Slumbot NL: Solving Large Games with Counterfactual Regret Minimization Using Sampling and Distributed Processing

Eric Jackson
eric.jackson@gmail.com

Abstract

Slumbot NL is a heads-up no-limit hold'em poker bot built with a distributed disk-based implementation of counterfactual regret minimization (CFR). Our implementation enables us to solve a large abstraction on commodity hardware in a cost-effective fashion. A variant of the Public Chance Sampling (PCS) version of CFR is employed which works particularly well with our architecture.

1 Introduction

Slumbot NL is a poker bot that attempts to play according to an approximate Nash equlibrium. As such, it employs a static strategy; it does not adapt to its opponents nor attempt to exploit opponent errors. No-limit hold'em is much too large to compute an equilibrium for directly (with blinds of 50 and 100 and stacks of 200 big blinds, it has on the order of 10^{164} game states (Johanson 2013)). Following standard practice (dating back at least as far as (Shi and Littman 2000)), we create a simplified form of the game — an abstraction — that preserves essential features of the full game. We find a solution (an approximate equilbrium) to the abstract game and hope that it maps to a strategy that works well in the full game.

2 Algorithm

The algorithm we employ to compute our strategy is a variant of counterfactual regret minimization (CFR) (Zinkevich et al. 2007). CFR is an iterative algorithm which computes a strategy for each player at each iteration. The final strategy is the average of the strategies computed at each iteration. The strategies produced by CFR have been shown to approach a Nash equilibrium (Zinkevich et al. 2007). On each iteration, CFR computes a quantity known as "counterfactual value" for each action which is, roughly speaking, the expected value of that action against the opponent's current strategy. The "regret" for each action is determined based on the counterfactual value, and the probability assigned to each action is proportional to its positive regret.

The version of CFR initially described in (Zinkevich et al. 2007) (sometimes described as "Vanilla" CFR) performs a complete traversal of the game tree on each iteration. Each iteration computes the exact counterfactual value for each bucket and an exact update to the regret for each bucket. Already in (Zinkevich et al. 2007), and subsequently in several succeeding publications, variants of CFR have been proposed which incorporate sampling. In general, these approaches perform quicker and less accurate iterations that use sampling to estimate counterfactual values (and therefore regrets) instead of computing them exactly. It has been shown that even in the presence of sampling, convergence to an equilibrium is still guaranteed (under certain conditions) ((Zinkevich et al. 2007), (Lanctot et al. 2009), (Gibson et al. 2012)).

The sampling approach we adopt is a variant of Public Chance Sampling (PCS) as described in (Johanson et al. 2012). In PCS (when applied to hold'em) a single five-card board is sampled on each iteration, but the set of possible private card holdings for each player is fully evaluated. One main advantage of this approach is that it allows us to evaluate terminal nodes more efficiently using an $O(n)$ algorithm rather than the naive $O(n^2)$ approach (see (Johanson et al. 2011) and (Johanson et al. 2012)).

A single iteration of CFR implements a single tree traversal. We can view that traversal as a combination of a forward phase and a backward phase. As we traverse down the tree (the "forward phase") we pass down the opponent reach probabilities — how likely the opponent is to play to this point with any given hand. At terminal nodes we compute the counterfactual value for each hand of the target player given the opponent reach probabilities that have been passed down. As we traverse back up the tree (the "backward phase") we compute new regrets and a new strategy for the current iteration, and we update the accumulated strategy. See (Johanson et al. 2012) for a more detailed explanation.

We may describe the forward phase as operating at the card level. For example, the opponent reach probabilities are maintained per pair of hole cards. At the terminal nodes, we compute counterfactual values for each pair of hole cards the target player could have. In contrast, the backward phase requires bucket-level values. Counterfactual values and regret updates are computed for each bucket.

Our approach differs somewhat from normal PCS in that we choose to sample multiple boards on a single iteration.

To understand how this works, let us first examine the perfect recall case. The refinements needed to handle imperfect recall will be touched on later.

In order to sample multiple boards, we will perform multiple first passes, one for each sampled board. These first passes perform the role of the forward phase of regular CFR. During a first pass, opponent reach probabilities are maintained as we traverse the tree, and we compute counterfactual values at terminal nodes. The counterfactual values computed at the terminal nodes are aggregated at the bucket level and accumulated over the multiple first passes. We then perform a single second pass that passes back the accumulated bucket-level counterfactual values and updates the regrets and the accumulated strategy. This second pass performs the function of the backward phase of regular CFR.

The above algorithm suffices for systems with perfect recall, but not for systems with imperfect recall. (See, e.g., (Waugh et al. 2009) for discussion of imperfect recall and CFR.) With perfect recall, we can return counterfactual values at the bucket level from leaf to root. Street transitions are not a problem; the counterfactual value for a turn bucket is the sum of the counterfactual values of the corresponding river buckets. However, in the context of imperfect recall, a river bucket will not have a unique predecessor on the turn so we cannot simply pass counterfactual value back in the straightforward way.

With imperfect recall, we need to aggregate bucket-level counterfactual values not just at terminal nodes, but also at all street transitions. To accomplish this we need to pass back counterfactual values during the N first passes. These values are at the card level (e.g., the counterfactual value for $AhKd$ on a board of $As9h7s3d2c$); we aggregate them at the bucket-level at each street transition (as well as at terminal nodes). With these aggregated bucket-level values we can perform the second pass.

3 Distributed Architecture

To produce competitive poker bots, most researchers have attempted to solve as large an abstraction as possible. For systems that maintain their data in memory, the limiting factor for the size of the abstraction is typically the amount of RAM available. The cost of the RAM and the cost of a machine that can handle large quantities of RAM are often the main drivers of the total expense of the system.

In (Jackson 2012) we described a distributed approach that allows large games to be solved in a cost-effective fashion using commodity hardware. The key ideas are to maintain the regrets and accumulated strategy on disk, rather than in memory, and to distribute processing across multiple machines. Low-cost machines with limited RAM can be used with this approach.

This distributed approach works because CFR parallelizes well. For Slumbot NL we partition the game tree along two dimensions. First, by the preflop betting sequence (i.e., the sequence of actions leading to the flop) and second by the public cards. There are 835 preflop betting sequences in our abstraction. For the river we divide the game tree eight ways based on the public cards. So we have 6680 river "tasks" that can be performed largely independently. In our implementation, these 6680 tasks were divided across nine machines. There is also a single preflop task that is handled on a central machine.

Each task has a fixed assignment to a particular machine; i.e., that assignment never changes over the course of training. This allows the data for that task to be written to local disk, rather than to some central location, which reduces network overhead. The only information that needs to be passed over the network are opponent reach probabilities (output by the preflop task and sent to all the workers) and counterfactual values at flop-initial nodes (sent back from workers to the central machine).

4 Sampling

As alluded to previously, there are a number of variants of CFR with different types of sampling. It may be helpful to view the range of sampling algorithms as points on a spectrum. At one end of the spectrum are sampling variants that perform slower and more accurate iterations. Vanilla CFR (i.e., no sampling) would be the extreme example of this. At the other end of the spectrum are sampling variants which perform faster and less accurate iterations. Outcome sampling ((Lanctot et al. 2009)), in which only a single path through the game tree is evaluated on each iteration, represents perhaps the opposite end of the spectrum from vanilla.

PCS as described in (Johanson et al. 2012) samples one one board per iteration. While it does not sample as aggressively as, say, outcome sampling, it nonetheless traverses only a small fraction of the game tree (note that there are over 2.5 million five-card boards possible in hold'em even after accounting for suit isomorphism). Our variant of PCS in which we sample multiple boards per iteration allows us to adapt PCS in the direction of slower and more accurate iterations. The sampling rate — i.e., the number of boards sampled per iteration — is effectively a parameter that can be used to tune the speed versus accuracy trade-off for iterations.

The disk I/O in our system that is not present in a purely memory-based system adds a fixed cost to regret (and accumulated strategy) updates. For that reason, in our system it is advantageous to have slower, more accurate iterations. With slower and more accurate iterations we will perform fewer updates of the regrets and the accumulated strategy before convergence. Specifically, for Slumbot NL, we elected to sample $1/50000$ of the five-card boards on each iteration, which amounts to about 50 boards per iteration.

In general, optimizing the sampling rate is difficult because it is dependent on game-specific factors and results from smaller games may not extrapolate to larger games. We adopt the rule of thumb that the sampling rate should be large enough so that almost all buckets get a regret update on any given iteration.

With random sampling, a certain amount of variance is inevitable in the short run. For example, some boards will be sampled multiple times before other boards are sampled even in a single time. This can lead to inaccurate estimates of counterfactual value in the short run. For example, suppose by chance we sample a larger number of boards containing a 5 than the long-run expected quantity. This will lead to preflop hole card pairs with a 5 having their counterfactual values overestimated.

We attempt to address this by employing a form of quasi-monte-carlo sampling. We generate a random permutation of the 2.5 million possible five-card boards (which we call a "cycle") and, rather than randomly sampling a new board on each iteration, we iterate through this sequence. This guarantees that over the first 2.5 million sampled boards every board will be sampled exactly once. If more iterations are required, then we perform a second cycle with a new permutation. (In practice, for Slumbot NL we will not make it through the first cycle before training is completed.)

5 Abstraction

The abstraction we adopt for Slumbot NL can be viewed as the combination of a betting abstraction and a card abstraction. Following standard practice, we treat card abstraction as a clustering problem. Combinations of cards that are similar are clustered into "buckets".

The betting abstraction is a bit different. For example, in a certain game state our abstraction might allow bet sizes of ten chips or twenty chips but not any intermediate amount. These allowed bet sizes are exact amounts (and not, e.g., clusters of multiple possible bet amounts). At runtime we typically map the actual observed bet size to one or more nearby bet sizes handled in the abstraction.

5.1 Betting Abstraction

Our betting abstraction permits a variety of distinct bet sizes. We allow more sizes for the initial bet (on a given street) than for raises on the theory that initial bets are more common than raises. Along the same lines, we also allow more bet sizes for raises than for reraises ("three-bets") and more sizes for three-bets than for four-bets. For initial bets we allow the following eleven bet sizes expressed as fractions of the money that is in the pot:

$0.25, 0.5, 0.75, 1.0, 1.5, 2.0, 4.0, 8.0, 15.0, 25.0, 50.0$

For raises we allow the following eight bet sizes:

$0.5, 1.0, 2.0, 4.0, 8.0, 15.0, 25.0, 50.0$

For three-bets, the following three bet sizes:

$0.5, 1.0, 2.0$

For four-bets (and any subsequent raises), we only allow a pot size bet.

In addition to all the bet sizes listed above, an all-in bet is always permitted.

At runtime we use the translation scheme described in (Ganzfried 2013) to map the actual bet size of the opponent to bet sizes that are in our abstraction.

Our betting abstraction is symmetric meaning we do not allow a different set of bet sizes for the opponent than we do for ourselves.

5.2 Card Abstraction

Our card abstraction is hierarchical: we have a set of buckets for the public cards and a set of buckets for the combination of public and private cards. For example, for the river the public buckets are a clustering of all the possible five-card river boards. The private buckets represent a clustering of all the seven-card board-plus-hole-card pair combinations. To produce the card abstraction we first cluster the public cards to form the public buckets, and then we subdivide these public buckets to form the private buckets. This ensures that the abstraction is hierarchical.

For the public buckets, we characterize boards using features that correspond to how well those boards connect with a strong range of hands. For the most part these features measure whether the boards are rich in high cards or not.

For the private buckets, we characterize hands with features getting at hand strength and hand potential.

Given a feature representation of hands or boards, buckets can be produced by a clustering algorithm such as k-means.

5.3 Size of Abstraction

Slumbot NL's abstraction has 5.7 billion information sets and 14.5 billion information-set/action pairs. There are about six million betting sequences. We use between 2,000 and 4,000 imperfect recall buckets for each postflop street (3,904 on the flop, 3,602 on the turn, and 2,173 on the river). For preflop, we handle each of the 169 strategically distinct hands as a separate bucket.

Note that we devote most of our capacity to the betting abstraction as opposed to the card abstraction. This largely is a reflection of the fact that we are using imperfect recall buckets for the card abstraction, but not for the betting abstraction. It also reflects our judgment that a reasonably high quality card abstraction can be achieved with relatively few buckets, but that the ability to understand the difference between various bet sizes is vital.

References

Ganzfried, S. 2013. Action translation in extensive-form games with large action spaces: Axioms, paradoxes, and the pseudo-harmonic mapping. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-13).*

Gibson, R.; Lanctot, M.; Burch, N.; Szafron, D.; and Bowling, M. 2012. Generalized sampling and variance in counterfactual regret minimization. *Proceedings of the Twenty-Sixth Conference on Artificial Intelligence (AAAI-12).*

Jackson, E. 2012. Slumbot: An implementation of counterfactual regret minimization on commodity hardware. *Proceedings of the 2012 Computer Poker Symposium.*

Johanson, M.; Waugh, K.; Bowling, M.; and Zinkevich, M. 2011. Accelerating best response calculation in large extensive games. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11).*

Johanson, M.; Bard, N.; Lanctot, M.; Gibson, R.; and Bowling, M. 2012. Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. *Autonomous Agents and Multiagent Systems 2012 (AAMAS-12).*

Johanson, M. 2013. Measuring the size of large no-limit poker games. *University of Alberta Technical Report.*

Lanctot, M.; Waugh, K.; Zinkevich, M.; and Bowling, M. 2009. Monte carlo sampling for regret minimization in extensive games. *Advances in Neural Information Processing Systems 22 (NIPS)*.

Shi, J., and Littman, M. 2000. Abstraction methods for game theoretic poker. *Computers and Games 2000*.

Waugh, K.; Schnizlein, D.; Bowling, M.; and Szafron, D. 2009. A practical use of imperfect recall. *Proceedings of the Eighth Symposium on Abstraction, Reformulation and Approximation (SARA)*.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2007. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems 20 (NIPS)*.