Pokerbots Course Notes

Massachusetts Institute of Technology
6.176: Pokerbots Competition*

IAP 2022

Contents

*Contact: pokerbots@mit.edu

1 Lecture 2: Poker Strategy

This lecture is taught by Matthew McManus[2]. All code from lecture can be found at the public GitHub repository mitpokerbots/reference-lecture-2-2023. The slides from this lecture are available for download on GitHub.

At this lecture (and all others), we raffle off a pair of Sony MX 4s. Attend lectures in person if you want a chance to win!

We'd also like to thank our 13 Pokerbots 2023 sponsors for making Pokerbots possible. You can find information about our sponsors in the syllabus and on our website, and drop your resume at pkr.bot/drop to network with them. Our sponsors will also be able to see your progress on the scrimmage server, giving you a chance to stand out over the course of our competition.

Next, a reminder to check out Piazza Post @16 to find potential teammates.

Finally, our TAs recorded an introduction to Python and posted it in Piazza Post @9. This post also contains resources on introductory Python, and our course staff is happy to help in Office Hours![1]

1.1 Principles of Poker

1.1.1 The Strength Principle

This principle plays along with the strength of the hole cards. This involves betting strong hands, checking middle hands, and folding or bluffing weak hands.

1.1.2 The Aggression Principle

Betting and raising moves are considered aggressive, while checking and calling are considered passive. In general, aggressive moves are considered better than passive because it gives you two ways to win - one, by either forcing your opponent to fold to your bet, or by beating the opponent at the showdown.

1.1.3 The Betting Principle

A successful bet must be able to do one of three things - one, it should try to force a better hand to fold, a weaker hand to call, and cause a drawing hand to draw at unfavorable odds. If a player doesn't believe a bet could accomplish any of these things, a bet may not be the best option.

1.1.4 The Deception Principle

This principle simply recommends to never be following the same principle all the time. The idea behind this is to always keep the opponent guessing, and to not let them notice common patterns that they could take advantage of later.

1.2 Hand Types

A good way to understand hand types is by looking through examples. Slide 15 shows a board on the turn, with a 2c, Kd, Th, and 2s showing. Regarding hand notation, cards are described by value (2 through K) and suit (clubs, diamond, hearts, and spades). Note that it is impossible to have a flush on this board because regardless of the unknown fifth board card, at most two cards on the board could share a suit, which allows for at most four cards sharing a suit if the two private cards also match (five cards sharing a suit are required for a flush). Also, a pair of 2s is showing on the board—a board that has a pair on it should be played differently, as the baseline is stronger (everyone has a pair or better). The hand Jc and Qc in this scenario is called a ``drawing hand''; once the fifth and final board card is drawn, the board will either be really good or really bad for us. This provides us with some certainty on the river, which is very good to have in an uncertain game like poker. We could get a straight (with either a 9 or an A), so we have an "open ended straight draw" right now (there are two ways to get a straight with the river). If we only had

[1]The Office Hours schedule can be found in Piazza Post @8.

one way to get a straight, it would be a "closed straight draw", or "gutshot draw." Since 9 and A give some certainty of winning, these are called "outs."

Now, lets compute hand strength. Hand strength is computed on a scale from 0 to 1, and it measures how many hands out there will beat our hand. The graph on slide 16 gives us our hand strength in this scenario as a histogram, taken over all the 46 possible rivers. The likelihood of us having a particular hand strength corresponds to the area of the histogram bucket. If we get a 9 or an A, then our hand strength will be about 1, which is why there is a bump on the graph around 1.0. Similarly, the two small bumps above 0.5 and 0.6 in the graph correspond to us drawing a J or a Q respectively to give us a two pair. You might notice that we have a large probability of having a losing hand with strength ∼0.4—this corresponds to us not hitting anything that will improve our hand on the river. Note that this graph is very bimodal—this supports our intuition that a drawing hand provides avenues to either a very strong or very weak hand. The bimodal distribution is something we like to see, as it will settle to a point mass that has clear hand value by the end of the round.

Slide 17 has another example scenario, this time with pocket 3s instead of the J and Q. This hand already gives you a two pair, which is better than nothing, as the board already has a pair; however, this hand loses to a lot of possible hands, such as any higher pocket pair, a K, or a T, since this would make a higher pair with the board. This hand does beat nothing though, which is not too bad. Looking at slide 18, we see the hand strength graph for this pair—note that this graph looks very different than the one on slide 16. In Texas Hold'em, a player has roughly a 50% chance of getting a pair or better on a random draw, explaining why there's a bump on the distribution around 0.5. You might also notice that there is a bump around 1.0—this will happen if the last card is a 3, which would give us a "full house," one of the best possible hands on this board. A problem with this board though, is that even once all 5 cards are revealed on the board, we still don't know if we're winning or not—we don't know what the opponent has. This is different from the drawing hand, where we know exactly whether we're likely win or not based on the fifth card. Therefore, even though the low pair has more mass to the right than the drawing pair, its uncertainty means that it'll make us lose money more often because of how our opponent can fold if they have a bad hand or keep playing if they have a good hand (we are "adversely selected" against). When you bet high with a low pair, there aren't many opportunities for improvement, and you have a good probability of losing—you should be skeptical of playing longer when you have a hand with a distribution like that of the low pair.

Next, we'll talk about the ``made hand'' (slide 19). We'll talk a lot about hands that are good, because when you have a bad hand it's very easy to fold out. That's why the best pokerbots will be the ones that can differentiate good hands from better hands from the best hands, because they will know when to play and how. They're able to bet cleverly to extract the maximum value from their opponent. The made hand is also a two pair, and the K is called the "top pair." Even if the opponent has a T, we'll still win the two pair when it comes to hands, and we also have a good fifth card (the A), which is called the "kicker" or "tiebreaker" since it beats most cards.

Notice that even though we still have a two pair, the made hand has a very different hand strength from the low pair (slide 20). No matter what shows up as the fifth card, there are a lot of hands that our hand will strictly dominate, or win, in any scenario. In a game like poker with a lot of uncertainty, this is exactly what you want—and so the made hand is considered a good hand that will make us a lot of money. The reason why there is some mass around 1.0 is because we could get a full house if another K shows up. Note that there are still some hands that could beat ours (pocket As, for example). If you run into a scenario like this you could lose a lot of money thinking you have a great hand but run into an even better hand. However, in the long run betting there is net positive gain.

The last hand we'll be talking about is ``the nuts'' (slide 21); when we talk about the nuts, we mean that there exists no better hand than ours on the current board. When you have the nuts, you want to extract as much money from your opponent as possible; you want to bet, raise, or call every possible turn. The distribution for the nuts (slide 22) is basically 1: it is a point mass. In this case, by the turn the nuts are 2h 2d. We are simplifying this calculation a bit, as there are exactly two hands that would beat this: the opponent has pocket K's and the fifth card is a K, or the opponent has pocket T's and the fifth card is a T. This is an extremely unlikely scenario, one where both players have a four-of-a-kind but the opponent has a better one. If this scenario occurs and strong hands go against each other, both players will be aggressively betting and the pot will be massive.

Even though there exists a scenario in which our four-of-a-kind loses, that does not change the fact that

on the current board, this four-of-a-kind is the best possible hand. There is a small possibility that the fifth card will change the board to introduce a better hand, but there are still no two cards we would rather be holding right now than a pair of 2s to match the pair of 2s showing on the board.

Again, remember that you want your pokerbot to focus on the hands that are good, as you'd often be better bluffing with a drawing hand than bluffing with nothing, even if you're confident in your bluffing abilities.

1.2.1 Board Types

In Texas Hold'em, there are also types of *boards* you must consider. A simple example is the one we've been using this whole time: a 2, K, T, 2 board. In the first scenario, you have pocket Aces (slide 24). This hand is much better than almost every hand, unless your opponent has a three-of-a-kind or better. You would feel great about your hand strength here. Instead, let's think about the scenario in slide 25: a 3, 4, 5, 6 board. You'd almost certainly feel poorly about your hand strength here, even though you have the highest possible pair. The board has four clubs and several straight, flush, and straight flush possibilities. On a board like this, you can end up losing a lot of money to someone who gets an unlikely hand for an ordinary board, but a more likely hand in this "drawing board."

1.3 Pot Odds

Pot odds are very related to the idea of maximizing expected value. We're going to begin with a claim, which is that for any state of the game, there exists some probability of winning. Even if our opponent adopts a strategy that incorporates randomized behavior, this probability p still exists. Given that we have some probability of winning, we can calculate an expected value using the equation on slide 29. If we continue to play, the expected amount we'll win is $p \cdot \texttt{pot.grand\_total}$, and the expected amount we'll lose is $(1-p) \cdot \texttt{cost\_to\_continue}$. Note that these amounts aren't symmetric. That's because in poker, you should consider every cost a sunk cost; that is, never worry about money committed to the pot in the past, as it is already gone. The only cost you're considering is the further cost of continuing, which you're using as some stake to win all the money in the pot. When it comes to expected value, we don't want to make a negative expected value decision - in fact, we wish to maximize expected value. Note that if the expected value is 0, we're indifferent when it comes to folding or continuing.

The next step is to perform some algebra to separate out the probability of winning, and this gives us a cutoff for whether or not we stay in the game. We call the right hand side of the new inequality for p the "pot odds:"

$$\frac{\texttt{cost\_to\_continue}}{\texttt{pot.grand\_total} + \texttt{cost\_to\_continue}}.$$

If we know our opponent's strategy well enough to calculate p, we'll never have to worry as we can always calculate pot odds to make a positive expected value decision against every bet.

Now consider an example for calculating pot odds (slide 30). We mentioned this example briefly, but we did not consider pot odds. On average we'd expect our opponent to win this board, as they have a made hand and we're banking on a straight to win. Remembering our distribution, only ∼2 out of 13 cards will complete our straight, but our opponent already has a good hand. Suppose our opponent puts 10 more chips into the pot, and we now have to decide whether or not to continue. Let's calculate the pot odds using the previous formula (slide 32). If we think our probability of winning is greater than 0.1, we should continue, and otherwise we should fold. How do we calculate our probability of winning though? One method is by "counting outs." We can estimate our p by calculating the probability of getting the cards that complete our hand. Given our pot odds example, we have 8 possible out cards (4 As or 9s) and therefore around 16% chance of completing the hand. This is greater than our calculated pot odds of 0.1, telling us to call.

This explains why drawing hands are so much better than you might otherwise expect; estimating our probability of winning is easy, so it's easy to make good decisions with them. Our opponent gave us pot odds that look good enough for us to stay in with our drawing hand and make money on average, so they "underbet." This is highly undesirable in poker, because it will let your opponent stay in the game when they should've folded a long time ago.

Let's look at this from our opponent's perspective. If our opponent made a higher bet that caused us to fold, they would've won 80 chips 100% of the time. However, with us staying in the game, they will win 90 chips ∼85% of the time, for a win of ∼76.5 chips on average. They are worse off by underbetting!

This is a little unintuitive to people unfamiliar with poker; you might think that you should just bet proportionally to hand strength, but you should generally not underbet as this would give your opponent opportunities that they would've otherwise not had. In addition, it reveals too much about your hand. If your opponent had instead bet more confidently, you would've folded as your pot odds would've looked a lot worse. For example, if your opponent had gone from 40 to 80, your pot odds would be 0.25, which is not good enough to merit a call.

There's also something called "reverse pot odds," which unsurprisingly, are pot odds for your opponent that give your opponent the opportunity to call. When we talk about reverse pot odds, it means that we're considering whether or not our opponent will stay in the game. The way that we can give our opponent opportunities is by "overbetting" relative to the size of our pot, which will give our opponent the possibility to exploit the pot odds and take our money. Overall, when considering pot odds you gain much more control over the size of the pot and maximize expected value.

Now, we'll consider an example bot that you may have seen on the scrimmage server: the "all-in bot." Our opponent goes all-in before the flop which is a deterministic strategy. Note this is easy to beat, we can check-fold, letting our opponent collect the blinds until we are dealt a high pair to crush them and win big.

1.3.1 Implied and Reverse Implied Pot Odds

At times, we can better estimate our pot odds by making implied inferences about money we could win on later streets of the round. This is how we calculate "implied pot odds". This allows us to call when we would otherwise fold when considering normal pot odds, especially when holding a "drawing hand". Quantitatively, this can be seen by updating our pot odds cutoff with an "amount you expect to win" in the denominator accounting for future street winnings (make sure you factor in all future streets). Let's consider a simple example with our previous pot odds example. Say that our opponent raises by 20 instead of 10. Recalculating our pot odds, we get a value of 0.167 which is greater than our estimated 0.16. Therefore with regular pot odds we should fold. However with implied pot odds, we first make the assumption that our opponent has always been 1/4 of the pot. Then we estimate the next bet will be 1/4 of 120 which is 30. Recalculating our implied pot odds gives us 0.133, telling us to call instead of fold now.

You must be weary of "reverse implied odds" however. This is the amount you could expect to lose after hitting your draw, and balances out our implied pot odds. This serves more as a warning especially when we do not hold "the nuts." An example with reverse implied odds is hitting our straight with an A on the river in the previous round, but our opponent holds a full house with A, 2 causing us to still lose our money. A good pokerbot will be able to find an even balance between implied and reverse implied pot odds to find your true pot odds.

1.4 Ranges

When we're faced with a bet, everyone knows the pot odds. If we can estimate p better than our opponent, then on average we'll make money. Ranges are the types of hands that you play—when playing poker, you want to be restrictive about the hands that you play. Our opponents *range* is the distribution of hands we expect them to hold. We can estimate this during the course of the game—if our opponent hasn't folded late in the game and has bet a lot of money, we'll expect them to have a better distribution of cards. This means that ranges are key to calculating our probability of winning, which affects pot odds and our decisions. In a pokerbot, implementing ranges is much more difficult since it is not strictly quantitative. However, we recommend you still experiment with calculating ranges and updating them between rounds. Use your opponents' actions to narrow this range and remember "everything conveys information." Also, always compare your cards to their range and be aware of the range your opponent perceives from you.

When it comes to poker, the best play style is typically "tight-aggressive," which means folding early and often with bad cards and betting aggressively with good cards. If your strategy is more tight-aggressive than your opponent's strategy, then you will often win more money on average, because you will get into

high-stakes scenarios with better cards. For regular poker, the "tight-aggressive" strategy is folding ∼70% of hands preflop and betting frequently with the rest.

1.5 Variant Strategic Considerations

Considering our River of Blood Hold'em variation, we note that there is around a 50% chance at that the river card is red, which means there is around 50 percent chance that there will be more than 5 rounds played. All odds are inflated in this variant, and it is important to remember that all cards in your hands are "blockers". Consider ranges in the context of the number of streets played in the round, and if bots are betting randomly, wait for a good time to trap them on the later streets.

1.6 Coding `reference-lecture-2-2023` Bot

We'll build off our `reference-lecture-1` bot, where we created an all-in bot when we recognized a pair. For this bot, we want to run a Monte Carlo simulation to help estimate strength values using randomness and sampling. This simulation allows us to run games many many times to calculate what happens on average. The calculated win probability can then help us determine how we raise, call, and fold.

We start by initializing a new method called `calc_strength`. At the very beginning of our method, we want to first evaluate each of our hands; we do this by using `eval7`, a poker hand evaluation package in Python. We call `eval7.Deck()` to construct a deck that we will use for simulating multiple rounds of poker, and start by removing the cards that we in our hole_cards have from this deck as they cannot appear again. To evaluate the strength of our hole_cards, we will assign it a score (initialized to 0) representing its win frequency. Going through the numerous iterations of simulated potential rounds, we then declare variables like `_COMM` and `_OPP` to represent how many community cards we can draw and how many cards the opponent has pre-flop, respectively. We then draw the _OPP and _COMM cards from our constructed deck, and evaluate our hand strength against the opponent hand strength using eval7. To denote a win in our simulation we add 2 to our score, and to denote a tie we add 1; losing does not impact our score. After all iterations have run, we may now divide our score by twice the number of iterations to yield the calculated strength (or win probability) of our hand, stored in `hand_strength` – which we will use later on.

Now that we have set up our Monte Carlo simulation , we will want to use this in our `get_actions` method. Here we want to calculate our raise values, and for this example we want to play a little more conservatively pre-flop and a little more post-flop. This is shown by us multiplying `(pot_total + continue_cost)` by a factor of 0.45 on the pre-flop street against 0.7 on any post-flop street. Note that the my_pip and continue_cost factors of the raise_amount are the minimum contribution. Once we do this, we will want to check how much it costs to make such a raise against the min and max raises, and then commit the action depending on if we have enough chips in our stack, else call or check.

Note that working with Monte Carlo simulations can be very time-intensive, so we will want to limit the number of simulations we run. Accordingly, in this case we decide to run 100 simulations—as you can see in the `_MONTE_CARLO_ITERS` variable.

Finally, we get to the part where we implement our hand win probability from the Monte Carlo simulation. But first, if our opponent raised on a board, we expect them to have a good hand; thus we will want to tone down our strength by an intimidation factor that scales to the amount that they bet. This is not perfect since you should measure their bet to the size of the pot for a more accurate estimate of their strength. We then redefine our strength with the equation

$$\max([0, \texttt{strength} - \texttt{\_SCARY}]).$$

If our strength is greater than or equal to our calculated pot odds, then we may want to raise or call depending on our strength value and relative probability, and if not then we will want to fold. On the other hand if we are the first to bet, we can simply bet under the probability determined from our strength, and check otherwise.

Another important aspect of the lecture bot is the use of random.random() that generates an arbitrary random number between 0 and 1. This is used in cases to challenge bots that are more deterministic, and this ensures that the same outcome will not always happen if run twice exactly, and this is helpful in cases where the opponent bot is hard-coded to put some situations against your favor.

Comparing the results of this bot to `reference-lecture-1` yields a large chip lead over our opponent, demonstrating the power of poker theory in improving your bot.

Finally, a reminder to always commit code back to git![9] There's nothing worse than ending a marathon coding session with a hard drive failure and losing all your progress.