FAI Final Project Report

B09901142 EE4 呂睿超

June 2024

Method 1 - Q-learning

Introduction Q-learning is a model-free reinforcement learning algorithm that aims to learn the value of an action in a particular state. It uses a Q-table to store and update the values of state-action pairs, guiding the agent to take actions that maximize the cumulative reward.

Discussion I didn't try this for long, but briefly discussing about it is that I found it struggles with large state-action spaces due to the curse of dimensionality. Also, it requires a significant amount of exploration and time to converge. After some hyperparameter tuning, it stills stucks to consisting declaring folding.

Method 2 - PPO

Introduction Proximal Policy Optimization (PPO) is a reinforcement learning algorithm that improves upon policy gradient methods. PPO uses a clipped objective function to maintain a balance between exploration and exploitation, ensuring stable and efficient training.

Discussion I've tried multiple tuning and attempts on this method, including

1. Deeper policy network
2. Network also aim to learn the amount to raise
3. Utilize the HandEvaluator to get card strength of our current state as part of our state information.

Another part of difficulty is about training and determining the encoded state information

1. I can determine how to train the model since I can only utilize the baselines. But if I sequentially train the model on each baseline, I can't guarantee that it could do well on all baselines after the sequential training.
2. Some information that I encoded:
   (a) hold cards, community cards(Using the Card Class functions in the game engine environment)

(b) hand card strength evaluation

(c) current pot size

(d) current stack

But after all the attempts, I found it hard to very hard to tune and fit the nature since I don't know what information is suitable to be encoded as state information. Also, I found it outperforming

Method 3 - MCCFR

Introduction Monte Carlo Counterfactual Regret Minimization (MCCFR) is an algorithm used specifically for large imperfect information games. MCCFR uses sampling methods to estimate the counterfactual regret of actions and iteratively minimizes regret to converge to an equilibrium strategy.

Discussion I found this method when I'm doing research about implementations of Texas Hold'em AI agents on the Internet. The core idea of CFR is to minimize regret for decisions not taken by considering "what if" scenarios, referred to as counterfactuals. Regret is calculated as the difference between the payoff received and the best possible payoff that could have been received if a different action had been taken. Also, the conecept of averaging all the strategies achieves Nash Equilibrium is a cool discovery to learn for me.

I failed to implement this method correctly because I failed to find a simulation function that can correctly take an action and return a simulated next_state for me.

Method 4 - Monte Carlo Simulation

Introduction Monte Carlo Simulation (MCS) is a statistical method used to estimate the probability of different outcomes by running a large number of simulations. In the context of Texas Hold'em, MCS can be used to evaluate hand strength and potential outcomes based on random sampling.

Detail Lastly, I chose this method because it is rather easy to implement and I only succeeded on it.

1. The main concept of MCS is to run lots of experiements to evaluate the confidence of current state.
2. I determine different action and amount based on the following information
   (a) the opponent's action: When the opponent declares 'call', I would rather raise a little smaller to decrease risk.
   (b) the estimated win rate: I have three thresholds for all in, raising but depend on current pot and current stack, and call.
   (c) opponent modeling: I implemented a method to observe the opponents action for several rounds and determine whether the model is **'aggressive'**(prone to declare 'raise'), **'neutral'** or **'passive'**(prone to declare 'fold'). Depending on the opponent type, I would slightly shift the thresholds to fit the type to attack them.

(d) Specifically, if the opponent declares 'allin', I would shift the call threshold a lot to ensure that I am sure that I can beat the opponent.

3. Another observation is that the number of simulations affects the performance and also affects the tuning of the thresholds.

Configuration

Specifically, the hyperparameters that I select for my agent is as the following:

1. number of simulations: 3000,
2. thresholds for raising more, raising less, calling: [0.75,0.65,0.55]
3. opponent modeling threshold for determining opponent type:
   (a) observing rounds(actions): 8
   (b) aggressive models: over 50% 'raise'
   (c) passive models: over 40% 'fold'
4. Other detail configurations can be found in the code implementation

Final method

As I wrote above, the final method that I submitted is Monte-Carlo Simulation

Reference

1. lecture slides,
2. ChatGPT
3. Learn AI Game Playing Algorithm Part III — Counterfactual Regret Minimization