Reinforcement learning project

Michele Sanna

February 14, 2025

Abstract

1 Introduction

The aim of this work is to approach the game of Texas Hold'em Poker with reinforcement learning techniques, focusing on the algorithm rather than the neural network architecture and without using human data.

Texas Hold'em Poker is a typical example of an imperfect information game, i.e. a game where some information is hidden from the opponents: in the case of poker each player has its own private cards. The game is also a zero sum game, which means that the total reward across players will be 0. In this work will be treated only the Heads-Up Hold'em variant, that means that the game will be played only between two players; moreover the betting will be discretized in order to simplify the action space of the reinforcement learning agents.

2 A first naive approach

2.1 Observation space

The observation space of a Heads-Up Texas Hold'em game includes, as a public information, the stack sizes (how many chips each player has available for betting), the pot size, the bets of the players and the community cards that have been revealed. The private information instead consists in the private cards of each player. At the end of the poker hand, the combination of the private cards and the community cards will determine the ranking of each player hand, and the player with the best ranking will win the pot.

A potential issue when representing the state of the game consists in the representation of cards. A poker deck contains 52 cards, so each card can be represented as a one-hot encoded vector of 52 entries. To make the representation of the cards more efficient we can represent both the private cards and the community cards as a sum of, respectively, two and five one-hot encoded vectors (we can do this because there can't be two vectors that represent the same card, as the cards are dealt with no repetition). Even with this efficient representation, the observation vector is extremely sparse, with most of the entries consisting in zeros, and the success of the agent will be most likely dependent mostly on the capacity of the neural network to interpret this aspect of the observation space, making the success of the work highly dependent on the neural network architecture.

For this reason, in this work, the cards will not be directly represented in the observation space of the agents, instead their information will be condensed in a well known quantity among poker players, that is the hand equity (i.e. the probability of victory with that hand). Therefore the observation space of the agents for this work will consist in:

- A one-hot encoded representation of the game phase (i.e. how many community cards are revealed)
- The equity of the hand of the agent
- The normalized size of the pot

- The normalized size of both players stack
- The bet of the player
- The bet of the opponent
- A condensed betting history of the opponent (the total opponent bet for each game phase)

2.2 Equity

The hand equity is a vastly used quantity in poker, it refers to the chance your hand has to win the pot at a given moment. A typical way to estimate it is to run a simulation of the possible outcomes if all cards were dealt to the river.

This is also the method used in this project: every time a card is dealt, a Montecarlo-like simulation of many (from 2000 to 8000 depending on the game phase) possible outcomes is launched to update the equity estimate. Obviously, this type of calculation slows down each episode, indeed, it makes them roughly five times slower. However, the loss in speed is more than compensated for by faster convergence.

To get an idea of how precise this estimate of the hand equity is, it has been compared to the estimate of some online equity calculators. The difference, on average, was of 3.9%.

2.3 The algorithm

For this first approach two DQN algorithms were trained playing against each other (details of the training in the Appendix). The agent obtained at the end of this type of training plays in an unusual way: It never folds. The same behavior was obtained with various sets of hyperparameters, and also switching the algorithm from DQN to Double DQN [6].

As we will see later, this is due to the fact that RL algorithms created for single agent environments usually yield suboptimal performance when used in imperfect information games with 2 or more players.

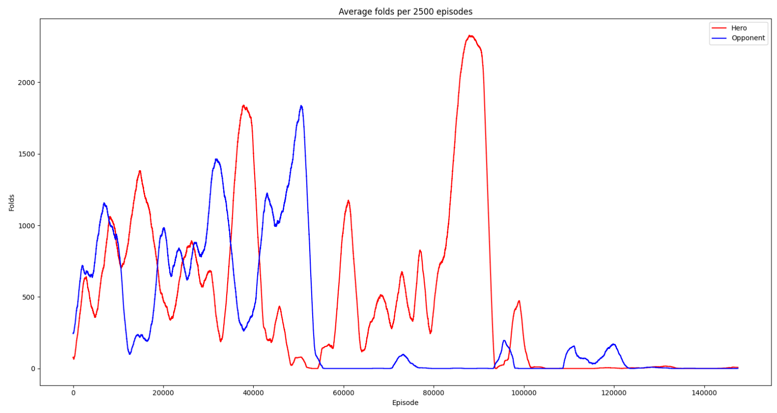


Figure 1: Typical evolution of folding strategy for two DQN algorithms battling each other

3 An alternative approach

3.1 The problem of using single agent RL algorithms

Imperfect information games are considered a challenge for DRL algorithms. Unlike perfect-information games and single-agent settings, the value of an action may depend on the probability that it is chosen, and the game generally requires stochastic strategies to achieve optimal behavior [2]. Also in this setting, a policy A can be the best response to some other policy B, but at the same time be extremely exploitable (i.e., there is a third policy C that can yield high rewards when playing against A). For this reason, algorithms like DQN can fall into cyclical best-response dynamics. Such an approach can fail catastrophically, producing policies that are maximally exploitable [4].

Moreover, if the game has a Nash equilibrium, most classical DRL algorithms are not guaranteed to reach it, and in the case of DQN even the average behavior of the DQN agent is not converging to a Nash equilibrium policy [2].

3.2 More suitable solutions

In the existing literature we can find various attempts to adapt deep reinforcement learning for imperfect information games, in particular for poker. The most cited approaches are ReBel [1], MMD [5] and NFSP [2]. ReBel uses a combination of reinforcement learning and search algorithms, in combination to a multi agent generalization of belief states; MMD uses a modified proximal policy optimization algorithm with a different regularization, and NFSP takes inspiration from the fictitious play algorithm and proposes a deep reinforcement learning adaptation of it. In this work NFSP will be used; the choice was led by the ease of implementation with respect to ReBel and the easier math with respect to MMD. The choice wasn't led by performance considerations, indeed MMD has likely better performance than NFSP.

3.3 NFSP (Neural Fictitious Self Play

As said, NFSP is an adaptation of the fictitious play algorithm. FP [3] comes from game theory, and consists in two agents that play against each other using the best-response strategy to the opponent's average (over the past actions) strategy. The average strategies of fictitious players converge to Nash equilibria in certain classes of games. NFSP adapts FP with an algorithm that resembles two DQN nets that play against each other, with the addition of a average behaviour net that approximates the average behaviour of each player. The average behaviour net plays with a probability $1 - \eta$, while with probability η the algorithm uses an epsilon-greedy strategy. If $\eta = 0$ the player becomes a DQN player.

Below is the original algorithm from the NFSP paper [2]:

The implementation used in this work differs a bit from the original, because it uses a DDQN loss function. Also, the results obtained in the original paper of NFSP in the game of Texas Hold'em poker are obtained with a different game state representation (they represented cards instead of using equity).

4 Experiments and results

To evaluate the effectiveness of NFSP it will be compared to the policies obtained by the two DDQNs playing against each other. Unfortunately, there is no exact and feasible way to determine if one policy is better than another in Texas Hold'em Poker. Usually, if a game is simple enough, we can compute the exploitability of a policy, that measures how vulnerable it is to being taken advantage of by an optimal opponent. In a small and tabular game this quantity can be calculated solving an optimization problem, but for large and continuous-state games this is impossible, and we have to rely on an approximation of a best-response opponent, and so an approximation of exploitability. A possible approximation of exploitability can be obtained with a DQN best-response, this approach is also used in other works [5]. The problem with this solution, besides the fact that is only an approximation, is that the results can be very fluctuating, with high variance between a test and another.

Another approach can simply be the head-to-head evaluation, but also these evaluations can be unsatisfactory because of the intransitive dynamics typically present in imperfect-information games (a good example is rock-paper-scissors). In this work, in order to give a more comprehensive overview, both of these evaluations will be used. In addiction, the policies obtained with both algorithms will be evaluated against different types of random players.

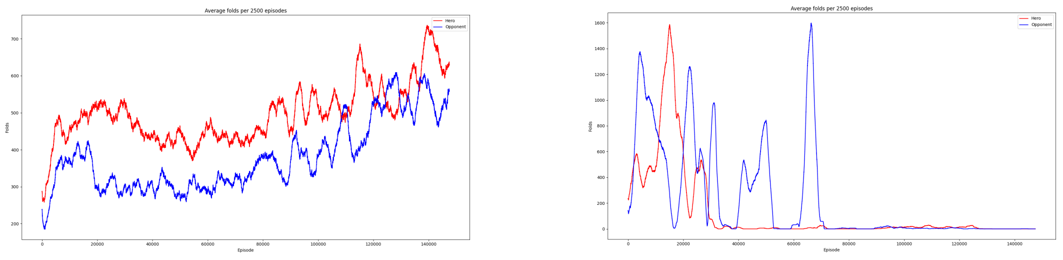
4.1 Training and experiments setting

The following comparisons will be between policies obtained with a DDQN vs DDQN training and policies obtained with the NFSP algorithm; in both cases the training consisted of 150k episodes. The

Algorithm 1 Neural Fictitious Self-Play (NFSP) with fitted Q-learning

```
Initialize game $\Gamma$ and execute an agent via RunAgent for each player in the game
function RunAgent($\Gamma$)
    Initialize replay memories $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir)
    Initialize average-policy network $\Pi(s, a; \theta^{\Pi})$ with random parameters $\theta^{\Pi}$
    Initialize action-value network $Q(s, a; \theta^{Q})$ with random parameters $\theta^{Q}$
    Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
    Initialize exploration parameter $\eta$
    for each episode do
        Set policy $\sigma \leftarrow$
            $\begin{cases} \epsilon\text{-greedy}(Q), & \text{with probability } \eta \\ \Pi, & \text{with probability } 1 - \eta \end{cases}$
        Observe initial information state $s_1$ and reward $r_1$
        for $t = 1, T$ do
            Sample action $a_t$ from policy $\sigma$
            Execute action $a_t$ in game and observe reward $r_{t+1}$ and next information state $s_{t+1}$
            Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reinforcement learning memory $\mathcal{M}_{RL}$
            if agent follows best response policy $\sigma = \epsilon$-greedy$(Q)$ then
                Store behavior tuple $(s_t, a_t)$ in supervised learning memory $\mathcal{M}_{SL}$
            end if
            Update $\theta^{\Pi}$ with stochastic gradient descent on loss
                $\mathcal{L}(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}[-\log \Pi(s, a \mid \theta^{\Pi})]$
            Update $\theta^{Q}$ with stochastic gradient descent on loss
                $\mathcal{L}(\theta^{Q}) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s', a' \mid \theta^{Q'}) - Q(s, a \mid \theta^{Q})\right)^2\right]$
            Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
        end for
    end for
end function
```

differences between the two types of policies are noticeable by looking at the evolution of the average number of folded hands during training. While NFSP keeps a balanced and ”human-like” number of folds, DDQN vs DDQN converges to a never folding strategy.



(a) NFSP (b) DDQN vs DDQN

Figure 2: Average number of folded hands (moving average over a 2500 episodes window)

4.2 Exploitability

As said, for a game with a large or continuous state space is not possible to compute exact exploitability. In this paragraph we’re approximating exploitability with a DDQN best-response trained on 100k episodes.

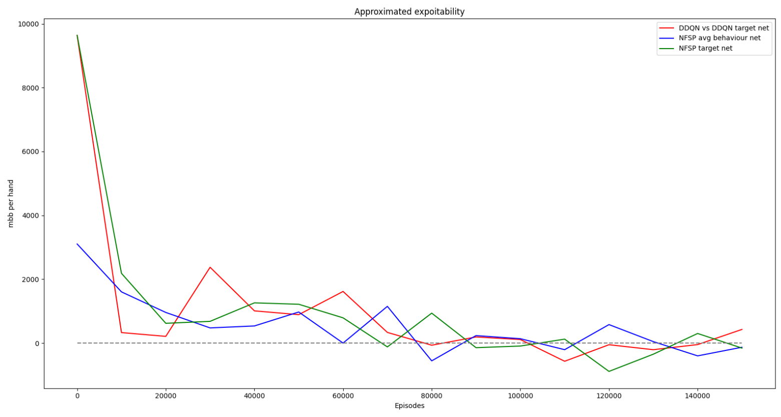


Figure 3: Approximated expoitability with a DDQN best-response

4.3 Evaluation against random players

In this paragraph, the strategies obtained with the two different types of training will be compared with three different types of random players: a player that takes each action with equal probability, a player that only goes all-in and a player that only checks or bets a small amount (up to 8 times the big blind).

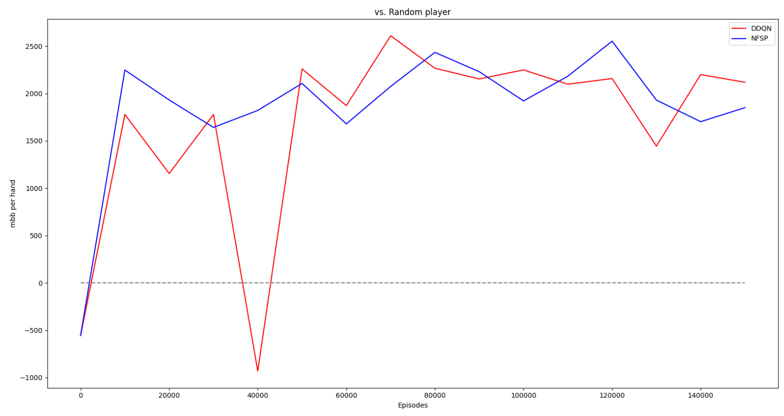


Figure 4: Performance against a random player

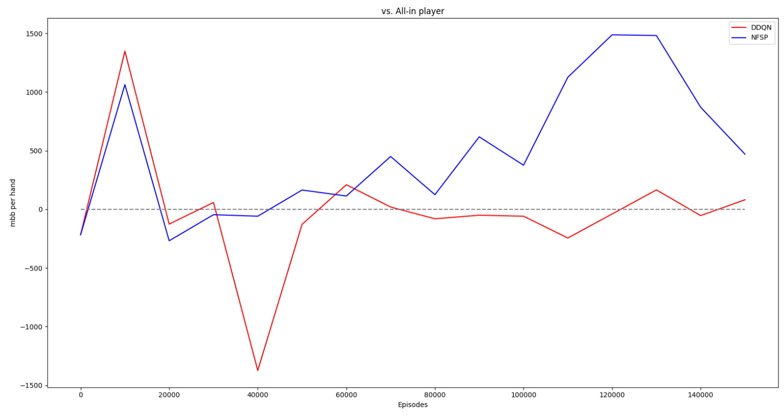


Figure 5: Performance against a player that always goes all-in

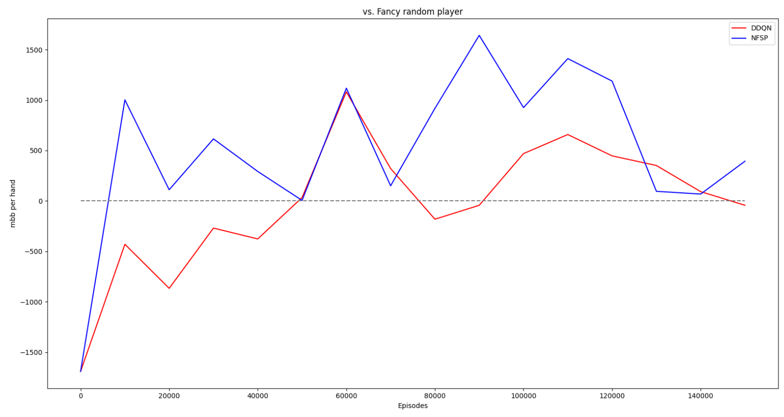


Figure 6: Performance against a player that only checks or bets a small amount

4.4 Head-to-head

Table 1: DDQN vs. DDQN advantage over NFSP. Head-to-head performance over training

Episodi	**mBb per hand**
0	0.0
10000	-1216.2
20000	-1645.1
30000	768.475
40000	-10662.34
50000	123.68
60000	1355.93
70000	369.98
80000	472.78
90000	200.22
100000	135.68
110000	933.87
120000	-67.15
130000	615.74
140000	1469.11
150000	449.6

References

[1] Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *CoRR*, abs/2007.13544, 2020.

[2] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016.

[3] Julia Robinson. An iterative method of solving a game. *Ann. Math.*, 54(2):296, September 1951.

[4] Max Rudolph, Nathan Lichtle, Sobhan Mohammadpour, Alexandre Bayen, J. Zico Kolter, Amy Zhang, Gabriele Farina, Eugene Vinitsky, and Samuel Sokota. Reevaluating policy gradient methods for imperfect-information games. 2025.

[5] Samuel Sokota, Ryan D'Orazio, J. Zico Kolter, Nicolas Loizou, Marc Lanctot, Ioannis Mitliagkas, Noam Brown, and Christian Kroer. A unified approach to reinforcement learning, quantal response equilibria, and two-player zero-sum games. 2023.

[6] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. 2015.

Appendix

Training details for DDQN vs. DDQN

A network of 4 hidden layers, each of 256 neurons with ReLU activation functions. The deep learning optimizer was AdamW, with a learing rate of 5e-4 and a batch size of 256. For the DDQN algorithm, gamma was set to 1, epsilon to 0.12 (with a epsilon decay down to 0.0001 after 2 thirds of the episodes) and tau to 400. The size of the replay memory was of 10000.

Training details for NFSP

The hyperparameters of NFSP were the same of the DDQN algorithms, with the addiction of the eta parameter, set to 0.2. The size of the reservoir memory was of 100000.

Author note: Does it really makes sense the reservoir mechanism for a memory so big?