Machine Learning Project

Application of Machine Learning on an Imperfect Information Game
(Poker)

Project by

Jaishree Dhage (140905350)
Tuhin Khare (140905056)

ABSTRACT

Game playing is a favourite application area of Artificial Intelligence, specially when it comes to imperfect information games where the optimal gameplay strategy is to be chosen without the complete information parameters. The game of poker offers a clean and well defined domain to investigate some of the truly fundamental issues in computer science, such as how to handle deliberate information and how to make intelligent guesses based on partial knowledge. Success in the game of poker not only requires a good understanding of the ground rules but also requires a strategy of informed guesswork, risk management and opponent modelling. Computer Poker Bots is a system of intelligent programs that attempts to model a human by learning these qualities with the goal of winning the game. The agents are adversarial and not necessarily equally smart, but provide a wide array of opportunities for research in machine intelligence. This paper mainly depicts the idea of developing a poker bot using concepts of reinforcement learning and bayesian decision theory.

INTRODUCTION

The game of poker is currently the newest and one of the most challenging areas for the researchers in the field of Artificial Intelligence and Machine Learning, being an imperfect information game in which each player has some hidden information which is central to the game in the form of private cards, developing a computer program which could play any variant of the game is giving the researchers a hard time. Many approaches have been used in order to develop agents which can mimic playing strategy to that of humans along with improving their gameplay strategies with every game they play.

Game Description

A hand of poker begins with the pre-flop, where each player is dealt two hole cards, face down, followed by the first round of betting. Then three community cards are dealt face up on the table, called the flop, and the second round of betting occurs. On the turn, a fourth community card is dealt face up and another round of betting ensues. Finally, on the river, a fifth community card is dealt face up and the fourth (final) round of betting occurs. All players still in the game turn over their two hidden cards for the showdown. The best five card poker hand formed from the two hole cards and the five community cards wins the pot. If a tie occurs, the pot is split. Typically the game is played with 8 to 10 players.

The order and amount of betting is strictly controlled on each betting round. We use 4 betting denominations: 1, 2, 5 and 10. When it is a player's turn to bet, one of five options is available: fold (withdraw from the hand, leaving all previously wagered money n the pot), call (match the current outstanding bet; if there is no current bet, one is said to check), or raise the bet (put something more than the current bet; if there is no current bet, one is said to bet). There is usually a maximum of three raises allowed per betting round. The betting option rotates clockwise until each player that has not folded has put the same amount of money into the pot for the current round, or until there is only one player remaining. In the latter case, this player is the winner and is awarded the pot without having to reveal their cards. There is a strategic advantage to being the last bettor in any given round; so to maintain fairness, the order of betting is rotated clockwise after each hand.

Our Approach to the Problem

This report describes a connectionist approach to the development of such an agent, through techniques like reinforcement learning and bayesian decision theory. In the approach used in this paper the work is divided i.e calculating the hand strength of the hand and choosing a strategy which would maximise the agents chances of winning the game.

The calculation of the hand strength is done without considering the dynamics of a multi-player game. This subpart of the agent's gameplay is called the *Learning Model*.

The *Playing Model* makes use of the hand-strength and the likelihood probabilities calculated in order to make a decision. This separation into learning and playing models enables us to manage a complex task better. The separation is also justified on the grounds that evaluation of hand strength is dependent only on the cards a player has – we are not determining relative hand strength in the learning model.

SCOPE

The major scope of the project are mainly to enable the bot to make decision which would maximise its chances of winning with every gameplay alongside the bot should also be able to memoize the sequence of actions which leads to the bots win and loss respectively, through a reward function which mainly provides a positive reward value for every action sequence vector which leads to the bots win and a respective negative reward value for every action sequence vector which leads to the bots loss. This is implemented using the concepts of reinforcement learning specifically Q-Learning. The bot uses these reward values in the action grid in order to memoize the action sequence vectors which lead to the bots loss or win.

The major techniques using which the bot functions are —

- Q - Learning
- Bayesian Decision Theory
- Likelihood probability calculation

The purpose of this project is mainly to understand how techniques in machine learning and artificial intelligence can be used in order to automate imperfect information games like poker, along with understanding the mathematical background such as Game Theory which lay the test bed such games.

SURVEY

Mathematicians and economists have studied aspects of poker for reasons ranging from purely monetary to modeling the dynamics of the game. One of the ways in which the problem has been attempted to solve is by using a simplified variation of the game. The simplification adds more constrains than the full-fledged game and hence makes it possible to analyze the game better. For example, the number of players could be limited to two only, or one could restrict the betting rules. The other approach has been to mathematically analyze the full-blown game and combine it with simulations, experiments and expert knowledge of "ad-hoc experts". This approach has been usually taken by expert players of the game with an inclination towards mathematics.

Though simplification is a common and useful technique for solving difficult problems, it may remove the complex activities that we are interested in studying. For example, Findler tried to model human cognitive processes and build a program that could learn. But his simplified approach was not very useful in the real world. Theoretical work by Koller et al provides an algorithm for finding optimal randomized strategies in two-player imperfect information competitive games. Their system builds trees to find the optimal strategy. But owing to the size of decision trees built, only simplified versions of poker can be solved.

GAPS

The following results came into the picture upon the game analysis of the project —

- Q Learning was not used as a learning technique in majority of the poker playing agents implemented till date.
- Usage of different player characterstics of player in order to make a comparison so as to how different players with completely different way of playing win or loose the through a series of successive games.
- To demonstrate an establishment of a nash equilibrium upon the action of reinforcement learning as the bot proceeds through successive games.

OBJECTIVES

We intend to devise a set of programs that is able to play the game of poker in way that reflects intelligent behavior. The programs are expected to model the human way of learning to play the game: initially they are only aware of the rules that govern the game and the goal to achieve, but have no strategies. By playing more often, the agents must learn ways to improve their chances of winning. Much like human players, all agents are not equally smart and may not make identical decisions in identical situations. Agents have memory but with varying retentive ability. Each agent also has to put up a face that is visible to others, and can be used as an input for making decisions. Finally, the agents are competitive and not collaborative – they are expected to do everything legal that may be reduces the chances of others.

Devising a poker agent is a difficult problem, and our approach to devising one has been to divide the task into independent sub-problems that can be tackled separately. Specifically, there are two major tasks that any poker player has to master: evaluating the strength of his cards, and making betting decisions that will maximize its chances of winning the pot (or minimizing the loss if the player cannot win the game). We tackle the first task without considering the dynamics of a multi-player game. This method is called the Learning Model. Making betting decisions is handled by a separate method, the Playing Model, and makes use of the output of the learning model, besides other playing information. This separation into learning and playing models enables us to manage a complex task better. The separation is also justified on the grounds that evaluation of hand strength is dependent only on the cards a player has – we are not determining relative hand strength in the learning model

The plan is to use three players 2 humans and 1 bot to find a population best-response. The bot is an *opponent model* bot which mimics the play of opponents observed in the population. The bot also acts as an *exploiter* bot which learns to optimise its return against the opponent population. It is intended that both should use the same basic connectionist structure with the opponent model being fitted by maximum likelihood and the exploiter being trained by policy-gradient reinforcement learning on simulated hands generated with the opponent model.
To be suitable for this plan, the basic structure must possess certain characteristics. It should employ soft selection; every legal action should be chosen with non-zero probability. This is for two reasons; firstly, so that when fitting the model, no action can have a zero-probability which would prevent the use of the logarithm function; secondly so that every branch has a chance of being explored by the reinforcement algorithm.

This structure should be parameterised by a finite (but possibly large) number of parameters. The output of the structure (probability of an action) should be a differentiable function of those parameters. Those derivatives should be in a closed form so that they can be calculated quickly.
It also should act quickly as the reinforcement learning algorithms envisaged uses simulated deals. Simulation only works if enough sample points can be taken to average out the noise. This means that time-expensive components should be avoided wherever possible. Time-expensive components would include any simulation within the strategy, any deep-tree search, or full enumeration of any variable with more than a few values.

METHODOLOGY

Learning Model

The goal of the learning model is to assess the strength of a particular hand of cards. Strength is defined precisely later in the section. Intuitively, the strength of a hand determines the expected utility value of the hand - greater the utility value, better are the chances of winning the pot. The learning model is based on principles of Bayesian decision theory. There are ten winning hands in poker: Royal Flush, Straight Flush, Four of a Kind, Full House, Flush, Straight, Three of a Kind, Two Pair, One Pair and No Pair. Each winning hand is treated as a class of a classification problem. The inputs on which the classification is to be made is the hand of cards, consisting of two hole cards and 3 to 5 community cards, that a player has to his/her disposal. According to Bayesian theory, the classification can be made on the posterior probability of a class given a hand.
Let $\omega_1, \omega_2, \omega_3, ..., \omega_n$ denote the various classes of winning hands. Let x represent the set of cards in a hand; x is thus the input feature vector. By Bayes' rule, the posterior probabilities of winning hands is given by

$$p(\omega_i|x) = \sum p(x|\omega_i)p(\omega_i) \quad i = 1 \text{ to } n$$

Here, $p(\omega_i)$ is the prior probability of reaching a winning hand ω_i. Given particular hand we calculate the posterior probabilities of all winning hands which we use to calculate the strength of that hand. The prior probabilities are easily obtained by elementary combinatorics: the prior of each class is the total number of hands that are possible for that class divided by the total number of possible 5-card hands.

There is no straight forward way to determine the likelihood probabilities $p(x \mid \omega_i)$. hence these need to be learnt. An important issue in determine the likelihood of hands is that the number of cards in the hand keeps increases from 5 in pre-flop to 7 before river. Hence we formulate the likelihood as,

$$p(x \mid \omega_i) = \prod_{c \in x} p(c \mid \omega_i)$$

where, c is a card in the hand x. Thus the task of obtaining hand likelihood is reduced to that of obtaining card likelihoods $p(c \mid \omega_i)$.

A simple procedure is used to determine the card likelihood: we play the game several thousand times with only one player. A counter is maintained for each card/class combination. In each game, seven cards are drawn at random, and we enumerate all possible winning hands that are possible using only 5 cards from the 7 drawn. If a card c is used in a making a winning hand (or class) ω_i

then the counter for c/ω_i is incremented. Finally, for each class, the sum of counters of all cards, s_i is computed. This gives the total number of games where that class was achieved. Dividing the counter for each card for that class by s_i gives the desired card likelihood $p(c \mid \omega_i)$.
Next, we define the utility value of a winning hand (or a class) in terms of its rarity: rarer the class, the better are the chances of winning the pot with that hand, and hence the higher is its utility. Currently we use the reciprocal of prior probability of a class ω_i as its utility,

$$U(\omega_i) = 1 / p(\omega_i)$$

A better utility function would incorporate not only the winning hand, but also the rank of the cards in the winning hand – this is useful in resolving ties. Actually, we do so depending on the cards available in a hand. The subset of cards that form a goal has a rank within all hands that belong to that goal. This position is captured by adding a hand- specific constant to the above utility function. Finally, given a hand of cards x, let ω_{max} be the highest order goal (winning hand) that can be formed using the hand. The strength of the hand S(x) is then defined as the utility of the best goal plus the expected utility of better goals. That is,

$$S(x)=U(\omega(max))+\sum_{\omega_i \in G} p(\omega i \mid x)U(\omega i)$$

where, G is the set of all goal states that are higher than ω_{max}. The strength value will be further used in the playing model.

Playing Model

The playing model can be very complicated. The goals that the playing model is expected to achieve are:

- to guess what the unknown cards are,
- to interpret facial expressions of others,
- to guess when others are bluffing,
- to understand betting patterns and to remember the outcome of previous games,
- to make bets – whether to fold, call or raise and the amount to raise,
- to decide expressions to make for others to see etc

Owing above.
to time constraints, we were able to address only a few of the goals described.

In the least, the playing model must be able to make a betting decision that improves the chances of winning for an agent. We use a simple "inward"-looking model for betting. The betting decision is based on two factors:

- the estimated strength of the hand calculated earlier, and
- the agent personality

We use an agent's "personality" to model different kinds of players. Some players are optimistic and raise bets even when they don't have strong hands in the hope that their hand will improve or that their opponents don't have strong cards. Others may want to play it safe, and make bets only when they are very sure of their winning chances. We use four classes of agents based on personality: aggressive-loose, aggressive-tight, conservative-loose and conservative-tight. The optimism of these agents decreases in the given order so that the aggressive agents are more optimistic than the conservative agents.

We have manually decided thresholds on the values of hand strength that enable an agent to make a decision. These thresholds divide the number space spanned by hand strength into 5 regions. For each agent type and each strength region, a set of probabilities is defined. These are the probabilities of specific betting decisions. In this way, given a hand and an agent type, the betting decisions are made with the assigned.

Table 1: Decision probabilities of aggressive-loose agent

Strength	Decision					
S	Fold	Check/Match	Raise $1	Raise $2	Raise $5	Raise $10
0 < s < 200	0.05	0.4	0.3	0.15	0.05	0.05
200 <= s < 500	0.05	0.4	0.25	0.15	0.1	0.1
500 <= s < 2000	0	0.3	0.25	0.15	0.15	0.15
2000 <= s < 10000	0	0.2	0.25	0.2	0.2	0.15
s >= 10000	0	0.25	0.25	0.25	0.2	0.2

Table 2: Decision probabilities of aggressive-tight agent

Strength	Decision					
S	Fold	Check/Match	Raise $1	Raise $2	Raise $5	Raise $10
0 < s < 200	0.15	0.35	0.2	0.15	0.1	0.05
200 <= s < 500	0.15	0.3	0.25	0.15	0.1	0.05
500 <= s < 2000	0.1	0.25	0.2	0.2	0.15	0.1
2000 <= s < 10000	0.1	0.2	0.2	0.25	0.15	0.1
s >= 10000	0	0.25	0.25	0.25	0.15	0.1

Table 3: Decision probabilities of conservative-loose agent

Strength	Decision					
S	Fold	Check/Match	Raise $1	Raise $2	Raise $5	Raise $10
0 < s < 200	0.2	0.45	0.15	0.1	0.05	0.05
200 <= s < 500	0.1	0.55	0.15	0.1	0.05	0.05
500 <= s < 2000	0.1	0.55	0.15	0.15	0.05	0.05
2000 <= s < 10000	0.05	0.3	0.2	0.2	0.15	0.1
s >= 10000	0.05	0.45	0.2	0.25	0.15	0.1

Table 4: Decision probabilities of conservative-tight agent

Strength	Decision					
S	Fold	Check/Match	Raise $1	Raise $2	Raise $5	Raise $10
0 < s < 200	0.2	0.65	0.1	0.05	0	0
200 <= s < 500	0.15	0.65	0.1	0.1	0	0
500 <= s < 2000	0.1	0.7	0.1	0.1	0	0
2000 <= s < 10000	0.05	0.5	0.2	0.1	0.1	0.05
s >= 10000	0.05	0.45	0.2	0.15	0.1	0.05

Likelihood Probabilities

Table 6: Likelihood probabilities of all cards for various winning hands

Rank	Royal Flush	Straight Flush	Four of Kind	Full House	Flush	Straight	Three of Kind	Two Pair	One Pair	High Card
Clubs										
2	0.00000	0.00000	0.01494	0.01651	0.04171	0.00185	0.01376	0.00757	0.01488	0.01215
3	0.00000	0.00000	0.02955	0.01680	0.05734	0.00567	0.01659	0.00936	0.01627	0.01194
4	0.00000	0.05351	0.03796	0.01503	0.04356	0.00873	0.01160	0.01053	0.01494	0.01245
5	0.00000	0.05351	0.01969	0.01735	0.04714	0.00741	0.01383	0.01019	0.01499	0.01224
6	0.00000	0.05351	0.01997	0.01393	0.04025	0.01400	0.01282	0.01059	0.01526	0.01286
7	0.00000	0.00000	0.03510	0.01175	0.03284	0.01022	0.01328	0.01013	0.01551	0.01281
8	0.00000	0.06933	0.01961	0.00893	0.03526	0.00930	0.01143	0.01038	0.01642	0.01236
9	0.00000	0.14649	0.01071	0.02052	0.03039	0.01309	0.01273	0.01189	0.01592	0.01275
10	0.00000	0.14649	0.02588	0.01088	0.03532	0.01148	0.01436	0.01004	0.01612	0.01238
J	0.00000	0.07716	0.03036	0.01577	0.02071	0.00739	0.01391	0.01021	0.01584	0.01226
Q	0.00000	0.00000	0.02287	0.01760	0.01725	0.00605	0.01074	0.01029	0.01657	0.01210
K	0.00000	0.00000	0.00856	0.01668	0.01605	0.00327	0.01099	0.00884	0.01642	0.01256
A	0.00000	0.00000	0.00902	0.00808	0.01525	0.00094	0.01385	0.00741	0.01668	0.01270
Diamonds										
2	0.00000	0.00000	0.01569	0.01311	0.00000	0.00250	0.01421	0.00697	0.01554	0.01286
3	0.00000	0.00000	0.01861	0.01408	0.00016	0.00249	0.01421	0.00771	0.01438	0.01237
4	0.00000	0.00000	0.02074	0.01551	0.00089	0.00644	0.01037	0.00969	0.01366	0.01315
5	0.00000	0.00000	0.01170	0.01870	0.00175	0.00722	0.01339	0.01045	0.01482	0.01240
6	0.00000	0.00000	0.01277	0.01286	0.00182	0.00963	0.01099	0.00975	0.01454	0.01318
7	0.00000	0.00000	0.01396	0.01191	0.00019	0.00898	0.01228	0.01062	0.01560	0.01250
8	0.00000	0.00000	0.03266	0.01182	0.00384	0.01325	0.01149	0.01141	0.01519	0.01228
9	0.00000	0.00000	0.02251	0.01414	0.00634	0.00907	0.01372	0.01083	0.01605	0.01243
10	0.00000	0.00000	0.01030	0.01480	0.00922	0.01100	0.01330	0.00874	0.01607	0.01305
J	0.00000	0.06933	0.01485	0.01301	0.00931	0.00679	0.01210	0.00892	0.01624	0.01274
Q	0.00000	0.06933	0.01420	0.02061	0.00825	0.00638	0.01187	0.00984	0.01573	0.01271
K	0.00000	0.00000	0.01544	0.01428	0.00814	0.00378	0.01015	0.00807	0.01603	0.01272
A	0.00000	0.00000	0.01967	0.01442	0.00871	0.00247	0.01386	0.00695	0.01638	0.01215
Hearts										
2	0.00000	0.00000	0.02362	0.02007	0.00000	0.00319	0.01400	0.00719	0.01461	0.01163
3	0.00000	0.00000	0.01647	0.01827	0.00000	0.00428	0.01220	0.00936	0.01459	0.01285
4	0.00000	0.00000	0.01427	0.01515	0.00000	0.00424	0.01153	0.00914	0.01424	0.01246
5	0.00000	0.00000	0.02715	0.01903	0.00076	0.01007	0.01590	0.01091	0.01725	0.01140
6	0.00000	0.00000	0.01605	0.00879	0.00135	0.01066	0.01033	0.01083	0.01434	0.01343
7	0.00000	0.05351	0.01639	0.01117	0.00186	0.01245	0.01192	0.01083	0.01652	0.01288
8	0.00000	0.05351	0.01720	0.01055	0.00261	0.00946	0.01150	0.01135	0.01651	0.01223
9	0.00000	0.00000	0.01241	0.01392	0.00444	0.01011	0.01436	0.00985	0.01656	0.01288
10	0.00000	0.00000	0.00934	0.01367	0.00568	0.01165	0.01077	0.01043	0.01575	0.01258
J	0.00000	0.00000	0.01801	0.01702	0.00802	0.00762	0.01327	0.01137	0.01553	0.01210
Q	0.00000	0.07716	0.01378	0.02242	0.00969	0.00661	0.01062	0.00969	0.01596	0.01264
K	0.00000	0.07716	0.00873	0.01451	0.01223	0.00428	0.00935	0.00729	0.01539	0.01305
A	0.00000	0.00000	0.01757	0.01775	0.00845	0.00187	0.01419	0.00705	0.01615	0.01232
Spades										
2	0.00000	0.00000	0.03369	0.01567	0.00000	0.00164	0.01233	0.00668	0.01368	0.01237
3	0.00000	0.00000	0.02090	0.01297	0.00000	0.00514	0.01149	0.00839	0.01403	0.01268
4	0.00000	0.00000	0.02182	0.01594	0.00000	0.00767	0.01156	0.00936	0.01469	0.01244
5	0.00000	0.00000	0.02130	0.01348	0.00000	0.00855	0.01302	0.01047	0.01528	0.01251
6	0.00000	0.00000	0.01260	0.01267	0.00254	0.01050	0.01339	0.00900	0.01332	0.01247
7	0.00000	0.00000	0.02488	0.01932	0.00139	0.01262	0.01266	0.01032	0.01420	0.01206
8	0.00000	0.00000	0.02380	0.01356	0.00348	0.01251	0.01265	0.01018	0.01475	0.01298
9	0.00000	0.00000	0.01470	0.01358	0.00182	0.01046	0.01018	0.01100	0.01538	0.01257
10	0.00000	0.00000	0.01259	0.01097	0.00686	0.00999	0.00882	0.01050	0.01566	0.01260
J	0.00000	0.00000	0.01198	0.01377	0.01169	0.01077	0.01069	0.01013	0.01669	0.01219
Q	0.00000	0.00000	0.01416	0.01536	0.00972	0.00487	0.01151	0.00936	0.01705	0.01174
K	0.00000	0.00000	0.01967	0.01515	0.01210	0.00392	0.01210	0.00866	0.01616	0.01267
A	0.00000	0.00000	0.01513	0.01329	0.00742	0.00225	0.01105	0.00658	0.01703	0.01207

Reinforcement Learning

In order to find the optimal policy that the agent should follow in his games, we resort to reinforcement learning. More specifically, we applied Q-Learning as shown in class, and in the expression below.

On each state, action, reward and successor state (s, a, r, s'):

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta[\widehat{Q}_{opt}(s, a; \mathbf{w}) - \left(r + \gamma \widehat{V}_{opt}(s')\right)]\phi(s, a)$$

Where $\widehat{V}_{opt}(s') = \max_{a' \in Actions(s')} \widehat{Q}_{opt}(s', a')$. η is the step size and $\gamma = 1$ is the discount factor. For this project, we take $\eta = \frac{1}{\sqrt{k}}$, where k is the iteration count.

Exploration probability in the epsilon-greedy algorithm was set to 0.2 during the learning phase. The main challenge was figuring out the feature extractor that would result in the best learning. We initially started with a features extractor that kept track of the total pot, the table and hand cards, both together and separately, the winning cards, the winning cards value as well as the individual cards remaining in the deck, similar to the blackjack implementation. This resulted in gigantic weight vectors that had redundant information.

Reward Function

Rewards:

$$R(s, a, s') = \begin{cases} pot - total\ agent\ bet, & if\ s'[4] = 4\ and\ agent\ won \\ -total\ agent\ bet, & if\ s'[4] = 4\ and\ agent\ lost \\ 0, & otherwise \end{cases}$$

where s[4] = 4 is the check for the winning condition upon which the result will be declared that whether the bot lost or won.

Nash Equilibrium

Game Theory is a branch of mathematics and economics that is devoted to the analysis of games. This is especially relevant since games can be used as models to many real-world decision problems. Nash Equilibrium is a concept developed from Game Theory.

A Nash Equilibrium is a strategy, for each player, with the property that no single player can do better by changing to a different strategy, meaning that no player has an incentive to deviate from the purposed strategy because the alternatives could possibly lead to a worse result.This implicitly assumes the opposition of perfect players, players that will always do the best possible move, which in real poker, is definitely not the case since players are highly fallible. Nevertheless, a Nash Equilibrium strategy represents a great achievement, especially in two-player zero-sum game, like in heads-up poker. If both players are based on this approach, the expected score for both of them will be zero. On the other hand, if only one player has a Nash Equilibrium approach, he can expect to do no less than to tie the game, since the opponent cannot do better by playing a strategy other than the equilibrium.

The Nash Equilibrium problem for two-player zero-sum sequential games can be modeled using linear programming. Unfortunately, creating a perfect Nash equilibrium strategy for a complex game like Texas Hold,,em is extremely difficult and, at the moment, computationally unfeasible. Instead what is currently used is ε-Nash Equilibrium, an approximation to the Nash equilibrium strategy, resulting in a suboptimal strategy that proposes a best response instead of the perfect response. ε is the value of the best response to the determined suboptimal strategy and is a measure of how far from the actual equilibrium the strategy is. If an opponent, either human or machine, makes mistakes then the equilibrium strategy can win over time.

Artificial poker players based on ε-Nash equilibrium strategy play close to the optimal strategy, making it near-unbeatable by any other strategy in that abstraction. This is particularly useful since it allows defense against optimal/near-optimal opponent strategies and/or safely learning of an opponent tendencies for several hands before attempting to exploit them. Also there are situations where obtaining a quick best response can compensate for the expected cost of computing the perfect response.

RESULTS

Updation of knowledge base upon the execution of an action chosen by the bot

Flop base

```
Fold,Check,Call,Bet-1,Bet-2,Bet-5,Bet-7
-6,0,-32,0,0,0,0
0,0,10,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0

```

River Base

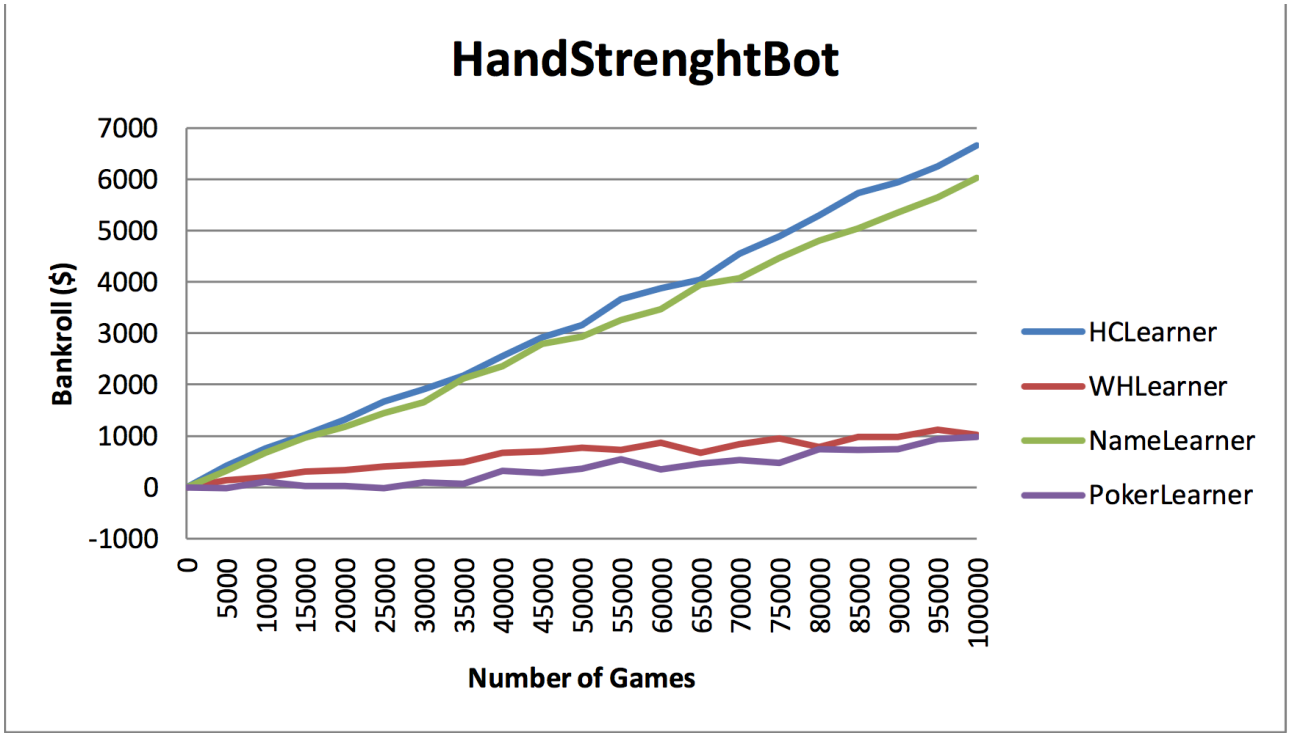
```
Fold,Check,Call,Bet-1,Bet-2,Bet-5,Bet-7
-5,0,-32,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
```

Turn Base

```
Fold,Check,Call,Bet-1,Bet-2,Bet-5,Bet-7
-5,-32,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
0,0,0,0,0,0,0
```

The above knowledge bases are just for a simulation of 15 consecutive games in which the bot won 3 times and lost 11 times. The bot gradually improves its gameplay strategy with the number of games it plays.

ANALYSIS



The graph on the previous page mainly shows the bankrolls i.e a function of the number of wins and the money bet by the bot as a function of the number of games played by the bot.

REFERENCES

- University of Alberta Poker Research Group (poker.cs.ualberta.ca/)
- https://www.pagat.com/poker/rules/ranking.html
- http://doras.dcu.ie/16765/1/NS_thesis.pdf
- http://www-users.cs.umn.edu/~abose/docs/pokerbots_report.pdf
- https://repositorio-aberto.up.pt/bitstream/10216/63296/1/000149512.pdf