STRATEGY REPORT: PINEAPPLE

By Milo Knowles (mknowles@mit.edu) and Arinze Okeke (arinze@mit.edu)

Counterfactual Regret Minimization with Average Strategy Sampling

We trained our bot using a variant of CFR that uses average strategy sampling to simulate player actions. “Vanilla CFR”, the brute-force approach to CFR, visits every single node of the game tree with each iteration, and the training process is very slow, albeit robust, as a result. Despite the massive size of the game-state space in no-limit Hold’em, there are many game states that a player would not encounter in a real game. For example, certain betting sequences would never be chosen by a reasonably good human poker player or a bot with basic knowledge of it’s hand strength. Therefore, it is a waste of computational resources to explore the entire game tree when only a fraction of the game states are likely to be encountered.

The general idea behind CFR with *average strategy sampling* (AS) is to sample player actions stochastically at each action node, based on that action’s weight in the current strategy. This allows us to visit game histories that are more likely to occur with higher frequency. As the player’s strategy improves during CFR training, each exploration of the game tree becomes more efficient; the subtrees of “good” actions are explored more often than “bad” ones.

In a 2015 paper by the Computer Poker Group at the University of Alberta, average strategy sampling is proven to converge much more quickly towards Nash Equilibrium that vanilla CFR and other variants. Because we had so little time to train our bot, average strategy sampling made sense over vanilla CFR. Even if we didn’t have time to explore every node of the game tree, AS would allow us to train the bot on game histories it was more likely to see against a real opponent.

Game Abstraction

In order to make CFR for No-Limit Hold’em with Discards computationally feasible, a very coarse game abstraction was used. In our abstracted game space, there are about 3.3 million possible game histories that the bot can arrive at.

First, hands are “bucketed” based on their expected value. On the preflop and turn, 3 hand buckets are used. This means that any hands with EV between [0, 0.333], [0.334, 0.666] and [0.667, 1.0] appear identical to our bot in the abstracted space. For the flop and river, where the player receives a significant amount of information, four hand buckets are used. Our thought was that slightly better knowledge of the hand strength would be more useful on the flop because this is where the player must make critical decisions about strategy for the rest of the hand. On the river, it is important to have a better sense of hand strength because there are no more chance events left.

In addition, betting actions had to be reduced substantially. Rather than allow our bot to bet any integer amount of chips, which would make the game tree impossibly large to explore, we limited our player to, at most, 3 bet sizes. The bot can bet or raise by half the pot, the full pot, or go all-in. This betting abstraction was taken from a paper by the team at Carnegie Mellon that worked on the poker bot "Tartanian." There, they justify each betting amount:

- **Half pot bets**: "Bets equal to half of the size of the current pot are good value bets as well as good bluffs. When a player has a strong hand, by placing a half-pot bet he is giving the opponent 3:1 pot odds. For example, if a half-pot bet is placed on the river, then the opponent only needs to think that he has a 25% chance of winning in order for a call to be "correct". This makes it a good value bet for the opponent who has a good hand. Half-pot bets also make good bluffs: they only need to work one time in three in order for it to be a profitable play" (Gilpin et. al.)
- **Full pot bets: "**Bets equal to the size of the current pot are useful when a player believes that he is currently "in the lead", and does not wish to give the opponent a chance to draw out to a better hand (via the additional cards dealt later on in the hand). By placing a pot bet, the player is taking away the odds that the opponent would need to rationally call the bet" (Gilpin et. al.)

Finally, betting rounds are limited. Theoretically, players can reraise each other close to 100 times if they only raise by the minimum each time. For obvious reasons, we needed to prevent this from happening during CFR training. During each street of betting, players can go through, at most, 2 rounds of reraising before they are forced to check or call and proceed to the next street. In a real-life game of poker, it is unlikely for players to reraise many times in a row, so we do not think that two betting rounds is a severe limitation to the bot.

Training Results

Although we finished the code necessary to abstract the Hold'em variant, iteratively train the bot using CFR, and map the abstracted version of the game back into the full game, there was not enough time to sufficiently train the bot. We ran CFR for approximately 50 hours, but with slow discard evaluation functions (involving look-ahead to the river), the CFR training reached only ~70,000 game histories, and sampled the game tree about a million times. By our calculations, there are about 3.3 million reachable game histories, even in our highly abstracted version of the game. During testing, we only encountered untrained game histories about 5% of the time, but many histories had relatively uniform action weights. This means that CFR only reached these states a handful of times, which is not sufficient to accumulate meaningful regret.

We competed the CFR trained bot against a bot with simple betting heuristics based on hand strength. Although the CFR bot occasionally won, it's tendency to choose large bets (especially all-in bets) in scenarios where it had not been properly trained made it easily exploitable. We ended up deciding to submit our simpler "backup" bot to the final competition, and continue to train our CFR bot as a future project. Using an AWS instance, faster discard evaluation functions, and tuned training parameters, we hope to demonstrate that our CFR algorithm can produce a viable poker player.

The code for our CFR implementation can be found at:
https://github.com/milokhl/pokerbots2017

Backup Bot Strategy

Our back up strategy was simple evaluation function. It used the 'deuces' library to evaluate hands by ranking them from 1 to 7462. We then converted this to a score in the range of [0,1) by doing 1-rank/7462. We had a minimum score cutoff for a playable hand at a score of 0.4. We initially did this with poker-eval and had a cutoff at 0.7 but because deuces only ranks hands and doesn't include the probability that an opponent will have a better hand than us, we had to lower the cutoff. The file is organized into HelperFunctions, which converted hands to card objects and parsed inputs; Player, which gave responses to the engine; and Strategies, which calculated which move to make.

In Strategies, we had a function corresponding to each bot we made. Our final bot was version 16 so it used Strategy16. Whenever it had the option to discard, it tried to discard each card and found the expected score after doing so to determine which card if any it should discard.

If it was looking for an actual move instead it would use one of the ChooseAction functions. Our final bot used ChooseAction9. This function set a maximum amount to bet preflop because scoring of the hand was grossly inaccurate at this point and we did not want to go all in. Afterward, it calculated the potsize after it made its bet and decided to make a bet that would bring the potsize to score*400, a ratio of the max potsize. For this there was a score cut off for which we would be alright going all in. We used this score cut off to calculate a multiplier. We would check if the potsize was between score*400 and that times the multiplier. This way anything lower than that score would not go all in, but rather fold or call. To save time, rather than betting a bit at a time and raising each time, we just jump straight to the desired point for the pot so the game moves more quickly.

Sources

Gilpin, Andrew, et. al. "A heads-up no-limit Texas Hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs." *Carnegie Mellon Computer Science Department*. 2008. Web. <https://www.cs.cmu.edu/~sandholm/tartanian.AAMAS08.pdf>

Gibson, Richard, et. al. "Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions." *University of Alberta Computer Poker Group.* <*http://webdocs.cs.ualberta.ca/~games/poker/publications/NIPS12.pdf*>