Reinforcement Learning for Texas Hold'em Poker

Michele Sanna

Motivation and Focus

This project explores the application of reinforcement learning to Heads-Up Texas Hold'em Poker, a zero-sum, imperfect information game. The focus lies on algorithmic performance rather than network architecture, using no human data.

Simplifying the Game

To handle complexity:

- Betting is discretized.
- Only the two-player (Heads-Up) variant is considered.
- Card information is abstracted using hand equity — the estimated probability of winning.

This abstraction avoids sparse card representations and accelerates learning.

Observation Space

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)

Equity Estimation

Monte Carlo simulations (2k–8k samples) are used to compute hand equity. Though computationally expensive, this significantly improves training convergence. Accuracy compared to online calculators shows a mean deviation of just 3.9%.

DQN: Deep Q-Network

DQN uses a neural network to estimate action values:

$$Q(s, a; \theta) \approx Q^*(s, a)$$

During training, the target is computed as:

$$y = r + \gamma \max_{a'} Q(s', a'; \theta)$$

Problem: The max operator can select and evaluate the same overestimated value, leading to:

- Overoptimistic Q-values
- Instability in training
- Poor performance in stochastic or noisy environments

Double DQN: Reducing Overestimation Bias

Double DQN decouples action selection and evaluation:

$$y = r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta), \theta^-)$$

Where:

- θ is the current network (used for action selection)
- θ^- is the target network (used for action evaluation)

Benefits:

- Reduces overestimation bias
- Improves stability
- Often leads to better policies in practice

Naive Approach: DDQN vs DDQN

Two Double DQN agents were trained by self-play. Despite variations in hyperparameters, all learned a suboptimal strategy: never folding. This reflects a known limitation—single-agent RL algorithms often fail in multi-agent, imperfect-information games.

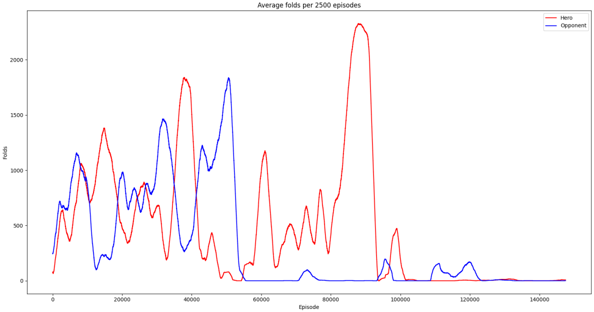


Figure: Folding behavior during naive DDQN vs DDQN training

Limitations of Standard RL

In imperfect-information settings:

- Optimal strategies are often stochastic.
- Best-response policies can be exploitable.
- Classical RL algorithms lack convergence guarantees to Nash equilibria.

These factors cause instability or cycling in self-play dynamics.

Better Solutions: NFSP over ReBel and MMD

Several approaches address imperfect-information games in deep RL:

ReBel: Combines reinforcement learning with search and belief state modeling.
MMD: Uses a modified PPO with tailored regularization.
NFSP: Based on *fictitious play*, where agents respond to the opponent's average strategy.

This work uses NFSP due to its:

- Simpler implementation than ReBel
- Simpler math than MMD

A More Principled Approach: NFSP

Neural Fictitious Self-Play (NFSP) adapts fictitious play to deep RL. It mixes:

- A best-response DQN policy (used with probability η)
- An average-policy network (used with probability $1 - \eta$)

Agents alternate between these strategies, gradually converging to more stable behavior.

NFSP Algorithm Overview

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

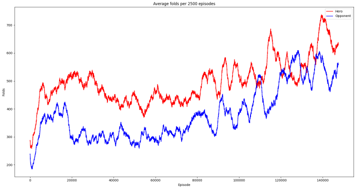
Initialize game Γ and execute an agent via RUNAGENT for each player in the game
function RUNAGENT(Γ)
  Initialize replay memories $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir)
  Initialize average-policy network $\Pi(s, a \mid \theta^{\Pi})$ with random parameters θ^{Π}
  Initialize action-value network $Q(s, a \mid \theta^{Q})$ with random parameters θ^{Q}
  Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
  Initialize anticipatory parameter η
  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 σ
      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 behaviour tuple (s_t, a_t) in supervised learning memory $\mathcal{M}_{SL}$
      end if
      Update θ^{Π} with stochastic gradient descent on loss
        $\mathcal{L}(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log \Pi(s, a \mid \theta^{\Pi})\right]$
      Update θ^{Q} with stochastic gradient descent on loss
        $\mathcal{L}\left(\theta^{Q}\right) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s', a' \mid \theta^{Q'}) - Q(s, a \mid \theta^{Q})\right)^2\right]$
      Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
    end for
  end for
end function

Figure: Original NFSP algorithm

This implementation uses DDQN as the underlying RL method, and equity-based state representation.

Training Comparison

Both NFSP and DDQN-vs-DDQN agents were trained for 150k episodes. The DDQN agents collapsed into non-folding behavior, while NFSP maintained a balanced strategy.



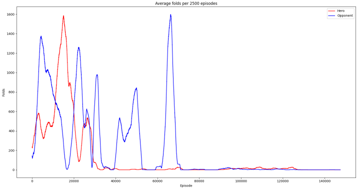


Figure: Fold frequency over time

Approximate Exploitability

Exploitability was approximated by training a DDQN best-response.

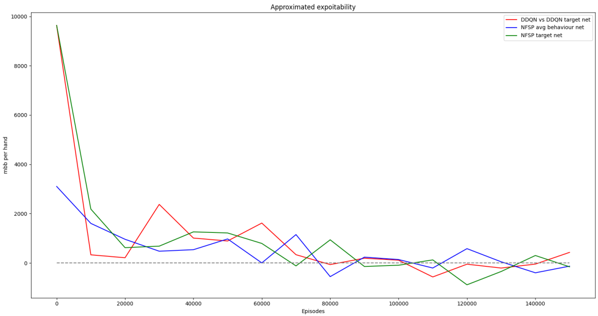


Figure: Approximate exploitability

Performance vs Randomized Opponents

Each agent was evaluated against:

- A uniformly random player
- A player that always goes all-in
- A conservative bettor (check/small bet)

Performance vs Randomized Opponents

Figure: Performance vs random strategy

Performance vs Randomized Opponents

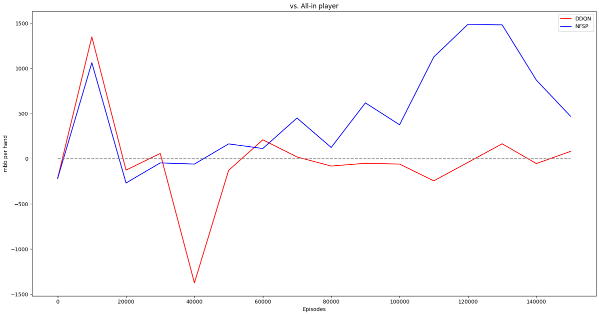


Figure: Performance vs all-in strategy

Performance vs Randomized Opponents

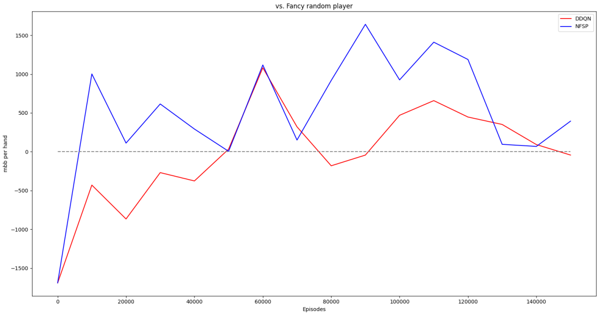


Figure: Performance vs conservative bettor

Head-to-Head

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

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