Ivey: A Reinforcement Learning Poker Engine

Aaron Jin
Department of Computer Science
Stanford University
aaronjin@stanford.edu

Ryan Cheng
Department of Computer Science
Stanford University
rcheng07@stanford.edu

1 Introduction

Poker represents a particularly challenging domain for artificial intelligence (AI) and machine learning (ML) due to its combination of imperfect information, strategic depth, and the need to model opponent behavior. While recent research has produced impressive results using techniques like Counterfactual Regret Minimization (CFR), we are interested in exploring how reinforcement learning (RL) approaches can be applied to develop effective poker-playing agents.

More specifically, our project investigates the effectiveness of Q-learning, focusing on preflop decision-making in heads-up (two-player) No-Limit Texas Hold'em. While initially constrained to a simplified push-fold framework where players could only go all-in or fold, we progressively expanded our model to include more sophisticated betting options like pot-sized raises, in order to better reflect real poker dynamics. This incremental approach allows us to validate our learning methods while keeping our research relatively tractable.

The input to our algorithm is a poker game state consisting of: the player's two-card starting hand, position (whether acting first or second), previous betting actions in the current hand, as well as stack sizes and pot size. Additionally, we use Q-learning, implemented through both traditional tabular methods and experimentation with deep learning approaches, to output optimal actions from a discrete set of choices (fold, call, raise, or all-in) for any given game state. Through self-play and iterative improvement, our agent learns to maximize its expected value in chips over long-term play.

Our work demonstrates, that even with relatively simple reinforcement learning techniques, poker agents can develop sophisticated preflop strategies that capture fundamental concepts like position play, hand strength evaluation, and appropriate bet sizing. More importantly, our incremental development process reveals important insights about the relative effectiveness of different training approaches, from dual-agent learning to single-agent self-play, and the crucial role of state space design in achieving optimal, desirable performances.

2 Related Work

Martin Zinkevich and Piccione (2007) were the first authors to introduce the Counterfactual Regret Minimization (CFR) algorithm. CFR builds off the concept of regret matching from Hart and Mas-Colell (2000), where they showed that, in zero-sum imperfect information games, if all agents minimize their "regret," they will eventually approach the game's Nash equilibrium. Specifically, "regret" is calculated in the self-play paradigm where an agent stochastically samples from a probability distribution for a given state corresponding to actions and determines if they would have received more reward had they sampled that action more or less. If the regret is positive, this means that the agent would be better off taking that action more, so they adjust their probability distributions for that state accordingly, and vice versa. CFR has steadily gained adoption since then, specifically for poker, as it is an algorithm uniquely suitable for imperfect information games. Notably, Michael Bowling and Tammelin (2007) used a variant of CFR to fully solve Heads-up Limit Hold'em Poker, a simpler version of poker with approximately 10^{17} possible states. Heads-up Limit Hold'em is able to be fully solved in a finite state-space due to having pot-sized raise limits. However, 6-max No Limit Hold'em, which is the most commonly played version of poker, has far too many states to be solved with current techniques. As a result, much of the recent literature is focused on either improvements to the

CFR algorithm as in Viliam Lisý (2015) or adapting neural networks to approximate the behavior of CFR as in Brown et al. (2019).

However, the scope of nearly all poker-related literature in recent years is attempting to find Game Theory Optimal (GTO) solutions. While these solutions are undoubtedly valuable and provide strong frameworks for understanding poker theory, they make a trade-off of solving for equilibria rather than maximizing expected value. When playing against a GTO-optimal agent, the EV-maximizing approach mimics this pattern by also playing a GTO-optimal strategy. However, in practice, most humans play suboptimally, so having a framework that trains EV-maximizing agents can potentially be even more valuable as a tool for poker players. As a result, for the scope of our project, we chose to use Q-learning instead of CFR as our RL algorithm in order to investigate the potential of solutions that do not necessarily aim to create GTO-optimal solutions.

3 Dataset and Features

Unlike supervised learning approaches that rely on pre-existing datasets, our reinforcement learning system generates its own training data in the structure of Q-tables through self-play. Each poker hand played represents a training episode, with the final chip differential serving as the reward signal. Our agents train on millions of hands, with each hand generating multiple state-action-reward triplets that populate our Q-learning experience buffer.

To highlight, our state space consists of several key components. First, for hand representation, each starting hand is encoded in a standardized format (e.g., "AKS" for Ace-King suited, "75O" for 7-5 offsuit, etc.). We then maintain a complete mapping of all 169 unique preflop hand combinations: 13 pairs, 78 suited combinations, and 78 offsuit combinations. In terms of position information, we implemented a binary encoding of whether the player is first or second to act; this allows agents to learn position-dependent strategies. Action histories are stored as sequential encodings of all previous actions in the current hand, where actions are discretized into a finite set: check, fold, call, or all-in.

To add, the process for generating our training data is extensive where, for each epoch of training, we simulate 100,000 hands, each of which produces branches through potentially multiple decision points, and then use the results of these simulations to update our Q-values. In terms of feature engineering, our key insight was that precomputing and caching certain features dramatically improved training efficiency. More specifically, in our preflop equity table, we now pre-calculate equity values for all possible hand versus hand match-ups. These values are then stored in a lookup table to avoid expensive runtime calculations. Overall, this dramatically increased training speed, since we eliminated Monte-Carlo simulations during training. Moreover, Q-values are normalized relative to the big blind; this creates a consistent scale for evaluating actions across different stack depths, which helps maintain greater numerical stability during learning. Lastly, to analyze our learned strategies, we developed a visualization tool that generates color-coded strategy charts similar to professional poker starting hand charts. These charts map Q-values to colors: blue for folds, green for calls, and red for raises.

4 Methods

Our approach to developing poker-playing agents evolved through several stages of increasing sophistication, centered around Q-learning as our core learning mechanism. Q-learning, a model-free reinforcement learning algorithm, learns a value function $Q(s, a)$ representing the expected future reward for taking action a in state s. The update rule for our Q-function is:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max Q(s', a') - Q(s, a)]$$

- α (learning rate) determines how much new information updates existing Q-values
- γ (discount factor) balances immediate versus future rewards
- R represents the immediate reward
- s' represents the next state
- $\max Q(s', a')$ represents the maximum expected future reward

Our initial implementation employed a dual-agent learning paradigm where two separate agents trained against each other. In this process, each agent maintained its own Q-table and updated their policies independently. Consequently, learning occurred through direct competition versus one

another, where updates were synchronized across episodes. While this approach was intuitive, it suffered from several limitations including slower convergence due to the moving target problem, higher variance in learned strategies, and inefficient use of experience data. These limitations led us to develop a more efficient single-agent learning system where one agent learns both Player 1 and Player 2 positions. After a predetermined number of epochs, Player 1's strategy is copied to Player 2, creating a form of iterative self-improvement. This approach showed significant improvements in performance, notably reducing BB loss and achieving faster convergence.

Additionally, to improve training efficiency, we implemented a more sophisticated batch Q-learning system that aggregates rewards for state-action pairs and updates Q-values in batches. Our experience replay mechanism then stores state, action, reward, and next state tuples. This implementation significantly improved learning stability and made better use of gathered experience data.

For action selection and exploration, we employ an epsilon-greedy strategy with some enhancements. The base exploration rate ϵ is set to 0.1, and we implement legal action masking to prevent invalid moves. The exploration rate progressively reduces over time, and we maintain position-dependent exploration rates to account for the asymmetric nature of poker positions. Thus, when selecting actions, the agent either explores randomly with probability ϵ or exploits its current knowledge by choosing the action with the highest Q-value among legal actions with probability $1 - \epsilon$. Furthermore, our poker engine implements efficient state transitions and reward calculations based on chip differentials. We use equity-based hand strength evaluation and maintain stack-aware action spaces that dynamically adjust based on remaining chip counts. This ensures that our agents learn realistic, logical poker strategies that account for stack depth considerations.

In addition to this, several critical optimizations dramatically improved our training efficiency. For instance, like mentioned previously, we precomputed equity lookup tables to avoid expensive runtime calculations, implemented cached state transitions to reduce computational overhead, and used vectorized batch operations for faster processing. All these optimizations allowed us to train on significantly larger datasets and achieve improved convergence. In order to validate our approach, we also implemented multiple evaluation methods. First, we established a baseline by comparing our agents against a random player, which provided a clear demonstration of sensible learning and convergence. We also cross-validated our agents' strategies against known optimal push-fold charts from poker theory. Additionally, we conducted expected value analysis across different positions and performed strategy consistency checks across similar hands to ensure our agents were learning meaningful, sound patterns rather than overfitting to specific situations.

Overall, the effectiveness of our methods seems particularly evident in the agents' learned behaviors. They demonstrate sophisticated concepts such as position-aware play, appropriate hand strength evaluation, position-specific exploitation tactics, and intelligent bet sizing. To underscore, the transition from dual-agent to single-agent learning proved especially valuable, as it led to more stable strategiesx—both in theory and in practice—while reducing training time.

5 Experiments / Results / Discussion

Our experiments used Q-learning with carefully chosen hyperparameters: a learning rate α of 0.01, discount factor γ of 1 (since there are no immediate rewards), and an initial exploration rate ϵ of 0.1. We selected a relatively small learning rate to ensure stable convergence given the high variance nature of poker outcomes. In addition, we selected the batch size based on the combinatorics of possible hand match-ups: with 169 unique starting hands, there are $169^2 = 28,561$ possible hand combinations. To ensure sufficient sampling of all possible match-ups and account for the various betting sequences that could occur with each match-up, we chose a batch size of over 100,000. This larger batch size ensure that our agent encounters each possible hand match-up multiple times within a single update, leading to more stable learning and Q-value estimates.

To validate our learning framework, we first tested our agent against a random opponent (Figure 1). More specifically, the random opponent selects a random action with equal probability. As such, the resulting strategy chart reveals a set of clear exploitation patterns. Since the Q-learning algorithm implicitly finds the transition probabilities between states and the random agent makes no dynamic decisions, this problem should be fully solvable. In order to verify that our results are approximately correct, we can manually verify some EVs. As an example, for AA, the EV when going all-in with stack sizes of 20 BB each is 14.3 BB. Therefore, if Player 1 has AA and raises, Player 2 will either

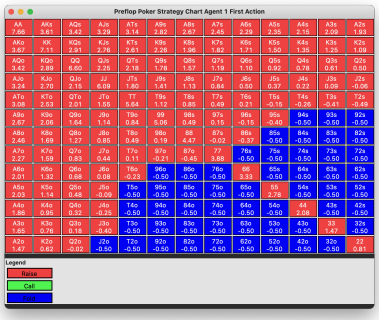

Figure 1: Agent 1 vs. Random

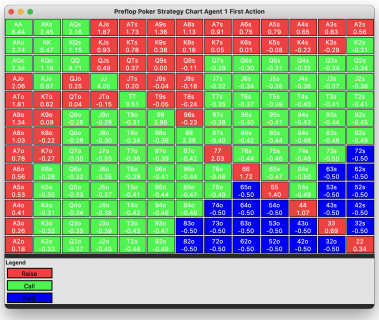

Figure 2: Agent 1 First-to-Act

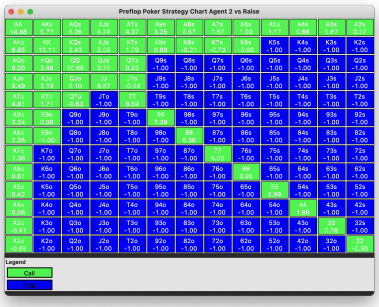

Figure 3: Agent 2 vs. Raise

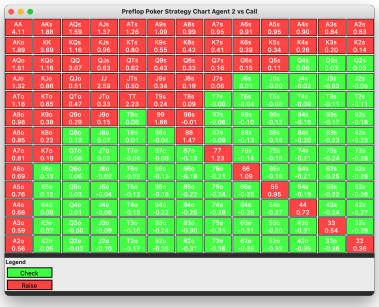

Figure 4: Agent 2 vs. Call

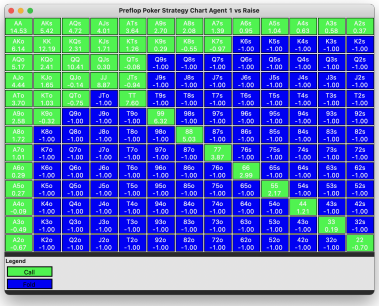

Figure 5: Agent 1 vs. Raise

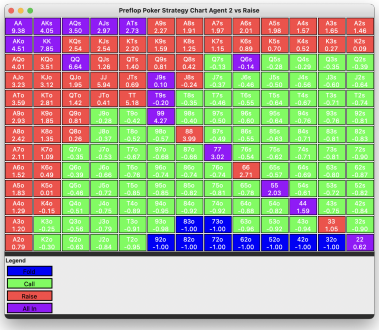

Figure 6: Agent 2 vs. Pot-Sized Raise

call or fold with 50% probability each. As a result, the theoretical EV of raising AA against a random agent should be approximately 7.65 BB; we can see in the chart that the Q-value for AA is 7.66 BB, reinforcing that our training environment works properly.

Our agent undoubtedly performs well against a random strategy baseline. Next, we analyzed the self-play results under the simple push-fold poker variant, which revealed sophisticated strategic adaptions across different positions and scenarios. Figure 2 shows Agent 1's first-to-act strategy, demonstrating a notably more conservative approach compared to playing against a random opponent. This strategic shift makes sense: when facing an intelligent opponent, the agent must account for being exploited if it plays too aggressively. The raising range tightens significantly, with only premium pairs (e.g. AA, KK, QQ) and strong Broadway hands (e.g. AK, AQ, KQ) being only "calls" in order

to trap and lure Agent 2 into putting more money into the pot. The agent also develops a mixed strategy with medium-strength hands like AJ and AT along with medium to lower pocket pairs (e.g. 77-22), sometimes raising and sometimes calling, which is a fundamental concept in poker theory for maintaining balance. Lastly, it conservatively calls with marginal hands (indicated by the green cells), and it folds any weak, unpaired hands that have negative EV.

When examining Agent 2's strategy facing a raise (Figure 3), we observe a highly polarized response pattern. The agent calls with strong hands—whether suited or unsuited, paired or unpaired—which we can see by the general green arrow shape in the chart. On the other hand, it folds most marginal holdings, creating a distinct gap in its calling range. This polarization reflects the correct theoretical adjustment to facing aggression: when facing a raise, continuing with medium-strength hands becomes less profitable due to their reduced implied odds and the difficulty of realizing equity out of position. Notably, pairs show higher calling frequencies than unpaired hands of similar strength, as they have more stable equity against opponents' raising ranges.

Agent 2's strategy when facing a call (Figure 4) reveals position-aware aggression. With position, the agent raises more frequently with medium-strength hands, exploiting the advantage of acting last in future betting rounds. The raising threshold loosens considerably compared to facing a raise, but maintains a logical structure where hand strength correlates with action frequency. This position-based adjustment demonstrates that our agent has learned one of poker's fundamental concepts: the value of position.

Figure 5 illustrates Agent 1's calling strategy versus raises, showing the most strict, conservative tendencies. This asymmetry in aggression makes theoretical sense: being out of position is a significant disadvantage in poker, requiring stronger holdings to continue. The agent maintains high standards for calling, particularly with unpaired hands, reflecting the challenges of playing out of position with hands that need to improve to win.

Finally, we also implemented an alternate version of our poker game where we allow players to make pot-sized raises along with also going all in. In Figure 6, we show Agent 2's strategy when facing an initial raise. Interestingly, we see that this increased complexity leads to even more interesting mixed strategies where most of the strong hands are raised, but the strongest ones and also some weaker hands are raised all-in as bluffs.

Throughout all these scenarios, we observe coherent hand-strength hierarchies and position-based adjustments that align with established poker theory. The agent has learned to balance aggression with caution, adjust its ranges based on position, and maintain theoretically sound response patterns to different actions. These strategic elements emerged purely through self-play, without any explicit programming of poker concepts, which demonstrates that our agent is indeed learning logical poker principles.

6 Conclusion / Future Work

Our project demonstrates that Q-learning, despite its relative simplicity compared to more sophisticated algorithms like CFR, can effectively learn complex preflop poker strategies through self-play. Our agent developed sophisticated concepts including position-based play, hand strength evaluation, and appropriate bet sizing—all without explicit programming of poker theory. The transition from dual-agent to single-agent self-play proved especially effective, leading to more stable learning and better convergence. For future work, the natural progression would be to extend our framework to post-flop play, including the flop, turn, and river stages. This expansion definitely presents some challenges due to the exponential growth in decision points and state space. For instance, while preflop play has a relatively contained state space with 169 possible starting hands and limited betting options, post-flop play introduces 1,755,600 possible flop combinations alone—each creating new decision points and requiring consideration of bet sizing, board texture, and drawing possibilities. Furthermore, with the expanded state-space its likely necessary that we would need to use neural networks as part of a Deep Q-learning framework in order to reduce training time. However, despite these challenges, our success with preflop strategy suggests that our approach could be adapted to post-flop play through careful state space design and potentially incorporating function approximation methods to handle the increased complexity.

7 Contributions

Aaron implemented the charting scripts, calculated the cached preflop matchup equities, and built a Deep Q-learning script, which we ended up not using for our final results. Ryan worked on the original Q-learning scripts for both the push-fold game as well as the full preflop version that also allows pot-sized raises. He furthermore worked on optimizations to make the training scripts run order-of-magnitudes faster. Finally, this report was written collaboratively by both of us.

References

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. 2019. Deep counterfactual regret minimization. In *Proceedings of the 36th International Conference on Machine Learning*.

Sergiu Hart and Andreu Mas-Colell. 2000. A simple adaptive procedure leading to correlated equilibrium.

Michael Bowling Martin Zinkevich, Michael Johanson and Carmelo Piccione. 2007. Regret minimization in games with incomplete information. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Michael Johanson Michael Bowling, Neil Burch and Oskari Tammelin. 2007. Heads-up limit hold'em poker is solved. In *Communications of the ACM, Volume 60, Issue 11*.

Michael Bowling Viliam Lisý, Marc Lanctot. 2015. Online monte carlo counterfactual regret minimization for search in imperfect information games. In *AAMAS*.