Matthew Peveler
CSC370: Artificial Intelligence

Final Report: Reinforcement Learning for Texas Hold'Em

INTRODUCTION

Through the use of reinforcement learning, a bot can be made to learn a game (or other things) with no prior knowledge of outcomes, so long as it knows when it has won and when it has lost. This learned knowledge could then be applied by the bot to better make decisions as to what it would want to do in a given situation when playing against a human opponent. It would allow for convincingly real artificial intelligence with enough learning done on a varying of actions and states.

BACKGROUND

Reinforcement learning is the broad topic where an artificial intelligence bot generally may start with no knowledge of a situation, but through repeated action, learn the optimal action for a given situation. It can be applied to things such as finding a way through a maze, or something like card games. It is very well suited for card games as there are many different hand combinations in all the popular card games, and through reinforcement learning, run over a wide number of hands, it can generate an appropriate action based on the probability of a given hand winning. This can be done with absolutely no knowledge of the game beyond what a particular hand means, and then whether or not that hand won at the end of the round.

For this particular project, the card game of choice was Texas Hold'em. This poker type game consists of four rounds of betting, after a number of cards are dealt. The first round consists of each player getting two cards which makes up their pocket. The following three rounds all deal with having cards placed into the "community pile", with three cards placed for the first round, and then one card placed down for the following two rounds. The players around the table are then free to make a hand out of any of the seven cards on the table (the two in their hand, and the five community cards). The players compare their hands and whoever has the stronger hand wins the pot. If it's a tie, then the pot is split between the players. There are 10 different types of hands (card high up to royal flush) with several having

different strengths based on the kicker (the cards not making up the particular type of hand. So for two pair, the kicker would be the highest card available not in one of the two pairs). After $\binom{52}{7} = 133,784,560$ getting cards, the players are free to make bets on the game, check (if no bets have been made), call (if a bet was made) or fold. There are than many different strategies that exist on whether or not to continue with a hand at different parts of the game, bluffing against your opponent, and several other factors that contribute to making Texas Hold'em a hard game for an AI to perfectly replicate beyond at least perfectly knowing how strong a given hand is. There are distinct hands in Poker. This does include repeats for hands that consist in multiple different suits (such as four different ways to have a pair of twos), but goes to show just how many different hands there are, and that it's impossible for a human to know the probability of any given hand during play.

METHODOLOGY

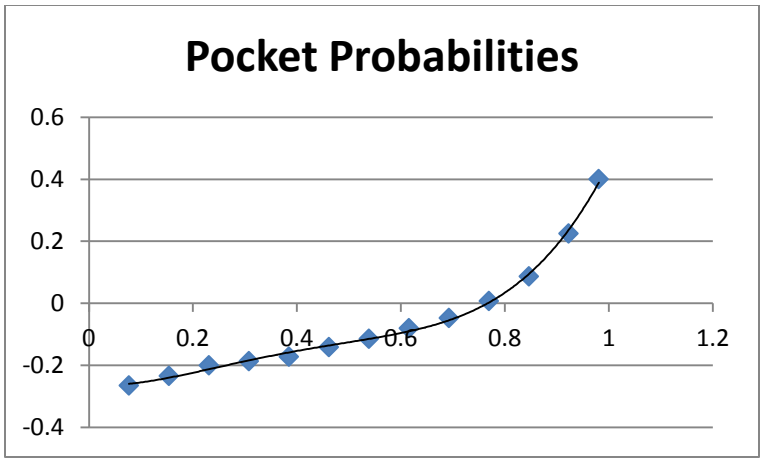
The bot initially started the project knowing only how to get the strength of a particular hand (these calculations can be seen from lines 155 to 392 in game.py). However, beyond that initial knowledge (easily available in a slightly less mathematical format to any player of the easily), the bot contains no other knowledge about the game. It is then through reinforcement learning that the bot learns what constitutes a good hand and what isn't a good hand at each part of the game by playing against itself, which is done in the setup.py file. For this particular example, only 200,000 rounds of poker were played (generating 400,000 hands, though there are many repeated hand strengths during execution). Each stage of the game has its own lists that contain each previous hand strength encountered (stored only once). The program first checks to see if a given strength has been encountered once, and if it hasn't, it then adds a new value to the lists, saving the index (it just saves the index of number in the list if it has been encountered before) and then uses that to update the other lists for that particular hand strength. It gets the index for pocket, flop, turn, and river lists which it then passes to an update function so that it can update the probability list for all the different rounds. It uses an average function to update the probability based off previous results for a particular hand. As stated previously, it does this for 200,000 rounds before then saving the results to several files (one for each round) that can be viewed at results/. The output for the each file of the format

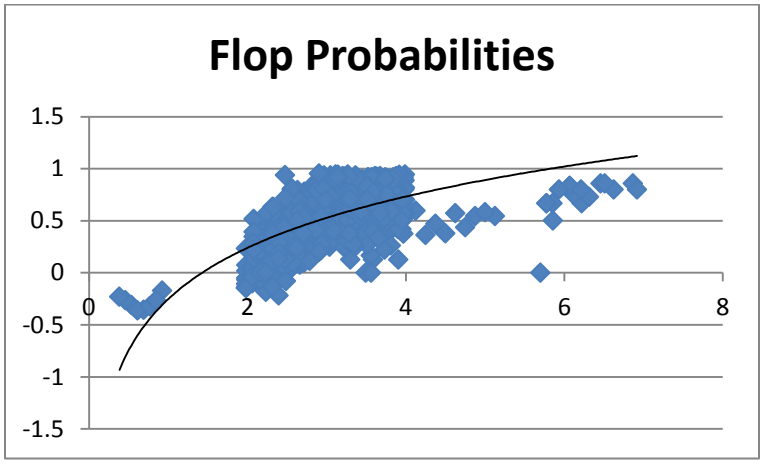
“#.###:#.###:##” where the ‘:’ symbol acts as a separator. The first batch of numbers is the particular hand strength, the second set is for probability of winning with that hand, and the final number is for how many times that hand was encountered. Each result file is a text file, but was imported into Excel to allow for graphical analysis of the results and provide the graphs shown below.

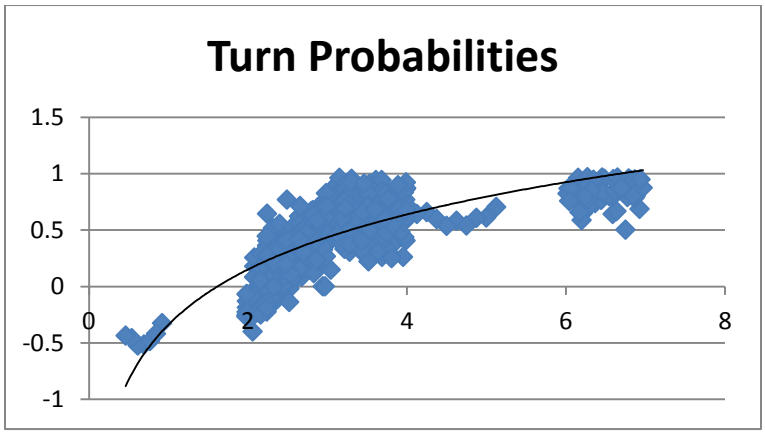
After calculating these probabilities for a hand strength, the next step was to make some way for the bot to be able to play a game of Texas Hold’Em (at this time, only against one other person). This is done through the main.py file. During each round, the bot checks its hand, and sees if it has encountered that particular hand strength before. If it has, it grabs the value from the probability list for that particular strength and makes a decision based off that. If the probability is below .3, the bot looks to check if it can, and fold otherwise. If it’s below .6, the bot will attempt to check or call. Above that, the bot will bet against the player. This does allow the player some amount of knowledge of a bot’s action unfortunately, but shows the computers power to analyze and remember hand states and probabilities for a multitude of values.

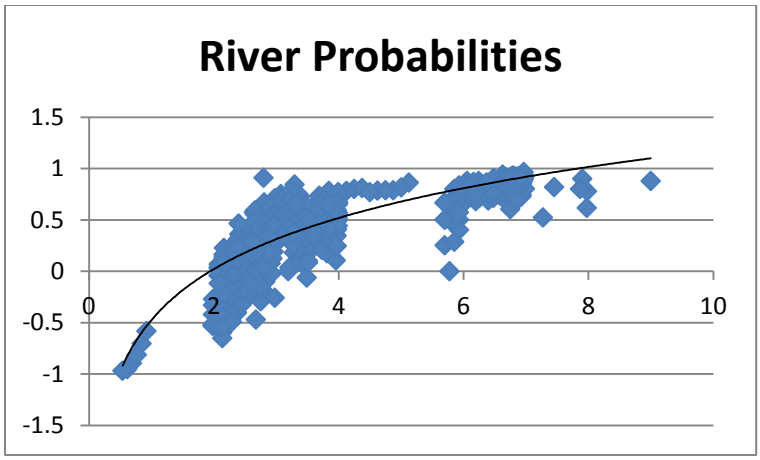
RESULTS

The following tables show the results of each round, with regards to hand strengths. Each whole number (starting at zero) is the weakest way to get that particular hand (so a pair of twos with 3, 4, and 5 kickers is a 1). The whole number is then split into sections that are equal to the number of different of combinations possible for a particular type of hand. As each round is highly unique, each graph shows a different round. Each chart has hand strength on the x-axis with the y-axis being probability of winning.


Pocket Probabilities
0.6
0.4
0.2
0
-0.2
-0.4
0
0.2
0.4
0.6
0.8
1
1.2


Flop Probabilities
1.5
1
0.5
0
-0.5
-1
-1.5
0
2
4
6
8


Turn Probabilities
1.5
1
0.5
0
-0.5
-1
0
2
4
6
8



While the results show a very wide spread on a given strength, this is due to not a significant number of hands showing this strength (less than 50) which might lead to variance. Cases that had less than 5 plays were removed from consideration due to their nature of being outliers. However, in each case, it shows that as hand strength increases, probability of winning increases in a logarithmic fashion up to 1, which is a guaranteed win. This makes sense for the game of poker and is what human players manage to grasp subconsciously, without having to do these calculations.

CONCLUSION

As shown above, the bot accurately was able to generate realistic probabilities (for the most part) for various different hands, and would then be able to act on these results. However, one thing that I didn't take into consideration was the low distribution of hands in the pocket (the first two cards dealt) and then how to best use that. Based off these results, only two hands would get above the line necessary for the bot to want to act on it. This means that perhaps a different calculation is needed for better generating the desire to act in the pocket situation, as well as needing to run far more trial runs as there was not a single pocket pair in 200,000 rounds which, while unlikely is not impossible. However, the longer the program ran, the closer to accuracy the results would come. The next step beyond that would be to allow for the bot to change the way its betting based off how the player is betting (the more bets would make the bot perhaps more likely to fold) as well as implementing a way to bet varying amounts. This step could most likely be applied in another reinforcement algorithm, but it

wasn’t done for this project. As a proof-of-concept though, the bot can play poker to some extent against a player that is unknowing about the bounds of its decision making.