Solving Heads Up Texas Hold’em Style Poker Games

Author: Benjamin Tong

Advisor: Dana Angluin

CS 490 Final Report

1: Introduction

No limit Hold’em is the most common variant of the family of card games popularly known as poker, played across the world recreationally and even online. The game involves incomplete information, with each player knowing their own cards without knowing their opponents’, and each player obtains more communal information as the game progresses at the cost of potentially more chips, when a winner is finally revealed at the conclusion of each round by comparing the player’s cards against one another. Although a single hand of no limit Hold’em has an incredible amount of variance, there are professional players who are able to make a living by playing for long term expected value; even when a professional loses a hand, they can still say they made the correct play if the play had positive expected value.

Due to the history and popularity of no limit Hold’em, there have been many attempts to solve the game. Most recently, the Libratus Poker AI beat four professional players in heads up (one versus one) poker at a rate of 14.7 big blinds per 100 hands, which is a rate most professionals would not be able to make themselves against weaker players. The results were over 120,000 hands and the results were deemed statistically significant with p=0.5%. In this project, we attempt to create a bot that can play games similar to heads up no limit Hold’em and hold its own using similar reinforcement learning techniques.

We use a technique called counterfactual regret minimization, a form of reinforcement learning. Regret minimization begins with a random strategy, and on each iteration it calculates
the regret of each action it makes based on the each action’s utility. Then, the new probability distribution of actions at each step is calculated using these regret values. This process is iterated as many times as necessary until the average strategy converges towards a Nash Equilibrium. This process guarantees such a convergence, but for games with as many possibilities as no limit Texas Hold’em, the algorithm requires too many iterations to converge in a reasonable amount of time. Techniques such as partial pruning can reduce the time required to compute the optimal strategy drastically.

In this paper, we implement the regret matching algorithm for counterfactual regret minimization. We make several simplifying assumptions to reduce the complexity of the game and implement pruning techniques such as partial pruning to solve the game in a much shorter amount of time. The final algorithm can be used to solve a wide variety of games similar to Texas Hold’em, and we present a few examples of the calculated optimal strategies.

2: Poker Background

In heads-up, or two player, Texas Hold’em, each player receives two private cards from a deck of 52 cards. There are 13 ranks of cards in the deck and each rank has 4 different suits. The game begins with 0 community, or public, cards. This is called the preflop. After the first street, or round, of betting, 3 community cards are revealed simultaneously. After the second and third street of betting, another community card is revealed, for a total of 5 total community cards. There is a final street of betting after the fifth card is revealed. Then, the player with the best five card hand takes the pot.

On each player’s turn, there are four different possible actions at each step: check, call, raise, and fold. A player can only check if each player has the same amount of chips put into the
pot, and that player does not add any chips. A player can call only if the opponent has just raised, and they add the same amount of chips into the pot as the raise amount by their opponent. A player can raise as long as they still have the chips to do so, and they choose some number of chips to add into the pot, in addition to the amount that the previous player raised by if the previous player raised. The option to choose any amount to raise by is what makes the game termed no limit, with the restriction A player can fold when their opponent has just raised, thereby putting no additional chips into the pot and surrendering the game to their opponent. A street of betting ends preflop when a player has called a raise or when the dealer calls initially and the big blind checks. A street of betting ends on all other streets when a player has called a raise or when both players check.

3: Counterfactual Regret Minimization

We consider counterfactual regret minimization in the context of two player Texas Hold’em style games. Define a history as a sequence of actions by the players and by chance. We can define a tree of histories, where the children of a history at a node on a player’s turn are the histories of taking each possible action at the current node, and the children of a history at the end of a street of betting are the possible community cards that are revealed for the next street. A terminal history is a history in which no more actions by players or chance are possible. This is equivalent to the end of the last street of betting. For each terminal history, we define a utility function for each player that is the payoff of that history, or the expected value of that history.

An information set for a player is a set of histories that are indistinguishable from the perspective of that player. For example, the first player may hold two aces or two kings, and they may have just raised into their opponent preflop. These two cases represent two different
histories, but from the perspective of the opponent, they represent the same information set, as the first player’s cards are unknown to the second.

A strategy in an information set for a player is a probability vector over the possible actions of that player. All histories in an information set are indistinguishable from the perspective of the current player, so the strategies are identical among them. A strategy for a player defined over the entire game is a set of strategies over all information sets in which it is that player’s turn to act.

The reach probability under a strategy from one history to another is the probability of reaching the second history from the first if all players play according to the same strategy. The counterfactual reach probability from the perspective of a player is the same, but we consider the other player to always choose actions leading to the end history.

A Nash Equilibrium is a strategy for each player such that each player plays a best response strategy, which is a strategy that maximizes the player’s utility regardless of the opponent strategy. A Nash Equilibrium in Texas Hold’em style games may have different overall utilities for the two players, but since the positions of the players are flipped after each game, the overall utility is still 0 under a Nash Equilibrium strategy for the players.

The counterfactual value of an information set is the expected utility of an information set given that the opponent tries to reach it. This is computed by first summing over the probabilities of reaching each terminal history from the current information set times the expected utilities of the terminal sets. This is then multiplied by the counterfactual reach probability of reaching the information set.

The instantaneous regret of an action at an information set is defined as the difference between the counterfactual value of the information set after taking a specific action and the
counterfactual value of the information set itself on the current iteration of the algorithm. The total regret of an action at an information set is the sum of the instantaneous regrets over all iterations that have been computed so far. The positive regret is the maximum of the total regret and zero.

Regret Matching is an algorithm for counterfactual regret minimization in which at each new iteration, the strategy at each information set is chosen based on the proportion of the positive regret of taking each action over the sum of the positive regrets over all actions. The average strategy is the average of the strategies at each information set over all iterations. It has been proven that under regret matching, as the number of iterations approaches infinity, the average strategy forms an epsilon-equilibrium strategy, which is a strategy in which the utilities of the strategy are within epsilon of a Nash Equilibrium strategy.

4: Implementation Assumptions

We first assume that suits do not matter, so flushes are not important in the calculation of the strength of hands. This leads to a much smaller game space which allows for faster calculations. For example, if there are 3 ranks, then there are 3 possible pairs of the same rank and 3 possible combinations of two cards of different ranks for a total of 6 possible starting hands. On the other hand, with 4 different suits, there are still 3 possible pairs of the same rank, but there are now 18 (3 * 4 C 2) possible combinations of two cards of different ranks and suits and 48 (3 * 4 * 3) possible combinations of two cards of different ranks, for a total of 64 possible starting hands.

We assume that the dealer acts first preflop and acts second on every other street of betting. The dealer must also place 1 chip initially while the big blind must place 2 chips. In
essence, we are trying to mimic the style of a 1-2 betting game, where the small blind is $1 and
the big blind is $2.

We assume that the only raise size available preflop is 3x the last bet. For example, the
first initial raise size for the dealer is 6, since it is 3 times of 2. If the big blind raises again, then
the raise is to 18. The only available raise size on all other streets of betting is two-thirds of the
total amount in the pot. We limit the raise size to only one possible size in order to simplify the
game space. The sizes of the raises were chosen to represent bet sizes that are most commonly
seen in high level no limit Hold’em games. In essence, we are looking at a variant of limit poker
games, except the sizing is what is typical of a no limit poker game, which discourages marginal
hands from calling and promotes a more aggressive playstyle compared with a normal limit
poker game.

5: Implementation Details

The regret matching algorithm is implemented in Python 3. We discuss the Poker.ipynb
file except where noted otherwise.

The Card class defines each card object as a suit and a rank. Although the suit is defined,
the suit is not ultimately used in the algorithm for simplicity, as the number of combinations of
cards increases significantly, as stated in the previous section.

The Deck class is defined as a list of cards. The Draw() function draws a random card
from the deck. The Remove() function removes a specific card from the deck.

The FiveCardHand class is used to compare the strength of two hands. These objects are
initialized in the HoldemEnv class.
The Hand class is just a list of two cards with some information about the suits and numbers they contain.

The HoldemEnv class is the main class for holding public information about the history. Each node in the tree of histories contains a HoldemEnv object that gives information including the betting action that has occurred and the community cards revealed. It also contains functions to advance the history to the next, including the actions the players can make (Check(), Call(), Raise(), and Fold()) and revealing the next cards (NextStreet()). The HandValue() and Compare() functions are not directly used in the implementation of the INodes below, but are called beforehand to create the hand_winrate tables that are used by the INodes. The hand_winrate tables are computed in the Calculate Preflop Table.ipynb file with different parameters. These tables are more clearly defined in FileList.txt.

The INode class is the class representing information sets and is updated each iteration of the regret matching algorithm. Each INode object represents one information set. They are organized in a tree in which the children of an INode in the perspective of one player are the indistinguishable histories that form an information set for the other player. (This holds except for the case of the last INode on each street. Then, the children of that node are the possible cards that will be revealed on the board in preparation for the next street.) Each INode contains a HoldemEnv object, hist, as the main public information on the information set. INodes contain private information regarding the player whose current turn it is in the HoldemEnv. This information is not represented as an actual listing of the cards in their hand though; instead, all possible hands they could be holding are represented in the INode by representing the probabilities of the player’s actions as a 2-D array indexed by the current player’s holding and the current player’s action. In terminal information sets, the utility is stored as an utility_table if
neither player has folded. This table is of dimension X by X, where X is the total number of possible starting hands. The values of the utility_table are calculated beforehand using the hand_winrates table. The function PrintAvg() is used as a visual representation of the average strategy. The related functions CreateGameTree() and its helper function CreateGameTreeHelper() are used to instantiate a tree of INodes. The specifications for the tree are all in the INode class and the hand_winrates table.

The next section consists of functions that implement the regret matching algorithm. ReachProbability() calculates the probability of getting from one start INode to the end INode given the hand holdings of both players. The optional parameter cf can be set to determine whether a counterfactual reach probability is desired instead, in which the player who is not set in the parameter would be assumed to always move towards the end node with probability 1. Note that the first player is 0 and the second player is 1.

CounterfactualValue() computes the counterfactual value of a current hand in an INode. The optional parameter action can be used to specify a computation of the counterfactual value of the current hand in the information set upon the current player taking the specific action. Note that the actions are ordered as 0, 1, 2, and 3 for check, call, raise, and fold.

InstantaneousRegret() computes the instantaneous regret of an action given a player’s hand and the current information set. RegretMatching() then uses these functions to process the regret matching algorithm for some number of iterations. On each iteration, first the regret is computed for each action. The regret of an action at an INode is stored in the child of that INode corresponding to the action. The probabilities are then set according to the regret. This completes an iteration, and the process repeats.
6: Results for a Sample Game

We ran the regret matching algorithm over a simple game in which each player is dealt two cards. The ranks range from 2 to 4 inclusive, for a total of 12 cards in the deck. There is one card revealed after the first round of betting and a second card revealed after the second and final round of betting. On each street of betting, a maximum of 2 raises between the two players is allowed. After the second raise, no further raises are allowed, so the only available actions are call and fold. Besides the raise limit restriction, there is no limit on stack sizes. We ran 300 iterations of regret matching with partial pruning. The full table to action sequences is stored in SampleGame.txt. Note that the number in the pairs (x, y) are 2 less than the card we receive. We present some sample sequences of actions here. Note that some of the action probabilities are close to 1 or close to 0, in which it is likely the action should always be taken or never be taken, but is not 1 or 0 because we have taken an average over an insufficient number of iterations.

Suppose we are the dealer and we are dealt the cards three and four. Then, we should call 30.5% of the time and raise 69.5% of the time. Let’s suppose we raised. (Line 650) Then, suppose our opponent called. Now the first card is revealed. Suppose the card revealed was a two. It is our opponent’s turn to act again. Our opponent checks. (Line 667) We currently have no pair yet, but we should still raise 93.7% of the time here, and we should only check 6.3% of the time. We choose to raise. Now our opponent raises us here. (Line 699) We now are not favored against our opponent even with one card to come. We fold here almost always, 98.3% of the time.

Now suppose we are the big blind and we are dealt a pair of fours. This is the best possible starting hand we could have, as four is the highest rank in this game. Our opponent is the dealer and first to act and they raise. (Line 650) We should call here 17.6% of the time and
raise here 82.3% of the time. (One might wonder why we do not raise 100% of the time here; although the algorithm does not give the reason for this decision, we can intuitively find an explanation. If we were to raise 100% of the time, then when the first card is revealed a 4, we will never have the best hand possible, triple fours, on that board if we only called. Since the algorithm is converging to a Nash Equilibrium strategy, the strategy must be balanced and must be able to respond to any actions in a balanced manner.) Now, our opponent just calls. Suppose the card on the board is a 4. Our opponent raises. (Line 842) As we have the best possible hand regardless of what card comes up next, we definitely need to raise, and the algorithm does indeed tell us to raise essentially always, 99.6% of the time. Our opponent then folds.

Notice that in both of these simulations we only used our private information, the cards in our hand, and our public information, the actions made by us and our opponents and the cards on the board.

7: Partial Pruning

In order to speed up the algorithm, we implemented partial pruning, which skips a portion of the INode tree on each iteration of the regret matching algorithm when that portion of the INode tree is never reached because the counterfactual reach probability with respect to the current player is 0. Implementing partial pruning brought at least a two times speed increase over not using partial pruning, showing that it skips a large portion of the game tree on each iteration. However, more iterations of the algorithm will need to be used, since some subtrees that are skipped over will need to be processed by the algorithm in the future. Although we were unable to quantify exactly how many more iterations were necessary, the speed increase was clearly noticeable.
8: Future Work

The original goal of the project was to work on solving no limit Hold’em. In theory, the regret matching algorithm can be sufficient to find an optimal strategy in no limit Hold’em that cannot be exploited regardless of the opponent strategy. However, in reality, the algorithm is too slow to process the entirety of the no limit Hold’em game. We made several simplifying assumptions, such as limiting the raise size to only one specific size relative to the previous bets, using a much smaller deck, ignoring suits, and having fewer community cards on the board. These assumptions were all made in order to allow the algorithm to complete in a reasonable amount of time. Thus, our primary focus for future work will be on speeding up the algorithm. There are a variety of techniques that can increase performance, and partial pruning was only the first step in the series. Sandholm et al. discuss such methods, including regret-based pruning strategy and dynamic pruning. Regret can also be calculated in a different manner, in which regret itself is considered equal to positive regret. A weighted average strategy would then be necessary. Regret matching is also not the only counterfactual regret minimization algorithm out there; in particular, Hedge is another major algorithm that could potentially converge faster, depending on the learning rate that is set. However, it was avoided due to its more complicated algorithm. In conclusion, there is still a lot of work that can be done to improve performance, thereby increasing the scope of games that counterfactual regret minimization can solve in a reasonable amount of time.


