Solving Discard Poker

Kundan Chintamaneni

February 1, 2017

Abstract

As part of MIT's annual Pokerbots competition, we were tasked with creating an artificial intelligence that plays a type of poker called Discard Holdem. We used counterfactual regret minimization (CFR), a machine learning algorithm, to create our bot's strategy. In this paper, we describe the implementation details of adapting CFR to Discard Holdem.

1 Introduction

Discard Holdem is a variant of the popular card game, Heads-up No-limit Holdem (NLHE). After the flop and after the river, both players have the option of discarding one card from their hand and drawing one card from the deck.

This new mechanic causes traditional NLHE strategies to fail. For example, pocket two's is a strong hand in NHLE. It is a favorite against hands like Ace King, and professional players will almost always raise with it. However, pocket twos is among the worst hands of Discard Holdem. Because players have the option to discard, they are likely to improve to a better hand.

Because of our unfamiliarity with Discard Holdem, we decided to use machine learning to come up with a strategy. One of the best NLHE playing robots, Claudico, was trained using counterfactual regret minimization [1]. This algorithm attempts to find a game theory optimal solution through self-play. In loose terms, the algorithm will play matches against itself and record how effective each decision was. Over the course of many iterations, this algorithm is guaranteed to converge to a Nash Equilibrium. Claudico's

success inspired us to use counterfactual regret minimization to solve Discard Holdem.

2 Challenges

NLHE has too many game states for CFR to be able to solve it directly. The developers of Claudico first made an abstraction of NLHE, which was very similar but had fewer game states. Claudico trained on this abstraction for three months on a supercomputer with 961 cores [1]. Unfortunately, we did not have the time or the resources to do this. In order to create a working bot by our deadline, we had to significantly cut down the number of game states in our abstraction of Discard Holdem. This was done in two major ways: hand abstractions and betting abstractions.

2.1 Hand Abstractions

Many hands in poker share similar characteristics. For example, having a pair of twos on an Ace-King-Queen board is almost the same as having a pair of threes on an Ace-King-Queen board. Both hands would likely be played in the same way. In order to take advantage of this likeness, we classified hand, board pairs into different buckets. We created 169 different buckets for preflop hands, 100 buckets for postflop hands, 100 buckets for postturn hands, and 100 buckets for postriver hands.

There are 169 distinct preflop hands after accounting for suit isomorphisms, so we put one hand in each bucket. To put the postflop hands into buckets, we followed the abstraction set forth by Johanson et al. [2]. First, we partitioned the 169 preflop hands into eight ranges representing different hand textures. For example, one range includes all the high pocket pairs while another range includes suited connecters. Then, for each hand and flop pair, we calculate the equities against the eight different ranges. This gives us a mapping between the hand and flop pairs and points in 8-dimensional Euclidean space. Then, we clustered the points into 100 clusters using the kmeans algorithm. Similar steps were used to put the postturn and postriver hands into buckets.

Another trick to reduce the number of game states was using imperfect recall while training. A more complete description can be found in [2], but in simple terms, the robot does not remember earlier details about the hand.

After the flop flop bucket, the forgets its preflop bucket; after the turn, the robot forgets its postflop bucket and so on. If the robot had perfect recall, this would create $167 * 100 * 100 * 100$ possible hand states. Using imperfect recalls allows us to cut this to $167 + 100 + 100 + 100$ different states.

2.2 Betting Abstractions

The two main problems we have to deal with are bet sizing and the number of bets. In Discard Holdem, there is great freedom in the amount of chips that can be bet each turn. For example, a player can raise anywhere from 4 chips to 200 chips on the first turn. We want our robot to display different behaviors when our opponents raise 4 chips compared to when they raise 200 chips. However, the more bet sizes we include in our abstraction, the more game states we will have and the longer it will take for our robot to train.

We prioritized having a small number of game states and so we decided to use only two bet sizes, a moderate bet of 66% of the pot and a large bet of going all in. Similarly, we used only two raise sizes, a moderate raise of 100% of the pot and a large raise of going all in.

The second issue we had to deal with is the number of bets we should allow in each round. After one player raises, the second player and an option of reraising. The first player then has the option of rereraising and this cycle can continue until both players run out of chips. It is prohibitively expensive in terms of game states to allow for multiple rounds of raising and reraising. To deal with this, we capped the number of actions each player could take to two per street. This number was chosen using our knowledge playing NLHE. Allowing two actions per player preflop allows for 3-betting, and allowing two actions per player in every other street allows for check-raising, which are both essential tools to prevent aggression.

While limiting the number of bets helped with training, it created problems when playing matches. For example, if an opponent 4-bets us, then we have no direct response. Our bots were only trained to deal 3-bets or lower. To fix this, we had to map 4-bets into options our bot could understand: we had to map it either to a call or to an all in. We used the formula presented in [1] to determine whether we should interpret a 4-bet as an all in or a call. Similarly, we used this formula to map small bets to checks and small raises to calls.

3 Discard logic

To determine whether we should discard a card, we count the number of outs we have. This is the number of cards remaining in the deck that will improve our hand if we draw them. If the number of outs we have is greatest when we discard the left card in our hand, then we discard the left card, and so on for the right and and for not discarding. We checked that our discard logic made sense by running equity calculations against a random hand.

During training, we assumed that our opponent would always discard if they can do so and reach a better hand. We did this to cut down the number of game states we had to train over. However, this assumption leads to exploitable play. If a robot always discards when they can improve their hand, then when they don't discard, we know that they have a very strong hand. Using discard information in this way is a fascination topic that we wish we had more time to investigate.

4 Implementation Details

The counterfactual regret minimization algorithm was implemented in c++, following the pseudocode presented in [3]. I used the pbots_calc library [4] in order to perform equity calculations for bucketing, discard logic, and for showdown evaluations. The algorithm was ran on my 2015 MacBook Pro Laptop for six days, with a speed of 100 iterations per second. As the program was not able to reach convergence, we thresholded the output by 10%. That is, if in a certain scenario our bot would take an action 10% or less of the time, we made it so that our bot would never take that action.

5 Conclusion

Applying CFR to Discard Holdem provides unique challenges, particularly when time and resource constraints are applied. We decided to aggressively limit the number of game states at the cost of losing information in order to allow CFR to have more iterations to train. By applying heavy hand abstractions, betting abstractions and using strong discard logic, we hope to emulate the success of Claudico and create a strong poker bot.

References

[1] https://www.cs.cmu.edu/ sganzfri/Claudico15.pdf

[2] http://dl.acm.org/citation.cfm?id=2484920.2484965

[3] http://modelai.gettysburg.edu/2013/cfr/cfr.pdf

[4] https://github.com/mitpokerbots/pbots_calc