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 creating 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 Deep Blue's win over Garry Kasparov in 1996. The capabilities of AI extend into games with imperfect information, and AI agents have beaten the top human experts.

An interesting example would be Texas Hold'em Poker, a complex game including elements of bluff from both players. More interestingly, it is not enough to merely win the opponent, but also to 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 prime functionality was for the agent to be rational and able to recognise statistically optimal hands to invest in. Given the limited time for our poker agent to decide on the next action, we employed various optimisation strategies such as using abstraction to reduce the game complexity. In particular, we focus on how our agent abstracts poker hands.

Upon recognising strong and weak hands, our agent has to be able to make the right decision (whether to fold, call or bet). In order to give our agent the ability to make such decisions without being predictable, we programmed our agent with Counterfactual Regret Minimisation (CFR). CFR allows our agent to quickly find Nash Equilibria and make decisions based on a probability distribution.

However, calculation alone does not make our agent a winner. We trained our agent to have the ability to model and subsequently predict how an opponent is behaving. This enhancement allows our agent to exploit imperfect information and maximise our gains in a stochastic game.

1. Why did you choose this implementation (i.e. why should the reader be interested?)
2. What is the result that you achieved? (i.e. why should the reader believe you?)

2 Modelling Game State

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

2.1 Hand Strength

The primary heuristic in a rational poker agent would be the hand strength, because that is how poker's end state is eventually evaluated. The player with the higher hand strength clears the pot. There is extensive research on evaluators for hand strength, such as CactusKev, TwoPlusTwo and 7-card.

These evaluators aim to classify cards into groups of similar play styles, which are called buckets. Since hand strengths (HS) 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.

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 whether the two hole cards 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 stage from $\binom{52}{2} = 1326$ to $13 \times 13 = 169$ states. Each state is identified using the value of the cards, and a boolean indicator if they are the suited.

Percentile Bucketing

However, card isomorphism does not simplify our the game complexity enough. We employ a second layer of abstraction, card bucketing, to group cards of similar strengths together. Two sets of hole cards like K♡ Q♢ and Q♢ J♡ can be handled with the same strategy.

Ganzfried and Sandholm found that using too many buckets meant that the training algorithm could not converge sufficiently fast, eventually setting on a 8-12-4-4 bucket size. We simplified their model and chose to use 5 buckets in all streets for our agent.

Our heuristic for bucketing is the expected win rate of a hand. We used a Monte-Carlo simulation to generate these win rates. In each street (pre-flop, flop, turn, river), we ran a Monte-Carlo Simulation. 1000 different hands were generated each time, with the hands consisting of 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 hand.

We used a percentile-win-rate clustering method, so the first bucket comprises the 20% of cards with the lowest win rates. Effectively, each bucket is identified by a minimum and maximum expected win rate.

can probably put a table here

In bucketing, hands can move into different buckets depending on the community cards that have been shown. Johansen suggests using the bucketing sequence in designing the agent, using information on how the hand has moved between buckets in the game. However, we chose to consider each street of the game as being independent and did not incorporate this into our agent.

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?

The current research into poker play suggests that every strategy can be exploited? (what about nash equilibrium LOL)

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 chances to observe the opponent's behaviour is diminished.

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. On the other hand, an aggressive player actively makes raises to increase the game stakes. Passive play, or trapping, 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.

An opponent's strategy at one point in time can be represented as a 2-tuple of (tightness, aggressiveness). The tuple represents an instantaneous strategy as the opponent could change over time to adapt their play to our agent.

3.2 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.

The opponent tightness can be approximated using the folding rate over multiple games. The drawback to using the folding-rate heuristic is that a large number of games is needed to make a prediction. By the time these observations are made, the opponent may have already adopted a different play style.

The opponent aggressiveness can be approximated using the raising rate. We make the assumption that when the opponent has a weak hand, they will only call or fold, and raising is only the result of relatively strong hands.

Our agent incorporates these heuristics into its modelling of the opponent by saving the opponent's action history each round. We extract these features and feed them into our CFR agent.

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.

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.

Given the current agent strategy s and the current opponent strategy s', every action x has a regret value that can be formalised:

$$ActionUtility(x) = E(\{x\}, s')$$
$$CurrentUtility() = E(s, s')$$
$$Regret(x) = ActionUtility(x) - CurrentUtility()$$

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). The expected value of both winnings are calculated relative to the opponent's current strategy.

The equation gives us two important insights into Regret Matching.

1. s 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.
2. The regret values of the actions are always relative to a certain opponent strategy. Regret Matching does not consider the case where the opponent changes their strategies over time.

The agent plays the actions with a normalised probability proportional to their positive regret. We work with a discrete probability distribution that can be represented as a 3-tuple

for the probability of playing each action (fold, call, raise). For example, one of the tuples generated could be (0.2, 0.3, 0.5).

4.2 CFR - Extending Regret Matching

In the Regret Matching equation, it was not specified how the expected value of winning was calculated. Poker has a large game tree and chance nodes which generate the community cards, making it problematic to accurately define the utility at each terminal node. CFR then extends Regret Matching in sequential games with chance nodes.

importance of bucketing in this part.

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.

Over time, we expect the absolute regret for each action to be minimised. Suppose for a contradiction that this is not the case, then there exist an action x' such that $Regret(x')$ is not the minimum. Then, there exists a non-zero probability of playing x'. In that case, playing the current strategy with action x' results in a higher expected gain. As $CurrentUtility()$ decreases, $Regret(x')$ decreases.

For all actions x, $Regret(x)$ continually decreases. However, we have yet to prove the converge of $Regret(x)$ to a minimum value.

spne shown to approximate equilibrium

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. Nevertheless, GTO agents are vulnerable to agents which often change their strategy mid-game.

4.4 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 bias 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.

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 trapping.

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 local maxima. Some researchers use coach-based agents, and this exploration is analogous to fielding a player the opponent has not met before.

** DOES CFR CONVERGE TO A PARTICULAR AGGRESSIVENESS AND TIGHTNESS VALUE? LITERATURE SUGGESTS SO.

5 Training with a Deep-Q Network

While we modelled our agent using knowledge-based approaches, we also recognise the importance of using a data-based approach to train our agent. 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.

Yakovenko's team was developing a generic poker agent trained on a Deep-Q Network to play different forms of poker. We used the same technique of a reinforcement learning algorithm, though our agent is only expected to play Limit Poker.

Deep-Q Algorithm

Q-learning aims to maximise the reward (amount of money at the end of the game), over any and all

this is too complicated let me read first

5.1 Initial Training with Simple Heuristic Players

A basic agent should win against the RandomPlayer and HonestPlayer, which are provided in the engine.

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. Given the training against RandomPlayer, we extend the training against HonestPlayer. Our intermediate player is now a statistically-optimising player that capitalises on hand strengths.

– explain why we train against these 2 players

5.2 Reinforcement Learning with CFR Agent

– explain why we train with CFR agent

6 Discussion

Our poker project takes poker research in two novel directions.

Firstly, we double-trained our CFR agent using Deep-Q Learning.

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.

8 Acknowledgments

A LaTeX and Word Style Files

References