Poker Project - Team 47: Modelling Poker Players

AUYOK Sean, CAI Jiaxiu, CHIK Cheng Yao, Joyce YEO Shuhui, ZHUANG Yihui
National University of Singapore
sean.auyok,e0201975,chikchengyao,joyceyeo,yihui@u.nus.edu

1 Introduction

In the quest to understand the human mind, scientists have delved into designing Artificial Intelligence (AI) agents in more structured environments like games. AI agents have long established their dominance in games, perhaps most famously in chess with IBM Deep Blue's win over world champion Garry Kasparov in 1996. The capabilities of AI extend to imperfect information games, where AI agents have beaten top human experts.

An interesting example would be Texas Hold'em Poker, a complex game including elements of chance and bluff. More interestingly, it is not enough to merely win the opponent, but also to value-bet and win by as large a margin as possible. This paper focuses on our team's development of a poker AI for Heads Up Limit Texas Hold'em played under strict time controls.

When designing our agent, the main functionality was for the agent to be rational and able to recognise strong hands to invest in. Given the limited time for our poker agent to decide on the next action, we employed various optimisation strategies to reduce the game complexity for our agent. For instance, we taught our agent to cluster poker hands together (Section 2), to reduce the number of hand variations it considers.

Upon evaluating the strength of a hand, our agent has to be able to make the right decision (to fold, call or raise). We give our agent the ability to make such decisions using Counterfactual Regret Minimisation (CFR), which aims to play a Nash Equlibrium strategy against all opponents. Our CFR agent aggregates opponent behaviour using the opponent's betting sequence (Section 3). At each decision node, our CFR agent takes each action with a certain probability, depending on how the opponent is playing and how strong its cards are (Section 4).

Limitations in the implementation of the CFR agent makes it slow in making real-time decisions. Furthermore, the training process of the CFR agent is inefficient. We circumvent these time constraints in training and decision-making using a Deep-Q Network (DQN) (Section 5). The DQN is a neural network optimised for games like poker, where we want to evaluate the value of making an action.

We then discuss the implications of our CFR agent's theoretically unexploitable play. How can an agent remain unexploitable in a stochastic game? How does the existence of a theoretically unexploitable agent change the way we perceive poker?

2 Modelling Game State

Poker has a plethora of game states an agent can tap on, such as the hand strength, pot size and player stack to name a few. Given the complexity of poker, it would be infeasible to use all the game states in designing our agent

The primary heuristic in a rational poker agent would be the hand strength, because that is how poker's end state is evaluated. The player with the higher hand strength clears the pot. In this section, we focus on how we abstracted hand strengths for our agent.

2.1 Hand Strength

There is extensive research on accurately evaluating hand strength, such as CactusKev, TwoPlusTwo and 7-card. Hand evaluators aim to assign numerical values to hands, and subsequently group cards with similar strengths together. These buckets of cards tend to have similar play styles.

Since hand strengths change across streets as community cards are revealed, the classification key is typically a function of the current hand strength, and the potential hand strength. We considered the theoretical benefits of these methods and the practical limitations of implementation in our final design.

Card Isomorphism
To effectively evaluate hand strengths, it is imperative to simplify the hands using abstraction techniques like card isomorphism. Card isomorphism exploits the fact that a card's suit has no inherent value. It is only useful to know the values of the hole cards and whether they are suited (have the same suit). For example, A♣ A♣ has the same winning chance as A♠ A♠, and these hole cards would be played with similar strategy.

Card isomorphism can reduce the number of game states in the pre-flop street from $\binom{52}{2} = 1326$ to $2 \times \binom{13}{2}$ offsuit combinations and 13 suited combinations, for a total of 169 states. Each state is identified using the value of the cards, and a boolean indicator if they are the suited.

Percentile Bucketing
Unfortunately, card isomorphism does not simplify the game complexity enough. The 169 game states in the pre-flop state

will exponentially increase as we introduce more information on community cards, the pot size and player actions taken. To further reduce the number of game states, we employ a second layer of abstraction, card bucketing, to group cards of similar strengths together.

For instance, two sets of hole cards like K♡ Q♢ and Q♢ J♡ can be handled with a similar strategy as they are offsuit highcards with a possibility of completing a straight. Card bucketing can significantly reduce the branching factor of our game tree.

Ganzfried and Sandholm, the team behind a world-class poker AI (Tartanian), found that using too many buckets meant that the training algorithm could not converge sufficiently fast. Eventually, they settled on a $8-12-4-4$ bucket size for the four different streets. We simplified their model and chose to use the same bucket size of 5 in all streets for our agent. The branching factor is large enough for our agent to differentiate hands, but not too large that any training would be intractable.

Our heuristic for bucketing is the expected win rate of a hand. Using the expected win rate would take into account different combinations of the community cards and opponent hands. Unfortunately, the number of combinations is very large, so we used a Monte-Carlo simulation to generate the win rates. The win rates are used to demarcate two buckets, so each bucket is effectively identified by a minimum and maximum expected win rate.

In each street (pre-flop, flop, turn, river), 1000 different hands were generated. The hands would have two pairs of hole cards and 0, 3, 4 or 5 community cards depending on the street. For each of these 1000 hands, 500 Monte-Carlo simulations were run to determine the win rate of the hands.

We used a percentile-win-rate clustering method, so the first bucket is marked by the win rate of the hand at the 20^{th} percentile. We stored the bucket margins for our CFR agent across different streets, as shown in Appendix A. Each margin demarcates the two adjacent buckets, so there are 4 margins given 5 buckets.

Our results demonstrate the effect of imperfect information in hand evaluation. In the pre-flop stage, the bucket margins were close and the difference in win rates of the first and last margin was only 0.182. Most cards in the pre-flop stage had a fairly similar win rate because of the uncertainty in the community cards. This difference increased to 0.634 in the river, when all the community cards have been revealed. As the uncertainty in the community cards is resolved, the only unknown factor is the opponent's hands. Our agent is better able to differentiate the expected win rates of different hands.

3 Modelling Player Behaviour

why do we need to model player behaviour if cfr approximates a gto solution in the long term fro all strategies? have to use all strategies to train first right ..

3.1 Tightness and Aggressiveness

A simple way to characterise player strategies is to consider betting behaviour when the player has weak hands and strong hands.

When a player receives a weak hand, they could continue playing or choose to fold. This behaviour can be defined on a tightness-looseness scale. A tight player only plays a small percentage of their hands, while a loose player would choose to take risks and make bets based on the potential of the hand. Loose play is called bluffing, and deceives the opponent into over-estimating the agent's hand strength and folding.

Theoretically, a tight play aims to reduce losses in the case of weak hands. However, a very tight play would also mean that the player keeps folding and the chances to observe the opponent's behaviour is diminished.

On the contrary, when a player receives a strong hand, they could call or raise their bets. This behaviour can be defined on a passiveness-aggressiveness scale. A passive player keeps their bets low and stable, often calling. On the other hand, an aggressive player actively makes raises to increase the game stakes. Passive play, or slow-playing, is another form of deceit which leads opponents to under-estimate the hand strength, thereby continuing to place bets and raise the pot amount.

An aggressive play style hopes to maximise the winnings when the hands are strong. However, an over-aggressive play will also encourage opponents to fold, reducing the overall winnings.

Heuristics for Tightness and Aggressiveness

The interaction with the opponent comes from the betting actions and the showdown. However, not every game ends in a showdown so using card information from the showdown may not be as effective. Hence, our heuristics for player tightness and aggressiveness is derived from the betting actions of folding and raising.

Our agent incorporates these heuristics into its modelling of the opponent by saving the opponent's action history each round. The betting sequence is a meaningful heuristic as it encapsulates when players make certain decisions (e.g. call then raise vs raise then call), while being short enough to track. Given the limitations of the game engine, the maximum betting sequence would be of length 16 (IS THIS CORRECT).

4 Counterfactual Regret Minimisation (CFR) Agent

Our agent is primarily trained using the Counterfactual Regret Minimisation (CFR) Algorithm, which is an extended version of Regret Matching. Regret Matching is a modern technique that Zinkevich et al from the University of Alberta developed, theoretically achieveing a Nash Equilibrium play. We discuss the relevance of our modelling of the game state (Section 2) and player behaviour (Section 3) in the implementation of our CFR agent.

4.1 Regret Matching

Regret is a concept of reward relative to an action, and can be positive or negative. It measures how much the agent retrospectively wants to take the action, given the past observations. Regret is then proportional to the loss observed, when the agent did not play the action. Effectively, the agent favours actions that score the highest in the regret heuristic. Every action x has a regret value that can be formalised:

$$Regret(x) = ActionUtility(x) - CurrentUtility()$$

The regret of an action is how much more the agent expects to win by always playing the action (ActionUtility) vis-a-vis the amount the agent expects to win with its current strategy (CurrentUtility). An action with positive regret can increase the agent's utility.

The strategy changes based on the updated regrets of the actions. The regret-based agent is engages in reinforcement learning to update its beliefs about each action's regret. Later on, we discuss the convergence of this iterative self-play towards a Nash Equilibrium.

The agent plays the actions with a normalised probability proportional to their positive regrets. Our agent will only take actions that have a positive regret. Actions with negative regret are played with probability 0 as they theoretical decrease the agent's utility. Our agent generates a 3-tuple for the optimal probability of playing each action (fold, call, raise). For example, one of the tuples generated could be $(0.2, 0.3, 0.5)$.

CFR - Extending Regret Matching

Unfortunately, poker has a large game tree and with chance nodes in the generation of community cards, making it problematic to accurately define the utility at each terminal node. The CFR technique then extends Regret Matching to games where multiple actions are sequentially taken, and there are chance nodes between these actions.

The agent is counterfactual as it takes into account the chance of each decision being made, and the expected utility from that node. The game tree is further complicated because each decision node continues branching.

Clearly, this is where abstraction is key in clustering nodes together, via the bucketing strategies we discussed earlier. At the start of each street, the branching factor at each chance node is then limited to the number of buckets defined.

Through iterative self-play, the agent updates the regret values over the entire tree.

4.2 Randomising Strategy

The exploitability of opponent behaviour is symmetric, so our agent has to acknowledge that the opponent can similarly extract patterns from our play style. It would then be beneficial to either mask our agent's biases or change our play style during the game.

Consider the tuple (0.2, 0.3, 0.5). We note here that the probability of raising is the most significant, so we can deduce that our player hand is strong. The actual magnitude of the probabilities is determined by our player aggressiveness and tightness, and the evaluated hand strength.

Purification

Some researchers employ purification techniques to overcome abstraction coarseness. These poker agents prefer the higher-probability actions and ignore actions that are unlikely to be played. In particular, Ganzfried and Sandholm found full purification to be the most effective. The full purification technique let the agent play the most-probable action with probability 1.

However, we argue against purification because noise in our player behaviour can disrupt the opponent's attempt in identifying patterns in our play style. Yakovenko et al's poker agent player against human players using a fixed play style, and after 100/ 500 hands, the human player was able to recognise mistakes made by the agent and boost their win rate. In particular, the noise supports our deceit techniques of bluffing and slow-playing.

Changing Play Styles

Besides using noise to generate randomness, we can also change our play style to obfuscate our opponents. Given the same hand as above, our agent may instead generate a 3-tuple like (0, 0.3, 0.7), displaying a significantly higher aggressiveness. We can draw an analogy to local-hill search, where continually exploiting the same strategy may just be leading us to local maxima. The sudden exploration could be less optimal, but may also lead us to an improved maxima.

However, this method is incompatible with our CFR agent. The trained regret values would have converged after numerous training cycles, and are guaranteed to be unexploitable in the long term for all strategies. Using a different strategy exposes a vulnerability to opponents, making the CFR exploitable by certain strategies.

4.3 Nash Equlibrium

In *The Mathematics of Poker*, Chen and Ankenman discuss "game-theory optimal" (GTO) solutions in two and three-player variants. They assert that in poker tournaments with more than three players, there could be implicit collusion between players and hence there is no unexploitable strategy. However, when considering two-player games, then a Nash Equilibrium does exist. Zinkevich et al also assert the convergence of CFR agents towards a Nash Equilibrium.

In particular, we expect the absolute regret for each action to be minimised over time. Intuitively, the minimisation occurs because regret is undesirable - the agent will play actions they regretted not playing, thereby increasing their expected reward (CurrentUtility). The added benefit of playing the action is then reduced, in turn minimising the regret.

$Regret(x)$ is inversely proportional to $CurrentUtility$. The minimisation of regret is also the maximisation of $CurrentUtility$. Eventually, the CFR agent converges to an equilibrium where its utility is fairly constant against all agents.

In training our CFR agent, we used xx cycles. We find this to be a sufficiently large number of training cycles because..... Training was slow...

Unexploitability of CFR Agent

The basis of CFR agents is that they are unexploitable in the long term, regardless of the strategy played by the opponent. In the training stage, the utility is maximised over all betting sequences, so the CFR agent has already taken into consideration different play styles.

While an agent following an optimal strategy may not win the most amount of money from their opponents, they cannot be exploited in the long run.

5 Training with a Deep-Q Network (DQN)

Yakovenko et al posited that a trained model is always more competitive than the model generating the data, since the model was trained to optimise relative to the data. Their team was developing a generic poker agent trained on a Deep-Q Network to play different forms of poker. We used the same technique, though our agent is only expected to play Limit Poker.

A model-based approach requires the agent to learn game states. Data-based approaches are useful because they do not require domain-specific modelling of games. Here, we map actions to values instead. For every action, we can calculate its expected reward, which is the Q-value. An action-based approach parallels the CFR paradigm, generating the regret or relative probability of playing each action. At each layer of the game tree, we have to calculate the Q-value of every action. Our agent learns these values using a Deep Q-Network.

Deep-Q Algorithm

Reinforcement learning introduces high flux. As our agent learns more about actions, the Q-values of the actions also change, making it difficult for the values to converge.

Using a DQN, we introduce a more stable training environment for our agent. The DQN stabilises the training environment in two ways:

- Use of two neural networks. The primary training network is constantly updated in every training iteration. DQN introduces an additional target network which occasionally synchronises with the training network. Since the Q-values in the target network are only updated occasionally, they are stable enough to be used for updating the training network.
- Experience Replay. A large database of training data is stored in a replay buffer. Training samples are randomly drawn from this database, mimicking supervised learning with more stable training datasets.

By stabilising the training, our Q-values can converge much faster than a CFR training. We use the DQN to observe the gameplay of our lightly-trained CFR agent

5.1 Reinforcement Learning Process

Our DQN observes the gameplay between two agents, then learns from the gameplay to update the Q-value of taking each action. We first observe RandomPlayer and HonestPlayer against each other, before observing RandomPlayer against our CFR agent.

Playing against Simple Heuristics Players

A basic agent should win against the RandomPlayer and HonestPlayer, which are provided in the engine. Our agent observed 5000 games of 1000 rounds each, which took about 1 day to complete.

Training against RandomPlayer teaches our agent to play rational, statistically-optimising actions. Our agent will learn to make bets on strong hands and fold on weaker ones. HonestPlayer is the rational version of RandomPlayer, keeping strong hands and folding weaker ones. By training against HonestPlayer, our agent is able to recognise that the opponents are also rational and will pick the best action for themselves at each decison node.

We verified Yakovenko's claim, and it is true that our agent is able to win against RandomPlayer (19980-0) and HonestPlayer (20000-0) after 1 game of 500 rounds. More tests can be done to determine the statistical significance of this result, though we omit this section because our goal is not to win against RandomPlayer and HonestPlayer.

Playing against CFR Agent

We then use the intermediate DQN agent to observe the gameplay between RandomPlayer and our CFR agent. While we expect more optimised results by observing HonestPlayer against our CFR agent, this training will take too long. HonestPlayer runs a Monte-Carlo simulation to determine the expected win rate at each decision node, whereas RandomPlayer does a simple random roll.

– explain why we train with CFR agent

6 Discussion

Theoretically, the CFR agent plays a Nash Equilibrium strategy that is unexploitable. What does it really mean to double-train our CFR agent using Deep-Q Learning?

Does the new DQN agent also approximate to a nash equilibrium?

Secondly, we consider how a Nash Equilibrium in a poker game would manifest in the results. Is the agent always winning?? Is the agent strategy static?? What does being unexploitable mean?

7 Conclusion

In designing our agent, we first looked at theoretical considerations of poker play. The concepts were then consolidated into a CFR agent. Our final agent is a product of Deep-Q training against the CFR agent.

References

A Appendix

B Percentile Bucketing Results

Street	b1-b2	b2-b3	b3-b4	b4-b5
Pre-flop	0.406	0.482	0.534	0.588
Flop	0.308	0.43	0.54	0.684
Turn	0.262	0.412	0.562	0.736
River	0.198	0.416	0.626	0.832