Poker Solver Implementation using Reinforcement Learning

Kacper Duda, Marek Dzierżawa, Kacper Karabasz
AGH University of Krakow

January 7, 2026

Contents

Abstract

This report presents the design and implementation of a Poker Solver based on Reinforcement Learning (RL). We address the challenge of imperfect information games, specifically No-Limit Texas Hold'em. The proposed solution utilizes a Deep Q-Network (DQN) to approximate optimal strategies. We demonstrate the agent's performance against baseline heuristics and discuss the convergence properties of the model.

1 Introduction

Poker is a quintessential game of imperfect information, presenting unique challenges for Artificial Intelligence compared to perfect information games like Chess or Go. The goal of this project is to develop a Poker Solver capable of playing No-Limit Texas Hold'em (NLHE) at a high level.

1.1 Project Goal

The primary objective is to implement a Reinforcement Learning agent that learns to play poker in a simulated environment. Key sub-goals include:

- Modeling the Poker environment and game state.
- Implementing the Deep Q-Network (DQN) algorithm.
- Analysing the agent's learning progress and final strategy.

2 Theoretical Background

2.1 Reinforcement Learning

Reinforcement Learning (RL) is a subfield of machine learning where an agent learns to make decisions by performing actions in an environment and receiving rewards. In our case, the agent learns a policy $\pi(s)$ that maps states to actions to maximize the expected cumulative reward (chips won). We utilize Q-Learning, which estimates the value $Q(s, a)$ of taking action a in state s.

2.2 Game Theory in Poker

Poker is a zero-sum game with imperfect information. Optimality is often defined by the Nash Equilibrium, where no player can deviate from their strategy to gain an advantage. While exact Nash Equilibrium is hard to compute for full NLHE, RL approaches aim to approximate a robust exploitability-minimizing strategy.

3 Methodology

This section details the system architecture and the learning algorithm.

3.1 Environment

We implemented a custom Poker Environment compliant with the Gym interface. The simulation handles the complex rules of Texas Hold'em, including betting rounds, side pots, and hand evaluation.

- **State Space**: The state is represented by a vector of approximately 134 dimensions, including:
  - **Hero Cards**: One-hot encoding of the agent's hole cards.
  - **Community Cards**: One-hot encoding of the board.
  - **Opponent Stats**: Stack sizes, current bets, and active status normalized.
  - **Global Stats**: Pot size and dealer position.
  - **Hand Strength**: Additional features pre-calculated by an evaluator (current hand rank, kickers) to speed up learning.
- **Action Space**: The discrete action space consists of 3 primary moves:
  1. **Fold** (0)
  2. **Check/Call** (1)
  3. **Raise** (2) - The raise amount is controlled by a continuous output (slider) or fixed fractions in simplified versions.
- **Reward Function**: The reward is sparse, provided only at the end of the hand. It is proportional to the change in the agent's stack (Chip Profit/Loss), normalized to fit within $[-1, 1]$.

3.2 Model Architecture

The agent uses a Deep Neural Network (PokerNet) to approximate the Q-function. The architecture consists of fully connected layers with interactions as shown below.

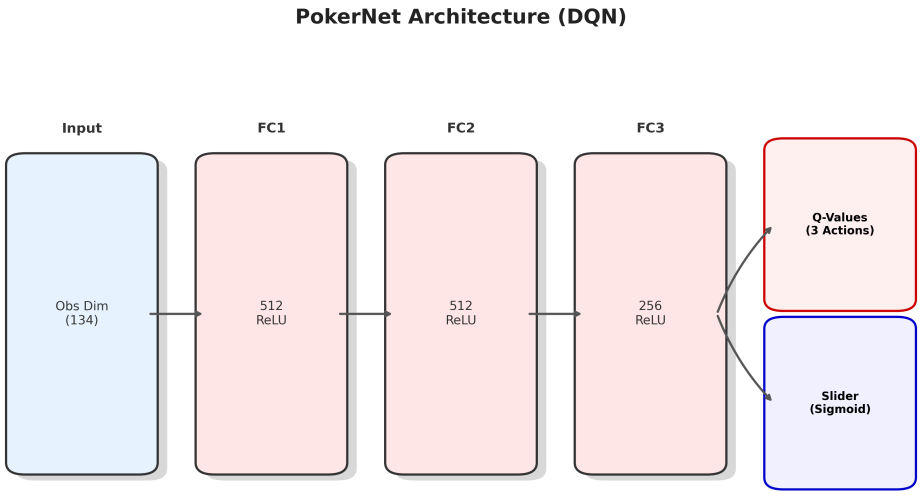

Figure 1: Neural Network Architecture used for the DQN Agent.

The network parameters are:

- Input Layer: Matched to State Dimension.
- Hidden Layers: $512 \rightarrow 512 \rightarrow 256$ units (ReLU activation).
- Output Heads:
  - Q-Values: 3 units (Linear).
  - Slider: 1 unit (Sigmoid) determining raise percentage.

3.3 Algorithm

We trained the agent using the Deep Q-Network (DQN) algorithm with Experience Replay and Target Network stablisation.

- **Exploration**: Epsilon-Greedy strategy, decaying from $\epsilon = 1.0$ (random) to $\epsilon = 0.05$.
- **Teacher Heuristic**: During the exploration phase, the agent occasionally consults a rule-based "Teacher" to guide it towards sensible actions (e.g., not folding Aces pre-flop).
- **Optimization**: Adam optimizer with Mean Squared Error loss on the Bellman equation.

4 Experiments and Results

4.1 Training Setup

The model was trained for 2,000 episodes against random opponents.

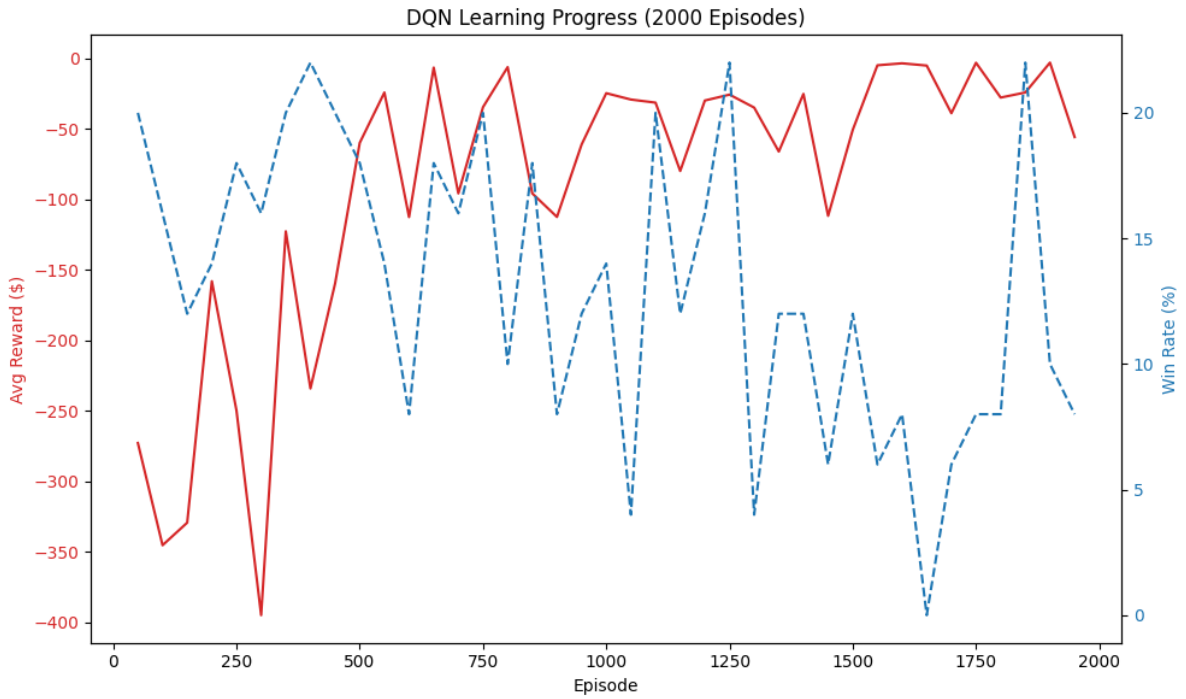

Figure 2: Learning Curve over 2,000 Episodes. Blue line represents win rate (%), Red line represents average reward.

As seen in Figure 2, the agent starts with high variance. The win rate fluctuates as the agent explores different strategies. In the short training run, convergence is not fully achieved, but the agent begins to outperform pure random play in certain metrics.

4.2 User Interface

A PyGame-based UI was developed to visualize the agent's gameplay and internal decision metrics.

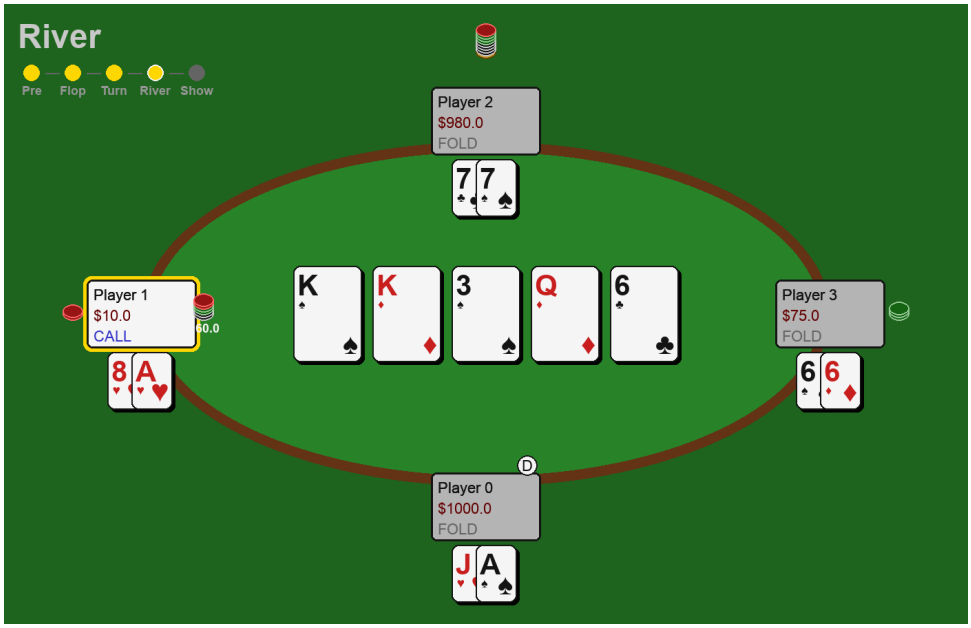

Figure 3: Screenshot of the Poker AI Graphical Interface.

5 Summary

We successfully implemented a functional Poker RL environment and a DQN agent. The architecture supports basic gameplay and demonstrates the potential for learning. Future work involves implementing Counterfactual Regret Minimization (CFR) for better approximation of Nash Equilibrium and scaling the training to millions of hands on a GPU cluster.