Cheat'em: A Value & Rival-Aware Algorithm on Limited Texas Hold'em

Wenzhuo Zheng [*1] **Jinghao Feng** [*1] **Mingyang Jiang** [*1]

Abstract

Limited Texas Hold'em is a world-wide popular card game, and also a typical imperfect information multi-agent game. It's a compound of fortune and strategy. This project studies application of MARL algorithms on this game. We first study performance of existing RL algorithms, and propose some customized modifications based on experimental observations. Then, we integrate our ideas to propose a brand new value-aware and rival-aware MARL algorithm that outperforms all baselines. Our project is available at https://github.com/Raingate/Texas-Cheat-Em.

1. Introduction

Limited Texas Hold'em is a classical multi-player card game. In each game, players are first dealt 2 hand cards, and then 5 community cards are dealt face up in 3 stages. Each player seeks the best five cards from combinations of all 7 cards. In every round, players have some betting options including call, raise, fold and check. The player who has the best hand and hasn't fold by the end of the betting round wins all chips. It's a typical imperfect information game, a good object of Multi-Agent Reinforcement Learning (MARL). In this project, we focus on heads-up version of the game, where only two players are involved.

In this project, We first construct a Proximal Policy Optimization (PPO) agent, and carry out tournament between PPO and all baselines. Experiments shows that PPO gives best response against all baselines. However, confronted with some simple rule-based agents defined by ourselves, PPO is easily defeated. After careful examining and analyzing, we conclude that the agents merely learn some local equilibrium, but doesn't really 'understand' the game: only the policy model is trained, while the value part remains in chaos.

To address the problem above, we first proposed reward allocation, as well as opportunity cost to improve supervision. However, due to the lack of modeling accuracy, the modifications above showed no improvement. Meanwhile, we found CFR (Counterfactual Regret Minimization)(Zinkevich et al., 2007) an ideal estimation of value model, however time-consuming that requires days to train. To reduce time cost while inherit the main idea of CFR, we propose a method that pretrains a Cheating Sheet (CS) matrix to attach estimated values to merged state and action pairs, where states are merged according to card type.

With the help of CS combined with policy distribution directly sampled from it, the new agent (called CS Agent) successfully outperforms rule-based agents. To model dynamic evolution of the environment and other agents, we further propose Advantage Probability (AP), which estimates the agent's hand advantage over the rivals'. Finally, a MLP policy modification network is applied to aggregate AP and CS and gives the real-time policy.

The content below is organized as follows: 2 introduces some previous works that inspired our framework, 3 introduces the evolution of our framework.4 introduces some details of the proposed model. Finally, 5 shows experimental results in our project.

2. Related Work

2.1. Proximal Policy Optimization (PPO)

PPO (Schulman et al., 2017) is a family of policy gradient methods for reinforcement learning. Different from classical policy gradient methods, PPO maintains an multiple-minibatch-aware objective function. PPO uses self-adapting KL punishment coefficient as the actor's loss, while common TD-Error is the critic's loss. PPO is comsidered a simpler and more general version of Trust Region Policy Optimization (TRPO)(Schulman et al., 2015), but retains TRPO's main idea: accurately estimating updating step-length. PPO performs well on simulated robotic locomotion and Atari game playing, and is a popular RL agent

[*]Equal contribution [1]School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University. Correspondence to: Wenzhuo Zheng <519030910360>, Jinghao Feng <519010910362>, Mingyang Jiang <519010910384>.

nowadays.

2.2. RLCard

RLCard (Zha et al., 2020) is a toolkit for RL in card games. It provides multiple card game environments with well-defined interfaces. Besides, RLCard contains multiple baseline agents based on some popular RL models (e.g. DQN, Rule-based, CFR and Deep Monte Carlo), some of which are also the baselines of our project.

3. Methodology

This part introduces the evolution process of our framework. In chronological order, we proposed:

- Reward Shaping, including Reward Allocation and Opportunity Cost
- Pre-trained Cheating Sheet
- Advantage Probability and Policy Modification Network

Algorithm 1 Reward Allocation

```
Input: Trajectories T of len m
Get each trajectory t from T, included state, action, reward
Determine winner and loser
for $i = 0$ to $m - 1$ do
  if playerid == loser then
    if is_last_state then
      reward_loser[i] -= sum(reward_loser[0:i-1])
    else if action == raise then
      reward_loser[i]=-(max(chips)-mychips+big_blind)
    else if action[i] == call then
      reward_loser[i]=-big_blind
    else if action[i] == check then
      reward_loser[i]=0
    end if
  else if playerid == winner then
    reward_winner[i] = -reward_loser[next_state]
  end if
end for
```

3.1. Reward Shaping

At the very beginning of this project, we constructed PPO agent to supplement RLCard's baseline library, then carried out on-line tournament between all baselines (PPO, DQN, Random, Rule-based). Analyzing the experimental results, we observed that both DRL agents (PPO and DQN) tend to become single-actional, where PPO only raises, while DQN only calls. We attribute this phenomenon to the highly imperfect information structure of Hold'em itself. In each game, agents may take many rounds of actions, but reward is given only at the end of the game, which leads to vague supervision of particular decisions at particular states. For example, a player certainly gets negative reward if it folds, but we all know that when hand cards are bad, we should fold as soon as possible to prevent further loss. More importantly, under vague supervision, agents learn nothing about the quality of states, namely how good their hand cards are.

To address this problem, we first proposed **Reward Allocation** strategy to manually allocate the total reward to each decision. Specifically, in each record of one game, for the loser, each decision is punished by the chips given by this action, while for the winner, each decision is credited by the chips given by the rival in the next step. Specific definition is described in alg1.

Meanwhile, we used for reference the concept of **Opportunity Cost** in Economics and proposed our method. For each game record, we first assess each agent's best 5 cards with a value defined by ourselves. The value is positively correlated to the superiority of card type (e.g. straight flush has the highest value, four of a kind follows, and high card ranks last). The difference between the agent's value and the rival's is referred to as opportunity cost (OC). We minus the final reward by OC to encourage the agent to act based on hand quality.

Unfortunately, due to high parameter sensitivity and the lack of effective evaluation criteria, the above two shaping strategies shows no apparent improvement. This is when Counterfactual Regret Minimization (CFR) (Zinkevich et al., 2007) came into our mind. CFR models state and action value at an adequately high accuracy, as it applies direct deep search to approximate the total state space.

3.2. Pre-trained Cheating Sheet

To better bridge the gap between state value and policy, it would be a good idea to pretrain a CFR model to provide supervision, compelling the agents to understand their own states. However, the time cost is unacceptable (in fact, it would take days to train CFR even on cluster with hundreds of CPUs). To find a balance between modeling accuracy and time cost, we propose Cheating Sheet, which inherits CFR's basic idea, but with highly compressed state space.

We first merge all possible card combinations into 45 categories according to the original card type superiority (an example of card type merging is shown in Tab.1), then construct a 45×4 matrix, referred to as Cheating Sheet (CS), where row represents merged card type, column represents 4 actions. $CS[i, j]$ represents the value of action j at card

Table 1. Card type encoder. The number in bracket denotes the number of cards turned over.

TYPE ID	DESCRIPTION	PROBABILITY(%)
0(5)	STRAIGHT FLUSH, FOUR, FULL HOUSE, FLUSH, STRAIGHT, THREE	2.87
1(5)	TWO PAIRS	4.75
$2 \sim 14(5)$	ONE PAIR	42.26
$15 \sim 19(5)$	HIGH CARD	50.12
$20 \sim 32(2)$	PAIR	5.9
$33 \sim 44(2)$	HIGH CARD	94.1

type i. In the sense of expectation, we expect that the sum of CS values of all state-action pairs of an episode equals to the total reward. Specific training procedure is introduced in 4.1. To better model the dynamic structure of the environment and the rival, we further propose online updating the cheating sheet in practice. The updating scheme is the same as pre-training.

To verify the validity of CS, we designed a simple agent, referred to as TableAgent, who responds to state i with action a randomly sampled from $Softmax(CS[i])$. In experiments, TableAgent successfully defeated PPO, DQN and other baselines.

3.3. Advantage Probability and Policy Modification Network

As mentioned above, TableAgent outperforms all baselines with well-defined value modeling, but a clever agent should also consider the dynamically changing environment, especially the 'personality' of its rival. To estimate the advantage of the agent's card type over the rival's, We propose Advantage Probability based on Bayes formula derivations:

$$\begin{aligned} &P(H > H_r | A_r, H = h) \\ &= \frac{P(A_r, H = h | H > H_r) P(H > H_r)}{P(A_r, H = h)} \\ &= \frac{P(A_r, H = h | H > H_r) P(H > H_r)}{P(A_r) P(H = h)} \end{aligned} \quad (1)$$

As a fair game, $P(H > H_r) = P(H < H_r) = 0.5$. And we assume that the rival's action is independent of the agent's hand cards. eq.1 can be further splited into

$$\begin{aligned} &\frac{P(A_r | H > H_r) P(H | H > H_r) P(H > H_r)}{P(A_r) P(H = h)} \\ &= \frac{P(A_r | H > H_r) P(H > H_r | H)}{P(A_r)} \end{aligned} \quad (2)$$

Here H and H_r denotes the agent's and the rival's hand cards respectively. A_r denotes the rival's latest action, R denotes raise. As we can see, there are 3 terms in AP: $P(a2 = R | H > H_r)$ is models the rival's 'courage' to raise with an estimated smaller hand, and can be estimated statistically. $P(H > H_r | H)$ can be estimated by Monte Carlo sampling. $P(A_r)$ is also a statistical term.

In conclusion, with AP, we model the hand card advantage along with the rival's action style. Though rough, the scalar brings dynamic properties to our model.

with CS and AP, we now construct the final model of our project: Cheat'em. The model takes actions based on CS, but also receives real-time modification from AP. We further apply a MLP policy modification network to better utilize the information provided by AP. For state-action pair (s, a) the network takes scalar AP as input, and is supervised by a proper mixture of CS values and total reward. Specific definition is introduced in 4.2.

Algorithm 2 Cheating Sheet Update

```
Input: Card type encoder $\Phi$, number of card type $C$,number of action $A$, number of episodes to play $E$, learning rate $lr$.
Initialize Cheating sheet matrix $M_{C \times A}$ .
for $i = 1$ to $E$ do
    Sample a trajectory $T_{0\sim n} = (ts_0, ts_1 ..., ts_n)$, where $ts_i = (s_i, a_i, r_i, s_{1i}, done_i)$.
    Randomly choose time $t$ from $0 \sim n$.
    Get the reward of the game $R = r_n$.
    Compute target value:
        $\hat{V} = R - \sum_{i \neq t} M[\Phi(s_i)][a_i]$
    Update value $M[\Phi(s_t)][a_t] \triangleq V$:
        $M[\Phi(s_t)][a_t] = V + lr * (\hat{V} - V)$
end for
```

4. Model Design

4.1. Pre-train Procedure of Cheating Sheet

To pre-train CS, we conducted 1,000,000 episodes between 2 carefully designed random agents.

The original random agent provided by RLCard uniformly takes legal actions, but a large proportion of the games would end up with folding. In order to sample adequate game records with rich diversity, we modify the action distribution of the original random agent by reducing the probability of fold.

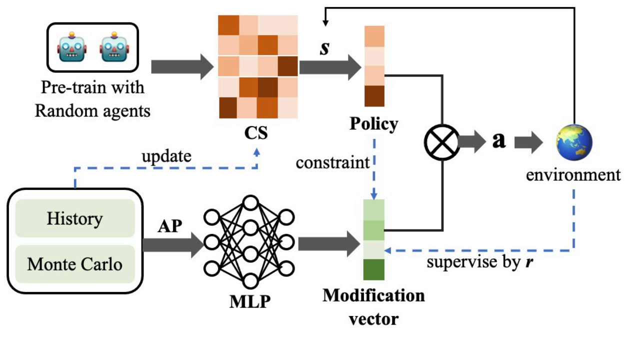

Figure 1. Complete structure of Cheat'em.

In each update stage, we randomly choose a step (s, a) from a game record, and update $CS[s, a]$ with the knowledge of the whole record. In experiments, we observe that when card type is good, CS matrix credits call and raise with high values, while punishes fold. When card type is bad, CS gives negative values to all actions, but fold always has highest values. This is consistent with our intuition. By the application of CS, we equip our model with prior knowledge of state value. The specific training procedure of CS is shown in alg 2.

4.2. Policy Modification Network

Given each (s, a) in training, we first calculate AP by statistics and Monte Carlo sampling, and feed it to the policy modification network, which is a MLP with multiple FC layers. The network outputs a policy distribution, which is supervised by 2 loss terms:

$$\begin{aligned} \mathcal{L}_1 &= \Big|\mathcal{S}(C_s) - \mathcal{D}(p, C_s)\Big| \\ \mathcal{L}_2 &= -\lambda_1 R, \quad R = \mathcal{D}(p, C_s) \cdot C_s + \lambda_2 \cdot r \cdot p(a) \end{aligned} \tag{3}$$

$$\mathcal{D}(p, C_s) = \frac{\mathbf{MLP}(p) \cdot \mathcal{S}(C_s)}{\sum \mathbf{MLP}(p) \cdot \mathcal{S}(C_s)} \tag{4}$$

where $\mathcal{S}$ denotes Softmax, C_s is the row of CS indexed by state s, MLP denotes the policy modification network (with Softmax), p denotes advantage probability AP, λ_1 and λ_2 are weight coefficients, r denotes the total reward of this game, and $p(a)$ the probability of a given by policy modification network.

Here we actually generate the policy mainly from CS, but dynamically modify the policy based on the output of MLP. Diving deeper, $\mathcal{L}_1$ expects that the modified distribution remains close to the original one. In other words, $\mathcal{L}_1$ restricts the scale of dynamic modification. Meanwhile, $\mathcal{L}_2$ expects to maximize the expectational value of policy, while also considers the environmental reward and dynamically encourages or discourages the probability of the current action a.

A brief sketch of our model is shown in fig 1.

Algorithm 3 Training Procedure

```
Input: Number of episode to play $E$, Rival agent $A_r$.
Initialize Agent $A$ with pre-trained $CS$.
for $i = 1$ to $E$ do
    1. Compute AP using eq.2 and get strategy distribution $D(p, C_s)$ using eq.4.
    2. Play an episode with $A_r$ using strategy $D(p, C_s)$, feed $A$ with an trajectory $T = (t_{s0}, t_{s1}, ..., t_{sn})$.
    3. Update the estimation of rival $A_r$'s action distribution $P(A_r|H > H_r)$ and $P(A_r)$.
    4. Update the network of $A$ with loss in eq.3.
    5. Update the $CS$ of $A$ using Alg.2.
end for
```

4.3. Training Procedure

To calculate AP in training, we maintain two history recorders. One is the rival's action history, the other is the rival's action history conditioned on that the agent's hand cards is better than the rival. The two recorders are used to calculate $P(A_r)$ and $P(A_r|H > H_r)$ respectively. Note that to get $P(A_r|H > H_r)$, we assume that the rival's hand cards can be observed at the end of the game. But we must emphasize that rival hand cards are not need in testing phase. To estimate $P(H > H_r)$, we apply Monte Carlo method to sample the rival's hand cards, and calculate the frequency that the agent has better hand cards. Then in every game, we first get a policy distribution from CS according to state s, while feeding AP to the policy modification network to get the modification vector, and multiply it to the original distribution. We use Adam(Kingma & Ba, 2014) optimizer to optimize $(\mathcal{L}_1 + \mathcal{L}_2)$ based on gradient descent . Meanwhile, we also dynamically adjust values in CS with history accumulated during training (with the same

Table 2. Competition between each agent.

	CHEAT'EM	TABLE	PPO	DQN	RAISE_RULE	RULE	RANDOM
CHEAT'EM	-	**0.198**	**2.064**	**2.667**	**2.377**	0.560	2.020
TABLE	-0.198	-	1.418	1.442	1.446	**0.588**	1.735
PPO	-2.064	-1.418	-	-0.030	-0.035	0.513	2.837
DQN	-2.667	-1.442	0.030	-	-0.112	0.558	2.108
RAISE_RULE	-2.377	-1.446	0.035	0.112	-	0.349	**2.969**
RULE	-0.560	-0.588	-0.513	-0.558	-0.349	-	-0.066
RANDOM	-2.020	-1.735	-2.837	-2.108	-2.969	0.066	-

scheme as pre-training). We further describe our training procedure in alg3.

5. Experiments

5.1. Experimental Setups

Baselines

We evaluate Cheat'em against following baseline algorithms under RLCard environment. TableAgent mentioned before is also evaluated. For PPO, we use the agent constructed by ourselves. For other models, we directly call RLCard's interfaces. Besides, a self-defined raise-only agent is added.

- **PPO**(Schulman et al., 2017): A commonly used policy gradient method, which is considered simpler and more general than its TRPO predecessor.
- **DQN**(Mnih et al., 2013): A classical RL model that first combines Q-Learning with deep neural network.
- **Raise_Rule**: A self-defined raise-only agent. Though simple, it shows surprisingly good performance.
- **Rule**: A simple rule-based deterministic policy agent provided by RLCard.
- **Random**: A uniform random policy agent provided by RLCard.

Metrics

As a gambling game, the money won or lost is naturally the measurement of model quality. In experiments, we use average money reward of 10000 games as evaluation metrics. By default, the small blind starts with 1 dollar, and the big blind starts with 2.

5.2. Competitions

We conduct tournaments to demonstrate the advantage of our Cheat'em, the results are shown in Tab.2. Overall, our algorithm, Cheat'em, dominates the leaderboard by beating all the other agents, including PPO and baselines provided by Rlcard. Although Cheat'em do not show obvious advantage when competing with TableAgent, it performs much better than TableAgent confronting with other baselines. It is also observed that when competing with random agent, PPO and Raise_Rule agents reap the highest returns. A potential reason is that, the best-response strategy to random agent is to raise no matter how bad the card is, with the observation that random agent winds up with folding with high probability. As a result, one can always raise and wait for the random rival to lose money. Cheat'em does not overfit on this extreme situation, as we use eq.3 to constrain the agent's action distribution. In spite of this, Cheat'em performs much better than others in almost all other cases, indicating that our algorithm is general enough.

5.3. Ablation Studies

In order to further investigate the effectiveness of our approach, we carry out an ablation experiments on Cheat'em. We apply Cheat'em without CS online updating or policy modification to see how these two methods influence the final result. The ablation models are as follows:

- **Static-Cheat'em**: Cheat'em agent without online CS updating. It uses the pretrained CS and never update it. The policy modification network is training just as Cheat'em.
- **Table**: This agent chooses action directly depend on CS and never update it.

We test these two agents and Cheat'em by conducting tournaments with agents including Random, Raise_Rule, and Rule. The result are shown in Tab.3. We can observe that Cheat'em shows the best performance compared to the others. Without CS online updating, static-cheat'em reaps lower reward than cheat'em, which needs to be attributed to the fact that CS faced a completely different opponent in training than it does now. This reason also explains the bad performance when Table agent plays with PPO or DQN as shown in Tab. 2. Besides, though all the above agents easily beat Random agent, Cheat'em wins the most.

Table 3. Competition between ablation agents and rivals.

	RULE	RANDOM	RAISE_RULE
CHEAT'EM	0.560	**2.020**	**2.377**
STATIC-CHEAT'EM	0.486	1.837	2.297
TABLE	**0.588**	1.732	1.446

6. Conclusion

In this project, we studied application of MARL algorithm on Limited Texas Hold'em. We first constructed PPO Agent to supplement baselines, then proposed two reward shaping schemes to address the problem of vague supervision, but saw no apparent improvement. Inspired by CFR, we proposed Cheating Sheet, which inherits the core idea of CFR but largely compresses the state space (by fusing states according to self-defined card types). Experiments suggest that CS successfully learns value information. To equip our model with Rival-awareness, we further propose Advantage Probability, and apply a MLP to utilize it. The final model, Cheat'em, outperforms all baselines. Throughout the project, we conduct abundant experiments, including tournaments and ablations to support our designs.

Acknowledgements

We would like to appreciate Prof. Wen and all TA's for their wise guidance and patient help. In this course, we took a glance of game theory world, and really acquired some insights of MARL.

References

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. A. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. Trust region policy optimization. *ArXiv*, abs/1502.05477, 2015.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017.

Zha, D., Lai, K.-H., Huang, S., Cao, Y., Reddy, K., Vargas, J., Nguyen, A., Wei, R., Guo, J., and Hu, X. Rlcard: A platform for reinforcement learning in card games. In *IJCAI*, 2020.

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. Regret minimization in games with incomplete information. In Platt, J., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf.