Creating a Texas Heads Up No Limit Poker Bot

Jonathan Lin
California Institute of Technology
jonathan@caltech.edu
Note: This project is still a work in progress.

1 Introduction

This paper serves as a journal for my experiences in creating a multithread version of the existing Monte Carlo Counterfactual Regret Minimization (MCCFR) algorithm to create a strategy that approximates Nash Equilibrium for Texas Heads Up No Limit (HUNL) Poker. My innovation improves MCCFR's performance and efficiency. It is meant to be read casually, and please reach out of there is any part of the paper that I could better clarify. Hopefully you can learn something, whether that's about game theory, reinforcement learning, incomplete information games, etc, that you can use in your own work. It is expected that the reader is familiar with Texas No Limit Holdem and have a high level understanding of reinforcement learning and deep learning.

2 Basic Game Theory

Game theory is the study of decision making in social situations involving competing players. Game theory is used in business, finance, politics, economics, psychology, and many more situations[1]. There is a type of game called zero-sum games. A player's benefit in a zero-sum game is exactly the other person's loss. So if all the gains and losses are added together at the end of a zero-sum game, the sum would be zero. Nash equilibrium, in the context of this paper, will be defined as a set of game theory optimal (GTO) strategies[2]. This Nash equilibrium will be where neither player can improve their expected payout by changing their strategy. Nash equilibrium and GTO will be synonymous in this paper.

A strategy is a probability distribution of actions in a given situation. All GTO strategies cannot be altered to increase expected payout, given that the other player is also playing GTO. Poker, without a rake, is a zero-sum game because any chips a player gains is a direct result of other players' loss of chips. In heads up poker, virtually all poker bots attempt to reach Nash equilibrium. Lastly, poker is known as an incomplete information game. This means that the different players have access to different information, namely their cards. So each player has to make the best decision in a given situation without knowing all the information - somewhat similar to decision making in life. You have to use information, such as past betting history, general priors, psychology, and other tells to make an optimal decision.

3 Counterfactual Regret Minimization

My bot, and most of today's best performing poker bots, use counterfactual regert minimization (CFR) to optimize their strategy. Essentially, CFR works by the model playing against itself and improving itself based on the outcome.

I recommend that you read the original CFR paper, first published by the University of Alberta in 2009, for more formal notation, but I will include some important definitions here[3]. These definitions will be in the context of poker, but generalizing these concepts should not be too dificult. History contains everything that defines a given situation in poker: the hands, the betting street, the action history, and the cards on the board. An infoset is the part of history only accessibly to a given player and is unique for each player. The main difference between a history and an infoset is that the infoset does not contain the cards of the other player. The utility of an action is how much money that action would have made, and the utility is solely measured by how many chips you gained or lost. Regret is the difference between the utility of a certain action and the action that was chosen. Lastly, the strategy is the probability distribution of the events that a player would do in a given situation. For example, the strategy of the small blind's first move after the flop, given that they hit top pair, could be:

Action	Probability
Fold	0
Check	0.2
Raise 1/3x pot	0.2
Raise 1/2x pot	0.25
Raise 2/3x pot	0.15
Raise 1x pot	0.1
Raise 1.5x pot	0.1

Vanilla CFR plays many games against itself and improves based on the results. Counterfactual regret is the regret weighed by the probability of the opponent reaching that state. This makes it so that events that may have a large impact on behavior, but are extremely rare, do not have a large effect on the strategy. For example, if the opponent puts you all in, you have ace high, there was a lot of preflop action, there are a lot of high value cards on the board, the board is extremely connected, and the board is mostly monotone (so there is flush potential), you may win a lot of money by calling with ace high. But the probability of you being ahead is so low that the Nash equilibrium strategy would put a low percentage on calling in this situation.

For each training iteration in vanilla CFR, the algorithm traverses the entire game tree. A game tree is a tree representation of all the possible game states. This is extremely inefficient and not practical in large scale games because it takes too long for the algorithm to converge to Nash equilibrium. So we utilize MCCFR. MCCFR essentially utilizes random sampling in the learning part of the agent. Although more iterations are required to reach Nash, it takes a lot less time per iteration. So instead of multiplying each regret by the probability of it happening, we directly use the fact that random sampling represents the probability that the other player will reach the current state to put more weight towards events that happen more often and vice versa for less common events. Similar to vanilla CFR, MCCFR improves its strategy by playing against itself.

4 Bucketing

A concept called "bucketing" is required for large games because it is not possible to reach of the unique infosets. In fact, there are more infosets available in poker than atoms in the universe. Therefore, bucketing decreasing the number of infosets by many orders of magnitude, making approaching Nash faster. I'm also training on my laptop (2021 Macbook Pro with M1 Max), so efficiency is crucial.

Unfortunately, I could not find any existing effective bucketing algorithms on the internet; therefore, I had to create my own. It is lossless for preflop and lossy for postflop. But the way different hands in the same bucket should be played are approximately the same. The preflop bucketing algorithm is simply <the rank of card #1><the rank of card #2><suited or not suited>. Figure 1 displays my bucketing algorithm for postflop situations. The postflop bucketing algorithm takes into account the board texture (if the board is double paired, number of suited cards, etc), the made hand, and potential draws. Board texture describes different arrangements of the board. For example, a "monotone" board texture means a board of many cards of the same suit. A made hand is the hand you currently have. For example, pairs, trips, and flushes are made hands. Draws are hands that are incomplete and requires additional cards to become valuable. For example, if your hand and the board creates 2345, then it is currently worthless (high-card). But any ace or six that comes out will turn your hand into a straight. My bucketing algorithm only considers flush and straight draws because they are the main types of draws people consider.

5 Multithreading

I implemented multithreading in the MCCFR algorithm. Multithreading is where computer can (essentially) do multiple things at the same time. I created a thread lock for every pair of infoset and matching regret sum, strategy, and strategy sum so different threads can alter the data of different infosets simultaneously. I altered the original MCCFR program to create a new thread each time the external_cfr function is called if there are more than 3 possible actions and it is preflop or flop. The specific hyperparameter of the minimum number of actions and rounds left required to create a thread was determined by a hyperparameter sweep. You cannot create a thread for every possible recursive call because it is expensive to create a thread and the diminishing positive gain from each additional thread. If the bot is on the flop round, then there are not many regret calculations left. So sacrificing computing resources to create the thread only to barely do any work is wasteful. Also, the locks are the main bottle neck of the multithreading algorithm anyway. If you are unfamiliar with multithreading, imagine that you are doing

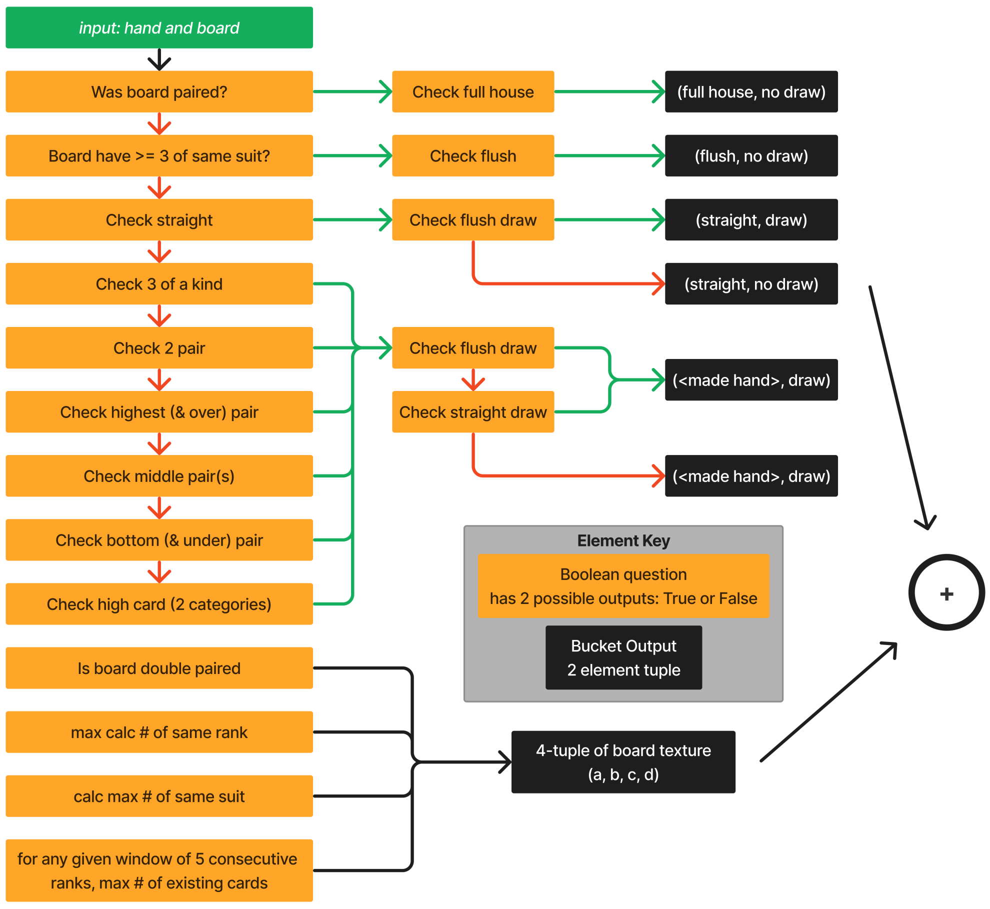

Figure 1: Postflop Bucketing Algorithm.

a 100 question test, and you can have n amount of students that are randomly assigned questions and can work simultaneously. But each question can only be worked on by a single student, and each student costs the school x dollars. In the beginning, each additional student is worth the x dollars because the speed of finishing the test significantly increases. But as you approach 100 students, the marginal benefit does not outweigh the marginal cost. And when you cross 100 students, many students at a time will be doing nothing.

6 Initial Weights

Typically in CFR, initial weights are uniform. So the probability of doing any given action is equal to the probability of doing any other action. Minimizing training time is crucial, so the starting weights for an unseen infoset are not uniform.

First, I simulated millions of games of poker. With each game, I kept track of the win, tie, and loss rates of each bucket. Then, I created a preassigned probability distribution for actions based on ranges of win rates. For example, if the win rate of a given infoset is below 5%, then the starting probability of folding/checking is extremely high, and the starting chance of going all-in is extremely low. Theoretically, this method should decrease the required training time for the algorithm to reach Nash, because the initial weights should be closer to GTO than strictly uniform strategies. I used existing knowledge I already knew about poker, called "domain knowledge" in general terms, to create a process for choosing the initial weights.

7 Exploitability

Exploitative poker is a playing style that is dependent on your opponent. For example, if you know that your opponent plays extremely tight and only plays extremely good hands, then you know to fold anytime that person raises. Being unexploitable means that even if the opponent knew your exact strategy, there is nothing they can do to win more chips from you. A strategy reaches Nash equillibrium is unexploitable.

I measure exploitability by playing the bot against different versions of itself. The strategies for these bot variations are created by taking the strategy profile of the original bot and modifying it appropriately for the given play style. If the bot is able to consistently be profitable against all the following play styles, the bot is unlikely to be exploitable. Obviously, exploitability is more detailed and nuanced than described, and there are more complex strategies than the ones I listed. But, my evaluation of exploitability is adequate.

- Aggressive: The bot folds less and raises larger amounts more often.
- Tight: The bot folds more and raises larger amounts less often.
- Random: All strategies in the strategy profile are uniform. All possible actions in a given situation are equally likely to be selected.
- Calling Machine: The bot is more likely to call than any other action.
- Tight Aggressive: The bot plays tight if the win rate of the current situation is less than a certain value and plays aggressive if the win rate of the current situation is more than a certain value. This is essentially an unbalanced player - their actions perfectly reflect their hand value.

8 Modifications to Game

I noticed that after the first set of training the bot, the bot played too aggressively. It folded less than it should have, and the bot would overbet more frequently than it should have. So, you could consistently beat the bot if you simply played tight. This problem can be seen in figure 2. The bot is consistently profitable against all play styles except for a tight version of itself. I determined the root of the problem to be that the pot would balloon to unrealistic sizes in some cases. For example, you could win around 5000 BB if there was a lot of action preflop and many reraises during postflop betting streets. Generally in poker, you play with 100-300 BBs. So if the bot typically won 50 BB per hand, then it would only have to win that hand that it played over-aggressively in $\frac{1}{101} \approx 1\%$ of the time. But given the previous constraint of pot sizes, this method of thinking is not reasonable.

I am modifying the game simulation to check if the pot size is above 200 BB after each betting street. If it is, then all betting is stopped, representing “all-in”. All future betting streets are void (both players check through), and the rest of the cards are dealt to the board. If the pot has not reached 200 BBs yet, then the simulation allows you to continue. This constraint should make the bot less aggressive.

9 Unexplored Situations

Even though the bot could’ve trained for a while, it could still come across extremely rare situations where it has not experienced. For example, it is possible for the bot, when playing someone, to come across a situation where there are quads (4-of-a-kind) on the board. Though unlikely, the bot still has to know what to do. These are options I am exploring:

- embed all current infosets into vectors and train a decision tree on it. Then embed any unknown situations any feed it through the decision tree
- Embed all current infosets into vectors. Then embed any unknown situation and have it have the same distribution as the nearest known infoset, in terms of euclidian distance.

10 Deep Learning

A problem with utilizing CFR for such a large scale game, such as HUNL Poker, is that the average strategy takes too long to converge to equilibrium. A possible solution to this is to utilize deep learning to output the strategy for a given infoset because deep learning has been proven to be effective in generalizing complex patterns. I plan to formalize the experimental results in the near future with proper figures,

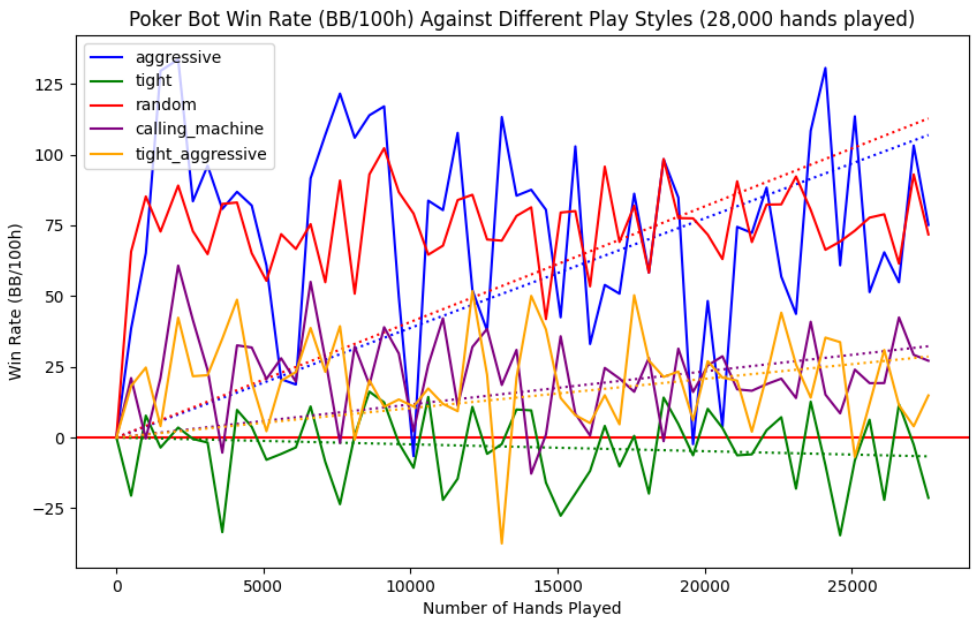

Figure 2: Exploitability of bot v1.

but deep learning has produced unsatisfactory results so far. Deep learning models tend to produce similar strategy outputs regardless of the bucket.

11 Future Plans

There is still a lot of work to be done on this project. Here is a list of things I plan to do in the near future regarding this project:

- Train multiple models and average the result. So for a given infoset, I run the infoset through x different models and then sample randomly from the outputs. I want to try this because the performance of the poker bot has extremely high variance. For example, sometimes the bot to be exploitable from tight players, and sometimes the bot will be exploitable from loose players. And the bot/weights were both trained from scratch in both situations. This method of averaging results obtained from independently trained models is known as *ensembling* in machine learning.
- Continue playing with deep learning and try to implement stuff I learned in CS155 (a theoretical machine learning course at Caltech) and see if performance improves.
- Create a poker bot for multi-way tables of Texas No-Limit Holdem Poker, though this will be far more complex than heads up.

12 Additional Resources

Here are some resources I found extremely helpful in teaching myself the concepts behind creating this bot.

References

[1] E. Picardo, "How game theory strategy improves decision-making," Dec 2023.

[2] J. Nash, "Non-cooperative games," *Annals of Mathematics*, vol. 54, no. 2, pp. 286–295, 1951.

[3] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," in *Advances in Neural Information Processing Systems* (J. Platt, D. Koller, Y. Singer, and S. Roweis, eds.), vol. 20, Curran Associates, Inc., 2007.