Poker Project - Team 15

Gong Changda, Kong Zijin, Liu Yingnan, Liu Yiyang, Yang Sihan
National University of Singapore
first,second,third,fourth@u.nus.edu

1 Introduction

Poker games have recently become a heated area of Artificial Intelligence research because of its characteristic that the dynamic environment and disclosed community card makes it impossible to directly calculate or predict the final outcome of taking a certain action.

In this project, we design and implement an AI-poker agent playing Limit Texas Hold'em, which is, in this project setting, a two-player zero-sum game with incomplete and imperfect information. The ability to act based on hand strength, interpret the opponent's actions and learn from their strategies is the way towards winning. We work on developing an intelligent agent that would find the optimal strategy and learn from the opponent's strategy.

This report illustrates the development of an intelligent agent using reinforcement learning. Our poker agent is expected to have the ability to design an optimal game strategy based on key information:

- Hand strength estimation
- Community card strength estimation
- Analysis of opponent's strategy
- Current reward evaluation

Our poker agent is supposed to create an optimal game strategy by itself, i.e. unsupervised learning given the information above. The optimal game strategy should include the part that it can analyze the strategy of the opponent and learn from it accordingly.

"Poker games can be defined as a partially observable Markov decision process (POMDP)". As its definition states, the POMDP has no complete and exact solution. Reinforcement learning is only capable of making models for the Markov decision process (MDP). Considering this, what we can do is to select 'significant' factors in a certain state and calculate an expected reward function instead of completely defining the current state and calculating the value function (which we are not capable of due to the natural character of a poker game).

Our overally training strategy can be categorized into two parts: Offline training and Online training. The offline training is used to estimate hand strength and the combined strength of cards in hand (hole card) and community card. The online training is the main part of our reinforcement learning process, which including training weights for every feature we select as well as learning from the opponent's strategy.

2 Past research

To implement an intelligent poker agent, one key point is to decide the player's next action at every game state based on the information obtained from the surrounding environment and its strategy. Utility maximization and regret minimization are two approaches that have been well explored and researched on.

Utility maximation

The most traditional method to optimize the value of the utility function is to enumerate the results for all possible solutions in each round and select the action that gives the highest utility value [Tipton, 2012]. Although this method is precise and implementation friendly, it is excessively computational expensive. To reduce the computational cost, researchers have proposed heuristics to evaluate the backward reward of the game. They also utilized decision trees to model all the possible next-actions and decide what one should be taken [Silva et al., 2018]. With a sufficient and reasonable amount of training, this method can yield good results.

Regret minimization

Regret is defined as the loss of utility that taking a certain action will suffer. The regret is generated when the agent failed to select the single best next-action, which means the player's strategy is not the single best deterministic strategy. And this approach aims to minimize regret value. In 2015, Bowling and other 3 scientists announced that they weakly solved Heads-up limit Texas hold'em by using a technique called counterfactual regret minimization based on regret minimization.

Reinforcement learning

Reinforcement learning is to learn how to map states to actions in order to maximize the numerical value of a reward. The learner in reinforcement does not know which action to take and is supposed to deduce the action that yields the highest reward. According to Sutton and Barto (1998), in the relatively challenging cases, actions affect not only the immediate reward but also the next and even all subsequent rewards. This enlightens the idea of our implementation. This is because in the context of Limit Texas Hold'em, an

action in a middle way does not have an immediate reward, and we need to calculate the expected reward at every step and adjust feature weights according to the difference between it and the final actual reward.

For MDP problems, most commonly-accepted algorithms can be categorized into two categories: direct learning (DL) and indirect learning (IDL). Direct learning includes a value-function based learning such as temporal difference (TD) learning, Q-learning. Our implementation is based on value-function based learning. The heuristic function we use is of <state, action>-tuple structure that estimates the expected reward that player can obtain when executing a certain action. Furthermore, since reinforcement learning cannot solve POMDP problems, we need to adjust our training strategy to make it best fits this Limit Texas Hold'em game.

3 Implementation

3.1 Agent strategy

To play Limit Taxas Hold'em Game, our poker agent evaluates the environment and estimates expected return of actions to make decisions.

1. When it's the poker agent's turn to bet, the poker agent will firstly evaluate the strength of hole cards combining revealed community cards. The hand strength is evaluated based on existing card features in hand and current betting round. The poker agent takes 18 cards features into consideration. Hand strength evaluation function is a linear combination of those 18 card feature's value. While playing, agent uses the function settled by offline training before game to estimate the strength of cards. The function and offline training will be further elaborated in the following section.
2. After evaluating hand strength, agent will estimates the expected reward of the whole round of game if choosing each action {'call', 'raise' or 'fold}. The expected reward is estimated by evaluation function E, which is a linear combination of hand strength, the difference between the current bet and the opponent's bet, the difference between the current remaining money and the money in the opponent's stack, and the difference between my gain and the opponent's gain:

$$E_{action} = E \cdot V_{action,street}$$

where action $= raise, call, fold$, street $= 1, 2, 3, 4$.
The expression of E is $E =< 1, h, c, s, g >$. 1 is a constant padded for the bias,h is the hand strength, c is the bet difference, s is the remaining stack difference and g is the gain difference.
The expression of V is $V_i =< v_b, v_h, v_c, v_o >$. v_h is the weight of h, representing how influential hand strength is to quality of action.
3. Using the evaluation function above, agent gets expected payoff for each action E_{raise}, E_{call}, E_{fold}. If agent has not raised for four times, it will choose the action with highest expected E among raise, call and fold. If agent has already raised for four times, it will choose the action with higher expected E among call and fold.
To make the evaluation function estimate expected payoff more precisely, our agent adjusts vectors for actions based on the actual payoff it gets after finishing the whole round of game.

3.2 Hand Strength Evaluation Function

Generally speaking, the hand strength of the combination of hole cards and revealed community card is static. The evaluation function of hand strength is obtained by offline training and settled before the start of game. It is not affected by opponents' behavior and will not be adjusted in the process of playing game. Hand strength evaluation function is a linear combination of all features appearing in the combination of hole cards and revealed community card(s). There are 18 features that our agent use to estimate hand strength.
All the 18 card features of 4 types are shown below.

	Card feature	Value	We
hole card feature	bigger value among two hole cards	v_1	w
	difference between value of two cards	v_2	w
cards of one type	pair	v_3	w
	number of pairs	v_4	w
	three of a kind	v_5	w
	number of "three of a kind"	v_6	w
	four of a hand	v_7	w
	full house	v_8	w
suit	two card of one suit	v_9	w
	number of "two card of one suit"	v_{10}	w
	three card of one suit	v_{11}	w
	number of "three card of one suit"	v_{12}	w
	four card of one suit	v_{13}	w
	flush	v_{14}	w
sequence	two consecutive cards	v_{15}	w
	three consecutive cards	v_{16}	w
	four consecutive cards	v_{17}	w
	straight	v_{18}	w

In the table above, fi represents the value of card feature i. f_1 and f_2 are scaled to a number between 0 and 1. f_3, f_4, ..., f_{18} are 1 when the feature appears in the combination of hole cards and revealed community card. They are 0 when the feature doesn't appear. v_i is the weight of f_i, representing how influential card feature i is to hand strength.
The hand strength evaluation function H is a linear combination of those 18 card feature's value:

$$F =< f_1, f_2, f_3, ..., f_{18} > V_i =< v_1, v_2, v_3, ..., v_{18} >$$

$$H(F, V_i) = F \cdot V_i = \sum_{1}^{18} fivi$$

$i = 1, 2, 3, 4$, representing the current street.
Vectors $V_i, (i = 1, 2, 3, 4)$ for the four betting streets are obtained by offline training. Before the training, Weight vector V_i are all set to $< 1, 1, \ldots, 1 >$. The agent is trained by playing with (random) player.
The training process is shown below:

1. At the beginning of each betting round, the agent finds all card features that appears in the combination of hole cards and community card revealed in this round. Value of card features $f_1, \dots f_{18}$ are set here. Then, the hand strength is calculated by the dot product of V_i and F.
2. Hand strength, bet has put, money in stack and other information describe the current game state will be combined into a feature vector E. The estimated final reward of choosing the action a in street s under the condition of the game state being E is the dot product of $E \cdot V_{a,s}$. For a in `{'raise', 'call', 'fold'}`, we compare their estimated final reward and choose the one with the highest possible reward.
3. At the end of the round, we trace back the action history. For every action a in the history at time t, we compare its estimated reward and the final true reward. The weight is updated by adding $\Delta V_{a,s} = \mu P(R_{true} - R_{estimated})E_{a,s}$, where μ is the learning factor, P is the probability of reaching the final state from choosing a at time t.

3.3 Evaluation Function for expected return of actions

To enable our agent to learn from games that it have been played with the opponent, the weight for each feature in the evaluation function is adjusted to improve the accuracy of our evaluation. As mentioned in section 3.2, our evaluation function for each action calculate the expected reward of taking the action. Based on the result at the end of every round, the value of the weight vector, θ will be adjusted to reduce the difference between expected reward and true reward.

At the very beginning when the is not input information from the game, hard-coded base weight is used in the evaluation function. The base weight of each feature was derived by playing against itself. (how the value is derived)

If the agent play with the same opponent for subsequent rounds, θ will be updated at the end of each round inside `receive_round_result_message()` function. Stochastic gradient descent method is adopted to update θ. The mathematic expression is

$$\theta \leftarrow \theta - \frac{\alpha}{2}\nabla_\theta L$$

where α denotes the learning rate and L denotes the loss function. The smaller the value of α is, the slower the learning process takes place. A relatively small learning rate (value) is chosen for the agent, to ensure that the agent is learning while prevent it from learning too fast and not converge on the true reward. We defined a loss function L that shows the difference between expected reward E and actual reward R. The expression of loss function is $L = (R - E)^2$. (is the difference always larger than 1?)Note that instead of calculating the absolute value of the difference between expected reward and true reward, we take the square of the difference so that it will be more significant and the adjustment will be more sensitive to even minor differences. Since the true reward at each round is achieved by a series of actions, each action contributes a fraction of the prediction, a scaling factor s was added to plot the fraction of the expected reward that is contributed by a single action. (the value of s ?) Thus, the mathematic expression we used to adjust the value of θ for each action is

$$\theta \leftarrow \theta - s\frac{\alpha}{2}\nabla_\theta L$$

As mentioned in section 3.2, $E = Q(\hat{\phi}; \theta)$, substitute E inside the expression and simply it, we get

$$\theta \leftarrow \theta - s\frac{\alpha}{2}\nabla_\theta (R - Q(\hat{\phi}; \theta))^2$$

$$\theta \leftarrow \theta - s\frac{\alpha}{2}(-2)(R - Q(\hat{\phi}; \theta))\frac{d}{d\theta}Q(\phi; \theta)$$

$$\theta \leftarrow \theta + s\alpha(R - \hat{Q}(\phi))\phi$$

This is the equation we used to update the value of θ.
To end up with a good estimate of e

4 Training Result and analysis

4.1 Offline training result

4.2 Online training result

4.3 discussion

Our implementation has three main advantages. Firstly, the approach we used to solve this problem is intuitive and fits the characteristic of poker game. Instead of generating game tree and running minimax algorithm to solve the problem, we use decision tree. The restriction of minimax algorithm in imperfect information game is mentioned by several researchers. Decision tree, in the other hand, requires less space and simulate well people's behavior in real life. Secondly, we took a wide range of features into consideration which makes our evaluation more complete and precise. Thirdly, by using reinforcement learning, we adjust the weight of each feature promptly.

5 Conclusion

References

[Abelson *et al.*, 1985] Harold Abelson, Gerald Jay Sussman, and Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, Massachusetts, 1985.

[Baumgartner *et al.*, 2001] Robert Baumgartner, Georg Gottlob, and Sergio Flesca. Visual information extraction with Lixto. In *Proceedings of the 27th International Conference on Very Large Databases*, pages 119–128, Rome, Italy, September 2001. Morgan Kaufmann.

[Brachman and Schmolze, 1985] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, April–June 1985.

[Gottlob *et al.*, 2002] Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *Journal of Computer and System Sciences*, 64(3):579–627, May 2002.

[Gottlob, 1992] Georg Gottlob. Complexity results for nonmonotonic logics. *Journal of Logic and Computation*, 2(3):397–425, June 1992.

[Levesque, 1984a] Hector J. Levesque. Foundations of a functional approach to knowledge representation. *Artificial Intelligence*, 23(2):155–212, July 1984.

[Levesque, 1984b] Hector J. Levesque. A logic of implicit and explicit belief. In *Proceedings of the Fourth National Conference on Artificial Intelligence*, pages 198–202, Austin, Texas, August 1984. American Association for Artificial Intelligence.

[Nebel, 2000] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research*, 12:271–315, 2000.