Report

1. Counterfactual Regret Minimization

The first method I have tried is the CFR(Counterfactual Regret Minimization) algorithm. The algorithm iteratively approaches the Nash equilibrium with the regret matching and counterfactual value. And I used the following pesudo code to implement this algorithm[1]. However, after 100000 iterations

Algorithm 1 CFR Algorithm

```
1: Initialize cumulative regret tables: $\forall I, r_I[a] \leftarrow 0$
2: Initialize cumulative strategy tables: $\forall I, s_I[a] \leftarrow 0$
3: Initialize initial profile: $\sigma^1(I,a) \leftarrow 1/|A[I]|$
4: function CFR($h, i, t, \pi_1, \pi_2$)
5:     if h is terminal then
6:         return $u_i(h)$
7:     else if h is a chance node then
8:         Sample a single outcome $a \sim \sigma_c(h,\ a)$
9:         return CFR($ha, i, t, \pi_1, \pi_2$)
10:    end if
11:    Let $I$ be the information set containing $h$.
12:    $v_\sigma \leftarrow 0$
13:    $v_{\sigma_{I\rightarrow a}}[a] \leftarrow 0$ for all $a \in A(I)$
14:    for $a \in A(I)$ do
15:        if $P(h) = 1$ then
16:            $v_{\sigma_{I\rightarrow a}}[a] \leftarrow$ CFR($ha, i, t, \sigma^t(I,a) \cdot \pi_1, \pi_2$)
17:        else if $P(h) = 2$ then
18:            $v_{\sigma_{I\rightarrow a}}[a] \leftarrow$ CFR($ha, i, t, \pi_1, \sigma^t(I,a) \cdot \pi_2$)
19:        end if
20:        $v_\sigma \leftarrow v_\sigma + \sigma^t(I,a) \cdot v_{\sigma_{I\rightarrow a}}[a]$
21:    end for
22:    if $P(h) = i$ then
23:        for $a \in A(I)$ do
24:            $r_I[a] \leftarrow r_I[a] + \pi_{-i} \cdot (v_{\sigma_{I\rightarrow a}}[a] - v_\sigma)$
25:            $s_I[a] \leftarrow s_I[a] + \pi_i \cdot \sigma^t(I,a)$
26:        end for
27:        $\sigma^{t+1}(I) \leftarrow$ regret-matching value
28:    end if
29:    return $v_\sigma$
30: end function
31: function SOVLE( )
32:    for $t = \{1, 2, 3, ..., T\}$ do
33:        for $i \in \{1, 2\}$ do
34:            CFR($\emptyset, i, t, 1, 1$)
35:        end for
36:    end for
37: end function
```

of training, I found this method is extremely time-consuming and space-consuming. Because I only collected about 1.2M information set after over 2 hours training process but the number of information set in No-Limit Texas Hold'em is up to 10^{162}[2], which is nearly impossible for me to collect and store. And I think it's also the reason why many paper about poker use Leduc Hold'em to demostrate how power their algorithm is because the number of information set in Leduc Hold'em is only about 100[2]. As a result, I have to abandon this method although it's well-recognized the best algorithm so far when it comes to Texas Hold'em.

2. Monte Carlo method

After the attempt above, I decided to use the basic Monte Carlo method[3] and some domain knowledge to implement a simple but still powerful enough agent to defeat public baseline ai first. The Monte Carlo method is to randomly sample a large number of possible game results to approximate the real win rate. And I use the proximate win rate and opponent's action to make decision. The hyperparemeter in this agent is the number of times in simulation, and I choose 6000 because the more times we stimulate, the more precise win rate we can get (couldn't be more due to 5 seconds limitaion). After carefully finetuning, this agent can defeat baseline 0, 1, 2, 3, 5 and it have about 70%

Algorithm 2 MCM Algorithm

```
1:  function MCM($s, holeCard, communityCard$)
2:      initialize $w : w \leftarrow 0$
3:      for $i = \{1, 2, 3...s\}$ do
4:          $w \leftarrow w +$ monteCarloSimulation($holeCard, communityCard$)
5:      end for
6:      return $w/s$
7:  end function
8:  function monteCarloSimulation($holeCard, communityCard$)
9:      Initialize opponent's hold cards: $oppCard \leftarrow$ fillCard($holeCard$)
10:     Initialize community cards: $communityCard \leftarrow$ fillCard($holeCard + oppCard +$ communityCard)
11:     myScore $\leftarrow$ evalHand($holeCard, communityCard$)
12:     oppScore $\leftarrow$ evalHand($oppCard, communityCard$)
13:     if myScore $>$ oppScore then
14:         return 1
15:     end if
16:     return 0
17: end function
```

win rates against baseline 4 in 20 rounds. However, if the agents compete in 200 rounds, this agent can always win. In my opinion, it's because the factor of luck will be diminished when the length of competition becomes longer.

3. Deep Q learning

The third method I have tried is deep Q learning(DQN)[4]. This reinforcement learning method is to use Q function as critic and we use deep learning and gradient descent to learn and optimize model(Q function). This model takes current state as input and the score of each possible action as output. The loss function I use is $MSE(Q(s_t, a_t), r_t + max_a Q(s_{t+1}, a))(Q(s_t, a_t)$ is the score of a_t that Q function predict under s_t.)

Moerover, I use replay buffer and epsilon greedy to further optimize the performance of DQN model. Replay buffer is to collect the data we collected from the interaction between model and environment in the past and when gradient descent, randomly sample the data in the replay buffer can help increase the diversity of training data and decrease the time spent in collecting data. Epsilon greedy is to let model attempt more possible action in early training stage, which is useful to converge the model.

Last but not the least, the reward function I use is $R_t = \gamma^{(T-t)} r_t$, r_t is the stack we get or loss in a single round, t is timestep the corresponding action made in, R_t is the reward of the corresponding action and γ is the decay rate.

I have tried three kinds of different state. The first one is to use 52 binary input to represent our hole card and community card. However, with this state definition, the model is very hard to converge(the reason may be the rule of poker is to complex for model to learn by itself), so I decided to make use of the Monte Carlo method. The second state is to use the win rate predicted by Monte Carlo method. With sufficient training, this model can defeat baseline0 but still hard to compete with baseline4 and baseline5. The third state I use is combine win rate and our and opponent's last two actions. Nevertheless, the model's performance against baseline4 and baseline5 becomes worse than the last one. In my opinion, maybe is's because the state is to complex for the model to converge.

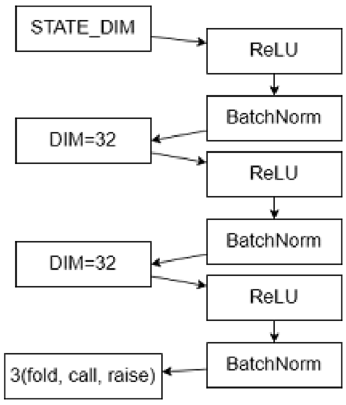

Figure 1: The architecture of model

Interestingly, I found all of these three model's strategy are very conservative. If the win rate is too low, they will choose to fold instantly. Even if the win rate is high enough to raise, they still choose to just call in order to prevent losing too much money.

4. Conclusion

Drawing a comparison among these three method I have tried, the CFR algorithm is too time and spaec-consuming, the DQN is too hard to converge to an acceptable performance; hence, even though the Monte Carlo method + domain knowledge seems like too simple for Texas Hold'em, I still choose this one as the final submission.

Reference

[1] Todd W. Neller, Marc Lanctot, An Introduction to Counterfactual Regret Minimization
[2] RL Card, available-environments
[3] PyPokerEngine
[4] Hung-yi Lee, DRL Lecture 3: Q-learning (Basic Idea)