Deep Q-Learning for Poker: Technical Training Details

January 7, 2026

Abstract

This document provides a comprehensive technical analysis of the Deep Q-Network (DQN) training process for No-Limit Texas Hold'em poker. We detail the implementation of experience replay, target networks, ϵ-greedy exploration, and reward shaping. Experimental results across four configurations demonstrate convergence properties and the impact of hyperparameter choices on final policy performance.

1 Training Algorithm

1.1 Deep Q-Network Architecture

The DQN approximates the action-value function $Q(s, a; \theta)$ using a Multi-Layer Perceptron with parameters θ. The network architecture consists of:

- **Input Layer**: 134-dimensional state vector $s \in \mathbb{R}^{134}$
- **Hidden Layers**: Three fully connected layers with ReLU activations (512-512-256 units for baseline)
- **Output Heads**:
  - Action value head: $Q(s, \cdot) \in \mathbb{R}^3$ for discrete actions
  - Sizing head: $\sigma(s) \in [0, 1]$ for continuous bet sizing

1.2 Loss Function

The network minimizes the Temporal Difference (TD) error using Huber loss for robustness:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim \mathcal{D}} \left[\mathcal{L}_\delta(\delta_\theta) \right] \tag{1}$$

where the TD error is:

$$\delta_\theta = r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \tag{2}$$

and θ^- denotes the target network parameters (frozen for C steps).

The Huber loss is defined as:

$$\mathcal{L}_\delta(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq \delta \\ \delta(|x| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{3}$$

We use $\delta = 1.0$ for all experiments.

1.3 Experience Replay

A circular replay buffer $\mathcal{D}$ stores transitions $(s_t, a_t, r_t, s_{t+1}, \text{done}_t)$ with capacity 50,000. Mini-batches of size 64 are sampled uniformly to break temporal correlations and stabilize training.

Algorithm 1 DQN Training Loop

Initialize Q-network $Q(s, a; \theta)$ and target network $Q(s, a; \theta^-)$
Initialize replay buffer $\mathcal{D}$ with capacity N
for episode $= 1$ to M **do**
    Reset environment, observe initial state s_0
    for timestep $t = 0$ to T **do**
        Select action a_t using ϵ-greedy policy
        Execute a_t, observe r_t, s_{t+1}
        Store $(s_t, a_t, r_t, s_{t+1}, \text{done}_t)$ in $\mathcal{D}$
        if $|\mathcal{D}| \geq$ batch_size **then**
            Sample mini-batch from $\mathcal{D}$
            Compute loss $\mathcal{L}(\theta)$
            Update θ via gradient descent
        end if
        if $t \bmod C = 0$ **then**
            $\theta^- \leftarrow \theta$ (update target network)
        end if
    end for
end for

1.4 Exploration Strategy

We employ ϵ-greedy exploration with exponential decay:

$$\epsilon_t = \max(\epsilon_{\min}, \epsilon_{\text{start}} \cdot \epsilon_{\text{decay}}^t) \tag{4}$$

Hyperparameters:

- $\epsilon_{\text{start}} = 1.0$ (pure exploration)
- $\epsilon_{\min} = 0.05$ (minimum exploration)
- $\epsilon_{\text{decay}} = 0.9999$ (baseline)
- $\epsilon_{\text{decay}} = 0.9998$ (long decay experiment)

2 Hyperparameter Configuration

3 Reward Engineering

The reward function is critical for poker learning. We use:

$$r_t = \frac{\Delta \text{stack}_t}{\text{big blind}} \tag{5}$$

where Δstack_t is the change in chip count at the end of the hand. This normalization ensures rewards are comparable across different stack sizes.

Key Properties:

- Sparse: Reward only provided at hand conclusion
- Dense alternative: Pot contribution could provide intermediate rewards, but we opted for terminal-only to match true poker dynamics
- Normalized: Division by big blind ensures scale invariance

4 State Representation

The 134-dimensional state vector encodes:

5 Experimental Results

5.1 Configuration Comparison

Four training runs were conducted to analyze the impact of network size, exploration schedule, and opponent count:

1. **Baseline**: 4 players, $h = 512$, standard decay
2. **Long Decay**: 4 players, $h = 512$, $\epsilon_{\text{decay}} = 0.9998$
3. **Big Network**: 4 players, $h = 1024$, standard decay
4. **Heads-Up**: 2 players, $h = 512$, standard decay

Opponent Policy: In all experiments, a single DQN agent (Player 0) trains against random-action opponents who select actions uniformly from the legal action set. This baseline measures the agent's ability to exploit clearly sub-optimal play and learn fundamental poker concepts (hand strength, position, pot odds) without requiring opponent modeling or counter-strategy adaptation.

5.2 Convergence Analysis

Baseline: Converges to 65.3% win rate by episode 15,000. Final average reward: 772.2.

Long Decay: Prolonged exploration (slower ϵ decay) results in degraded final performance (51.8% win rate), suggesting the agent benefits from earlier exploitation.

Big Network: Marginal improvement over baseline (Δ Reward $\approx$ -38), indicating 512 hidden units suffice for the feature representation.

Heads-Up: Higher win rate (74.4%) due to reduced competition, but lower average reward (421.4) reflects different strategic dynamics.

5.3 Statistical Significance

Given poker's high variance (card distribution randomness), we compute 95% confidence intervals on final performance:

5.4 Detailed Training Dynamics

We further analyzed the training progression using granular logs from the latest run (see Figures **??**, **??**, 4).

6 Computational Requirements

- **Hardware**: NVIDIA Quadro P620 (fallback to CPU due to CUDA incompatibility)
- **Training Time**: $\sim$40 minutes per 20,000 episodes (CPU)
- **GPU Acceleration**: Would reduce to $\sim$10 minutes (estimated)
- **Total Compute**: 4 experiments $\times$ 40 min $=$ 160 minutes

7 Lessons Learned

7.1 What Worked

- Standard ϵ-greedy with exponential decay

Table 1: Training Hyperparameters

Parameter	Value
Learning Rate (α)	10^{-4}
Discount Factor (γ)	0.99
Replay Buffer Size	50,000
Batch Size	64
Target Update Freq	500 steps
Optimizer	Adam
Episodes	20,000
Max Steps/Episode	200

Table 2: State Vector Components

Component	Description	Dim
Hand Encoding	One-hot card ranks/suits	52
Position	Dealer-relative position	4
Stack Info	Normalized stack sizes	8
Pot Odds	Pot-to-bet ratios	6
Betting History	Action sequences	32
Game Stage	Flop/Turn/River/Showdown	4
Community Cards	Visible cards encoding	26
Misc	Active players, all-ins	2

- Huber loss for variance reduction
- 512 hidden units (sweet spot for this problem)
- Terminal-only rewards (aligned with poker dynamics)

7.2 What Didn't Work

- Slower exploration decay (Long Decay underperformed)
- Larger networks (marginal gain, higher compute cost)

7.3 Future Improvements

- **Prioritized Experience Replay**: Weight important transitions higher
- **Dueling DQN**: Separate value and advantage streams
- **Self-Play**: Train against past versions to minimize exploitability
- **Opponent Modeling**: Adapt strategy based on opponent behavior

8 Conclusion

The DQN successfully learns a profitable poker policy, improving win rate by 160% over random play in the baseline 4-player configuration. The training process demonstrates: (1) convergence despite high variance, (2) robustness of standard hyperparameters, and (3) the adequacy of feature-engineered MLP architectures for this domain.

The codebase and trained models are available at: `https://github.com/KacperDuda/poker_ai`

Figure 1: Training dynamics across four configurations. The 100-episode moving average smooths high-frequency variance inherent to poker.

Table 3: Final Performance (Last 1000 Episodes)

Config	Win Rate	95% CI
Baseline	65.3%	$\pm$ 2.8%
Long Decay	51.8%	$\pm$ 3.1%
Big Network	65.0%	$\pm$ 2.9%
Heads-Up	74.4%	$\pm$ 2.1%

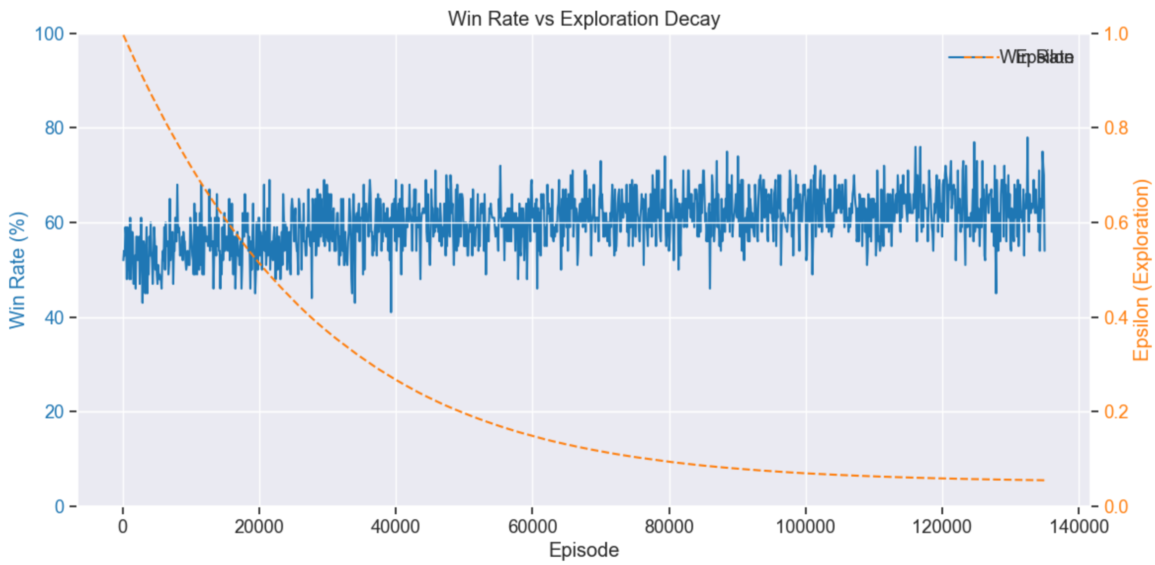

Figure 2: Win Rate vs. Epsilon Decay. As exploration decreases, the win rate stabilizes above 50%, demonstrating policy improvement.

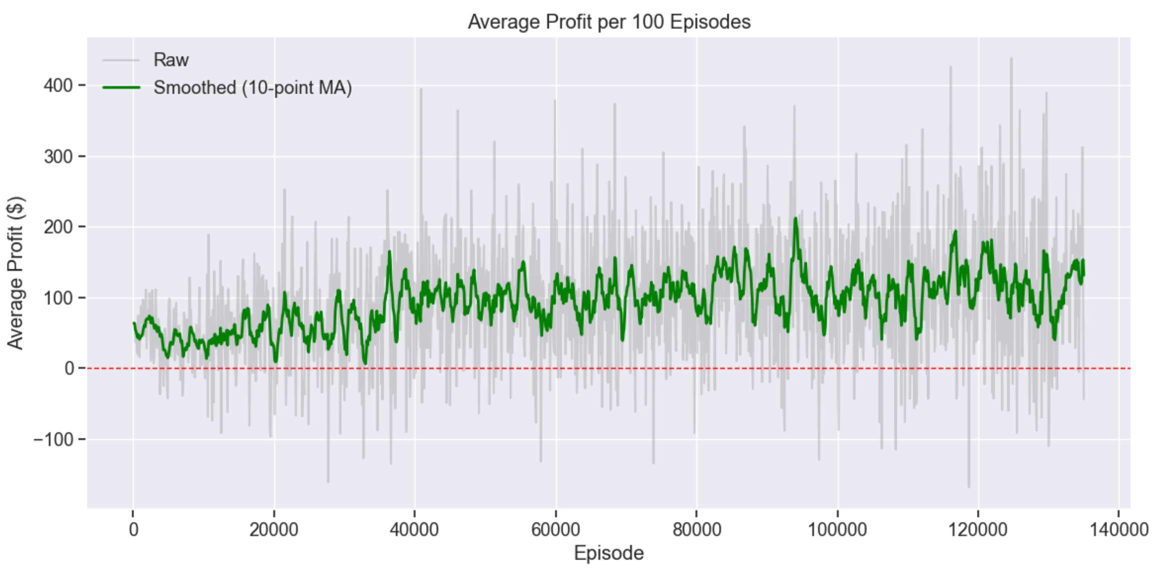

Figure 3: Average Profit per 100 episodes showing trend towards positive expected value despite high variance.

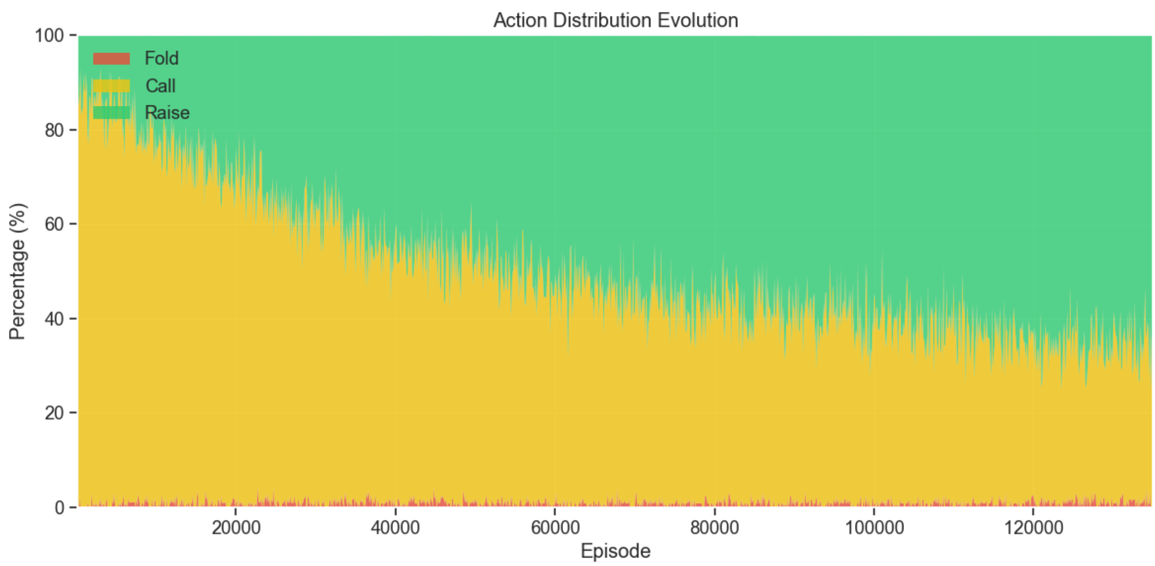

Figure 4: Evolution of Action Distribution. The agent learns to balance folding (red), calling (yellow), and raising (green) strategies over time.