Online Learning for Heads-up Poker

Yi-Hsien Lin
yishien@princeton.edu

Akshay Mittal
akshay@princeton.edu

1 Introduction

The problem of how to win in a Texas Hold'em Poker game has always been a popular research topic. Originally, game theory has been the dominant tool in solving poker games. However, as the result of drastically increasing computational power and ballooning interest over the past few years, machine learning has also taken a place in the poker solving domain.

Since we are both enthusiastic poker players, this problem quickly came to our mind while deciding the project topic. After surveying related works that have been conducted in the past, we discovered that due to the gigantic search space, most studies simplified the game a lot, such as restricting *raising* and shortening the game from four bets to one bet only. Clearly, such restrictions are very unrealistic and raised our interest in discovering if it is possible to develop an optimal or near-optimal strategy of an non-handicapped Texas Hold'em Poker game.

In this paper, a very brief overview of the rules of Texas Hold'em and our game setting is first provided. We then demonstrate our developed algorithm for poker playing, based on the Randomized Weighted Majority Algorithm. We also proved that there is a lower bound for the number of non-optimal bets this algorithm would make. Finally, future works and concluding remarks are provided.

2 Overview of Texas Hold'em Poker

Texas Hold'em is a variation of poker, one of the most popular game among other variations. Texas Hold'em poker consists of two cards being dealt face down to each player as hole cards, which is the "pre-flop phase". Then, five community cards are being placed face-up on the table. The community cards come in the order of a series of three cards, also known as "the flop", then an additional card, known as "the turn", and finally the last card, known as "the river". Players have the options of `calling, raising, checking, or folding` on each betting round. After the river card is dealt, each player plays the best poker hand they can make from the seven cards comprising the two hole cards and the five community cards. A player may use both of the two hole cards, only one, or none at all, to form his final five card hand. A player can win the pot when one have the best hand at showdown, or when all other player folds

3 Randomized Weighted Majority Algorithm (RWMA)

3.1 Game Setup

In order to apply online learning algorithms to poker, we first need to acquire many experts. The experts of the randomized weighted majority algorithm are the complete hands of poker of other expert players. When playing heads up poker, player A will follow the recommended bet of the algorithm's output. As for player B, it can either be a human player or a computer with some non-optimal betting strategies. There are many publicly available expert players' game plays [1].

Note that we follow the complete Texas Hold'em Poker game rule, which consists of four betting phases mentioned in the previous chapter. We also allow the player to play all four bets, which are `call(c)`, `raise(r)`, `check(k)` and `fold(f)` respectively. The game starts with two participating players having the exact same amount of money and ends whenever a player's money reaches 0.

3.2 Overview of RWMA

In the algorithm, the experts are predicting whether the learner should call, raise, check or fold and the learner makes a move by choice of the randomized weighted majority among the experts. The observed label y_t (win/loss) is then used to update the weights of all the experts. We keep track of the sum of the weights of all the N experts $W_t = \sum_{i=1}^{N} w_{t,i}$ on each round t. We define $q_{t,move=\{c,r,k,f\}}$ as the sum of the weights of the experts predicting the corresponding move at round t *i.e.*,

$$q_{t,move} = \sum_{i:\xi_{t,i}=move} w_{t,i}$$

Initially let $W_1 = N$, and on each round t, the probability (over the algorithm's randomization) that the algorithm makes a mistake is

$$\begin{aligned} l_t &= Pr[\hat{y}_t \neq y_t] \\ &= \begin{cases} (q_{t,c} + q_{t,r})/W_t & \text{if } y_t = lost \text{ or } 0 \\ (q_{t,k} + q_{t,f})/W_t & \text{if } y_t = won \text{ or } 1 \end{cases} \end{aligned} \tag{1}$$

The intuition behind the above probability is that when the learner loses a round of poker, then all the experts, who made the learner stay in the game by paying additional money, should be penalized and their weights be reduced. This corresponds to the experts that predict `call` or `raise` since the observation is a lost game and the learner loses money. By a similar argument, in the case of a win observation, the weights of the experts, who made the learner leave the game and lost the opportunity to make more money, are penalized, precisely the experts that predict `fold` or `check`.

In order to relate the predictions of the experts (in terms of *moves*) with the observations (result of game) of the learner, we define the *equality* operator $=^*$ as follows

$$\begin{aligned} call &\neq^* lost \\ raise &\neq^* lost \\ fold &\neq^* won \\ check &\neq^* won \end{aligned}$$

Using the above inequalities, we can rewrite Equation 1 as follows

$$l_t = \sum_{i:\xi_{t,i} \neq^* y_t} w_{t,i}/W_t \tag{2}$$

Therefore, we can compute the new sum of weights as follows

$$\begin{aligned} W_{t+1} &= \sum_{i:\xi_{t,i} \neq^* y_t} w_{t+1,i} + \sum_{i:\xi_{t,i} =^* y_t} w_{t+1,i} \\ &= \sum_{i:\xi_{t,i} \neq^* y_t} w_{t,i}\beta + \sum_{i:\xi_{t,i} =^* y_t} w_{t,i} \end{aligned} \tag{3}$$

In Equation 3, it is assumed the experts are penalized with the same weight factor β irrespective of the mis-predictions made on `call`, `raise`, `fold` or `check`. This leads to a bound similar to the randomized weighted majority algorithm *i.e.*,

$$\mathbb{E}[(\#\text{mistakes of learner})] \leq a_\beta \mathbb{E}[(\#\text{mistakes of best expert})] + c_\beta \lg N$$

where $a_\beta = \frac{\ln(1/\beta)}{1-\beta}$ and $c_\beta = \frac{1}{1-\beta}$.

We consider the above bound as a bound for the *naive* approach since it provides same penalty for making mistakes on different moves. We attempt to lose this assumption by modeling the poker experts in a more realistic manner.

3.3 Modeling a Realistic Poker Player

Let $\beta_{move=\{c,r,f,k\}}$ be the penalty factor for the expert making mistake on the corresponding move. When the learner loses a round, then the expert which predicted a `raise` should be penalized more than the expert predicting a `call` since the former caused a greater loss of money. Similarly, when the learner wins a round, then the expert which predicted a `fold` should be penalized more than the expert predicting a `check` since the former caused the learner to quit and lose the opportunity to gain money by staying in the game. Therefore, let σ be the Rademacher variable and let there exist $\epsilon > 0$ such that we let $\beta_c + \sigma\epsilon = \beta_r$ and $\beta_k + \sigma\epsilon = \beta_f$ and let $\beta_r = \beta_f$. Equation 3 can thus be re-framed for the following two cases

Case: $y_t = lost$

$$\begin{aligned}
W_{t+1} &= \sum_{i:\xi_{t,i}=c} w_{t,i}\beta_c + \sum_{i:\xi_{t,i}=r} w_{t,i}\beta_r + \sum_{i:\xi_{t,i}=^*y_t} w_{t,i} \\
&= \beta_c q_{t,c} + \beta_r q_{t,r} + \sum_{i:\xi_{t,i}=^*y_t} w_{t,i} \\
&= (\beta_r - \sigma\epsilon) q_{t,c} + \beta_r q_{t,r} + \sum_{i:\xi_{t,i}=^*y_t} w_{t,i} && (\because \beta_c + \sigma\epsilon = \beta_r) \\
&= \beta_q (q_{t,c} + q_{t,r}) - \sigma\epsilon q_{t,c} + \sum_{i:\xi_{t,i}=^*y_t} w_{t,i} \\
&= \beta_r l_t W_t - \sigma\epsilon q_{t,c} + (W_t - l_t W_t) \\
&= W_t(1 - l_t(1-\beta_r)) - \sigma\epsilon q_{t,c} && (4)
\end{aligned}$$

Case: $y_t = won$

By a similar analysis as for the case above, we get

$$\begin{aligned}
W_{t+1} &= W_t(1 - l_t(1-\beta_f)) - \sigma\epsilon q_{t,k} && (5) \\
&= W_t(1 - l_t(1-\beta_r)) - \sigma\epsilon q_{t,k} && (\because \beta_r = \beta_f \text{ by assumption})
\end{aligned}$$

$q_{t,c}$ is the sum of the weights of the experts predicting the move to be `call` at round t. We can provide an upper bound on $q_{t,c}$ and $q_{t,k}$ by the lowest weight of any expert at the end of T rounds *i.e.*,

$$\beta_c^T \leq q_{t,c} \tag{6}$$

$$\beta_c^T \leq q_{t,k} \tag{7}$$

Thus, using the above bounds, we can re-write Equations 4, 5 as

$$W_{t+1} \leq W_t(1 - l_t(1-\beta_r)) - \sigma\epsilon\beta_c^T \tag{8}$$

Then we can write the sum of the weights at the end of T rounds as

$$\begin{aligned}
W_{final} &\leq \prod_{t=1}^{T}[1 - l_t(1-\beta_r)]W_1 - \sum_{i=1}^{T}\left\{\sigma\epsilon\beta_c^T \prod_{t=1}^{T-i}[1 - l_t(1-\beta_r)]\right\} \\
&= N\prod_{t=1}^{T}[1 - l_t(1-\beta_r)] - \sigma\epsilon\beta_c^T \sum_{i=1}^{T}\prod_{t=1}^{T-i}[1 - l_t(1-\beta_r)] \\
&= N\prod_{t=1}^{T}[1 - l_t(1-\beta_r)] + \sigma\epsilon\beta_c^T \sum_{i=1}^{T}\prod_{t=1}^{T-i}[1 - l_t(1-\beta_r)] \\
& && (\because \sigma \text{ is } \pm 1 \text{ with equal probability})
\end{aligned}$$

$$\leq N\prod_{t=1}^{T}\exp\left(-l_t(1-\beta_r)\right)+\sigma\epsilon\beta_c^T\sum_{i=1}^{T}\prod_{t=1}^{T-i}\exp\left(-l_t(1-\beta_r)\right) \qquad (\because 1+x\leq e^x)$$

$$= N\prod_{t=1}^{T}\exp\left(-l_t(1-\beta_r)\right)-\sigma\epsilon\beta_c^T\sum_{i=1}^{T}\prod_{t=1}^{T-i}\exp\left(-l_t(1-\beta_r)\right)$$

$$(\because \sigma \text{ is } \pm 1 \text{ with equal probability})$$

$$= N\exp\left(-(1-\beta_r)\sum_{t=1}^{T}l_t\right)-\sigma\epsilon\beta_c^T\sum_{i=1}^{T}\exp\left(-(1-\beta_r)\sum_{t=1}^{T-i}l_t\right)$$

By defining $L_A=\sum_{t=1}^{T}l_t$ as the expected number of mistakes the algorithm makes, we have from above

$$W_{final}\leq N\exp\left(-(1-\beta_r)L_A\right)-\sigma\epsilon\beta_c^T\sum_{i=1}^{T}\exp\left(-(1-\beta_r)\sum_{t=1}^{T-i}l_t\right)$$

$$\leq N\exp\left(-(1-\beta_r)L_A\right)-\sigma\epsilon\beta_c^T\sum_{i=1}^{T}\exp\left(-(1-\beta_r)L_A\right) \qquad (\because \sum_{t=1}^{T-i}l_t\leq L_A)$$

$$= N\exp\left(-(1-\beta_r)L_A\right)-\sigma\epsilon\beta_c^T T\exp\left(-(1-\beta_r)L_A\right)$$

$$= (N-\sigma\epsilon\beta_c^T T)\exp\{(-(1-\beta_r)L_A)\} \tag{9}$$

If the expert i makes L_i mistakes, then we have

$$\beta_{move=\{c,r,f,k\}}^{L_i} = w_i$$

$$\leq W_{final}$$

$$\leq (N-\sigma\epsilon\beta_c^T T)\exp\{(-(1-\beta_r)L_A)\} \qquad \text{(from Eq. 9)}$$

The maximum possible weight for expert i can be $\beta_r^{L_i}$ and from the above inequality, we thus get

$$\beta_r^{L_i}\leq (N-\sigma\epsilon\beta_c^T T)\exp\{(-(1-\beta_r)L_A)\}$$

$$\frac{\beta_r^{L_i}}{(N-\sigma\epsilon\beta_c^T T)}\leq \exp\{(-(1-\beta_r)L_A)\}$$

$$\ln\frac{\beta_r^{L_i}}{(N-\sigma\epsilon\beta_c^T T)}\leq -(1-\beta_r)L_A \qquad \text{(assuming } N>T\epsilon\beta_c^T)$$

$$\boxed{L_A\leq\frac{L_i\ln\frac{1}{\beta_r}+\ln N+\ln(1-\gamma_\sigma)}{1-\beta_r}} \tag{10}$$

where $\gamma_\sigma=\sigma\frac{T\epsilon\beta_c^T}{N}$ and $\gamma=\frac{T\epsilon\beta_c^T}{N}$. Note that the above bound holds only when $\gamma<1$. Using the fact that $\beta_r=\beta_c+\sigma\epsilon$ and taking expectation with respect to σ, we get

$$L_A=\mathbb{E}_\sigma[L_A]\leq\mathbb{E}_\sigma\left[\frac{L_i\ln\frac{1}{\beta_r}+\ln N+\ln(1-\gamma_\sigma)}{1-\beta_r}\right]$$

$$= \mathbb{E}_\sigma \left[\frac{L_i \ln \frac{1}{(\beta_c + \sigma\epsilon)} + \ln N + \ln(1 - \gamma_\sigma)}{1 - \beta_c - \sigma\epsilon} \right]$$

$$\boxed{L_A \leq \frac{1}{2} \left[\frac{L_i \ln \frac{1}{(\beta_c + \epsilon)} + \ln N + \ln(1 - \gamma)}{1 - \beta_c - \epsilon} + \frac{L_i \ln \frac{1}{(\beta_c - \epsilon)} + \ln N + \ln(1 + \gamma)}{1 - \beta_c + \epsilon} \right]} \quad (11)$$

Equation 11 thus provides a bound on the expected number of mistakes made by the algorithm.

3.4 Analysis of the Assumptions/Results

Rademacher Variable σ

We introduce the Rademacher variable σ to randomly shift the weights of predicting `call` and `raise`. This follows the intuition from the actual game of poker. Usually raising exposes a player to lose more money than to just call and hence the former should be given more weight. On the other hand, there are game situations when calling would provide greater loss to the player than performing a raise. An example of this situation is when the player is bluffing, she would want raise in order to make the other player to fold, rather than call his bet and enter into the next round of better, thereby increasing the risk of losing more money.

Gamma γ

We define $\gamma = \frac{T\epsilon\beta_c^T}{N}$. The afore-mentioned bound on the expected number of mistakes holds only when $\gamma < 1$. We note that when $\epsilon = 0$ *i.e.*, all the weight factors are the same, then the bound from Equation 11 reduces to the bound as obtained in the *naive* approach. The factor β_c^T goes to zero exponentially compared to the linear factor T in the definition for γ. This implies that when ϵ is very small, then γ is bounded even when T grows. In fact, given a set of N experts, the learner algorithm can determine the optimal values of β_c and the number of rounds T to play the game for, in order to make the least number of mistakes and hence get the highest returns possible.

4 Experimentation

With the purpose of testing our online learning algorithm, we built a heads-up poker simulator which can be played between an human and a computer learning player. We aimed to implement randomized weighted majority algorithm using data from real experts and learning the other players playing pattern on the fly. However, in the interest of time and the complexity of building a successful simulator, we were able to build the game (`2393` LOC) but without the online learning component. We have made the game publicly available, and it can be played from this link `https://github.com/yihsien/H2HPoker`.

5 Related Work

In [2], Quek *et al.* develop a poker A.I. that plays approximately at Nash equilibrium (NE) using an evolutionary algorithm (EA) that employs offline competitive co-evolution as means

of adaptation. Although this work does not directly influence this project study, it is important to realize that besides online learning algorithms, EA algorithms can generate good computer poker players. The model is limited to a much simplified version of poker where there are only three cards and no concept of raising and multiple betting.

In [3], Lingg *et al.* create a Texas Hold'Em poker player using Reinforcement learning and Supervised learning schemes. The player plays optimally in 1 on 1 poker games but being a supervised learner, the computer player needs to play many games with any new opponent in order to determine the latter's playing pattern. In [4], Swanson *et al.* uses game theory to create an optimal poker player for an extremely simplified version of poker.

The number of nodes in a full Texas Hold'Em poker game are $O(10^{18})$ and Billings *et al.* [5] present several abstraction techniques to represent the game using closely related models with size $O(10^7)$. Linear programming solutions are used for building a poker player. Kumar *et al.* [6] use Markov Decision Processes to determine the best move available to the player given the current state of the game. Pfund *et al.* [7] build the player using support vector machines.

We note there is a plethora of research articles aiming to build an autonomous poker player which guarantees positive returns.

6 Conclusion

In this project study, we provide extensive analysis of applying an online learning algorithm to a complete version of Heads-up poker. Under minimal assumptions, we are able to provide a bound for the expected number of mistakes made by the algorithm in terms of the moves (`call`, `raise`, `fold`, `check`) of the game. We have not shown whether the bound is tight but an online learning poker player can decide the parameters - the number of experts and the number of rounds to play the game - in order to achieve positive returns from her investment. We have also provided a full-scale implementation of the Heads-up poker which provides a scalable infrastructure to add any number of computer players, following different strategies.

7 Future Work

We implemented a complete terminal based heads-up Texas Hold'em Poker game. However, this implementation took much longer than the time we initially estimated due to many detailed poker rules and game logic. Due to this reason, we did not have time to incorporate the experts into our algorithm and conduct experiments to verify our theory's correctness. Therefore, our first priority in the future is to finish the implementation and examine if the result aligns with our theory.

Currently, the label of each bet is determined by whether the player won that specific round or not. However, this labeling method is not very representative. For example, there are situations that a player might have a very good hand until the last table card appears. Under this scenario, a correct expert that suggests the player to raise for the first three bets and recommends folding after the last card shows up would be heavily penalized by our

algorithm. Therefore, in the future, we also plan to label each bet by computing the optimal hand according to game theory.

Acknowledgment

We would like to thank Prof. Robert Schapire for his excellent teaching which provided us with deep insight, knowledge and tools that helped us present this project study.

References

[1] Michael Maurer's IRC Poker Database. `http://poker.cs.ualberta.ca/irc_poker_database.html`.

[2] Hanyang Quek, Chunghoong Woo, Kaychen Tan, and Arthur Tay. Evolving nash-optimal poker strategies using evolutionary computation. *Frontiers of Computer Science in China*, 3(1):73–91, 2009.

[3] Machine Learning Applied to Texas Hold'Em Poker. http://cs229.stanford.edu/proj2007/LinggGoSrinivasan-MachineLearningPoker.pdf, 2007.

[4] Jason Swanson. Game theory and poker. *swansonsite. com*, 2005.

[5] Darse Billings, Neil Burch, Aaron Davidson, Robert Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In *IJCAI*, pages 661–668, 2003.

[6] Creating a NL Texas Hold'em Bot. http://cs229.stanford.edu/proj2012/Kumar-CreatingTexasHold'EmBot.pdf, 2012.

[7] Support Vector Machines in the Machine Learning, Classifier for a Texas Hold'em Poker Bot. `http://www.seas.upenn.edu/~cse400/CSE400_2006_2007/Pfund/PfundPaper.pdf`, 2006.