Google Colab notebook format in our Github repository. The user can choose from them based on which suits better her/his requirements.

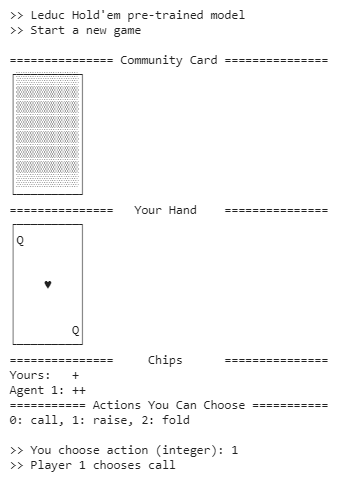


Fig. 4. Snippet from the Leduc Hold’em game in our notebook

Long-term planning poker bots have not been in the main focus of research. To handle this problem, we created a simple CFR agent by editing it’s code in the RLCard library. For computational reasons, our implementation only deals with 2 rounds of play, and it’s specialized in a curious way: its main goal is to have more chips at the end of the round than the opponent has. As CFR learns by traversing the tree and the correcting its policy by the calculated regrets, moving beyond two rounds resulted in really slow learning speed.

Table V.

RESULTS IN REGARDS OF WHO HAS MORE CHIPS IN LEDUC HOLD’EM

|  |  |
| --- | --- |
|  | Win percentage |
| DQN | 71.2 | |
| **DQN-CAR** | **76.99** | |
| DQN-CHR | 70.86 | |
| DQN-FR  CFR | 70.98  75.88 | |

In TABLE V. we can see that our DQN-CAR bot is heavily outperforming the others in this regard. It is important to mention that these results happened against a random agent, so this should not be so surprising. Intuitively, what DQN-CAR does is it encourages the bot to choose raise over call. Against a random agent who happened to raise randomly, raising instead of calling could very often be beneficial. Still, this experimentation should not be treated as a proof that the modern poker playing machines are performing well in every circumstances. These special types of long-planning problems are yet to be explored even in the field of poker, meaning that this field can not be ignored, even though some argued against its importance after Pluribus was made.

# Conclusions

In this paper, we have presented the methodology for creating a poker-bot based on a reinforcement learning algorithm. For this purpose, we selected the classical DQN agent, which performs well in Atari games [11].

As an additional component, we created 3 different versions of the original method to encourage the agent to play more aggressively. After comparing our results, the DQN-CAR algorithm against a random agent outperformed the other ones, including DQN and CFR. However, further investigations lead us to the fact these results highly vary from a game with a strong pre-trained opponent.

In conclusion, the power of the DQN agent in imperfect-information games is limited compared to stronger agents like CFR. Our agents have excellent performance against random agents but not against strong pre-trained agents. Due to this fact, we recommend our poker bot for beginner and intermediate level players especially.

References

1. David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis, “Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm”,2017, Available: arXiv:1712.01815
2. John von Neumann, “Theory of Parlor Games”, 1928
3. Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław “Psyho" Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, Susan Zhang, “Dota 2 with Large Scale Deep Reinforcement Learning,” 2019, Available: arXiv[:](https://arxiv.org/abs/1912.06680)1912[.](https://arxiv.org/abs/1912.06680)06680
4. Johannes Heinrich,David Silver, “Deep Reinforcement Learning from Self-Play in Imperfect-Information Games,” 2016, Available: arXiv:1603.01121
5. Matej Moravcik,Martin Schmid,Neil Burch,Viliam Lisy,Dustin Morrill,Nolan Bard,Trevor Davis,Kevin Waugh,Michael Johanson,Michael Bowling “DeepStack: Expert-Level Artificial Intelligence in Heads-Up No-Limit Poker,” 2017, Available: arXiv:1701.01724v3
6. Noam Brown, Tuomas Sandholm, “Superhuman AI for heads-up no-limit poker: Libratus beats top professionals,” 2018, Available: Science,  359, 418–424 (2018)
7. Noam Brown, Tuomas Sandholm, “Superhuman AI for multiplayer poker,” 2018, Available: Science,   365 (6456), 885-890 (2019)
8. Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo and Xia Hu, “RLCard: A Toolkit for Reinforcement Learning in Card Games,” 2020, [Online]. Available: arXiv:1910.04376.
9. Finnegan Southey, Michael P. Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings and Chris Rayner, “Bayes' Bluff: Opponent Modelling in Poker,” 2012, [Online]. Available: arXiv:1207.1411
10. DATA Lab at Texas A&M University (2020) RLCard [Source code]. <https://github.com/datamllab/rlcard>
11. V. Mnih, K. Kavukcuoglu, D. Silver, “Human-level control through deep reinforcement learning,” *Nature* 518, 529–533, Feb. 2015.
12. Ankit Choudhary, “A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python”, 2019, [Online]. Available: <https://medium.com/analytics-vidhya/a-hands-on-introduction-to-deep-q-learning-using-openai-gym-in-python-b15d7d8597d>

1. [↑](#footnote-ref-1)