Learning to Cooperate with Unknown Agents in the Hanabi Challenge

---

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

Yizhen Chen

December 2021

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: Yizhen Chen

This Thesis has been read and approved by the examining committee:

Advisor: Haifeng Xu

Advisor:

Committee Member: Hongning Wang

Committee Member: Yangfeng Ji

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Jennifer L. West, School of Engineering and Applied Science

Abstract

In recent works on multi-agent reinforcement learning, more and more researchers are looking at Hanabi Game, a board card game. Unlike previous popular studies on Go, Poker, and Atari, Hanabi focuses more on cooperation between players. Zero-sum games like Poker and Go are more straightforward for the agent to learn and understand the opponents. In a cooperative game like Hanabi, players need to exchange information frequently, and in addition, the information exchanged in the Hanabi game is imperfect. Hanabi game has become a new benchmark during the last two years. Cooperative games like Hanabi can teach AI to communicate more efficiently with other players, including human players. Such an environment can further enable AI to learn to communicate and cooperate with humans in the real world. There is a growing number of researchers focusing on the Hanabi challenge. However, most previous works still focus on learning agents in the self-play setting, with little focus on the team ad-hoc setting. We proposed a new multi-agent RL method and new metrics to measure the performance of each agent in the team ad-hoc setting. This thesis focuses on some of the approaches in which agents can improve their performance in team ad-hoc settings and comprehensively evaluate these approaches.

Acknowledgements

First of all, I would like to thank my advisor, professor Haifeng Xu for his full support. With his assistant, I have successfully completed this thesis project and learned relevant research experience in the process. In particular, he guided me to gain professional knowledge in the field I was not familiar with. In addition, he listened to every question I had and offered insights that could help me in every discussion.

Furthermore, I would like to thank all the committee members of my master thesis, professor Hongning Wang and professor Yangfeng Ji for listening carefully to my presentation and giving me many valuable suggestions to improve my research further.

Finally, I would like to give my family and friends special thanks for their full moral and physical support during my time of need. In particular, I also would like to thank my friend Alan Luo for his silent support.

Table of Contents

Abstract **iii**

Acknowledgements **iv**

1 Introduction **1**

1.1 Background 1

1.2 Thesis Overview 2

2 Related Works **3**

2.1 Policy Gradient 3

2.2 Multi-Agent Reinforcement Learning 3

2.3 Meta Reinforcement Learning 4

3 Learning to Reinforcement Learn in Multi-Agent System **6**

3.1 Preliminaries 6

3.2 Independent PPO and Multi-Agent PPO 7

3.3 Meta-Learning with PPO 9

3.3.1 Learning Architecture 11

4 Experiments **13**

4.1 Hanabi Environment 13

4.1.1 Game Rules 13

4.1.2 Hanabi Learning Environment 15

4.2 Implementations 16

4.3 Evaluation 18

4.4 Results 19

5 Discussion and Future Work **24**

5.1 Discussion 24

5.2 Conclusion 25

5.3 Future Work 26

References **27**

A Implementation Details **30**

A.0.1 Implementation 30

A.0.2 Hyperparameters 30

Chapter 1

Introduction

1.1 Background

In theory, most of the assumptions of the learning agents are homogeneous. However, in reality, each person is generally different. In human society, communication and division of labor among people are essential. For example, many existing multi-agent learning works are conducted on zero-sum games, such as Go[1] and Poker[2]. In a zero-sum game, the agents act as opponents, so there is little communication between them. In this case, it is difficult for the learning agents to be well utilized in the more complicated environment of the real world. Therefore, research on cooperative environments is of more practical value.

In a fully cooperative environment, agents need to cooperate by establishing communication to maximize the global reward. In general, agents in a cooperative environment are diverse, which means that a single learning agent will encounter other unseen agents that use different strategies. It is also a challenge to learn to cooperate with these unseen agents.

Hanabi game is a cooperative card game with limited information passed between players. It represents the type of cooperative game with incomplete information. DeepMind has proposed this Hanabi challenge[3], which includes two different challenges settings: self-play learning and ad-hoc teams. Most previous AI-related works focused on the challenge of self-play, while for the challenge of ad-hoc teams, there is no good approach to outperform existing hard-coded rule-based agents. Moreover, the research on the ad-hoc teams challenge is more relevant, making agents have a more comprehensive range of applications.

In a fully cooperative game, the goal of agents is to maximize the global reward, which is also true in the Hanabi game, except that it is difficult to measure the performance of each individual agent in the game. Thus we need to record the score of the whole system and calculate each agent's marginal contribution to evaluating the agent's performance.

In a cooperative multi-agent environment, conventional single-agent reinforcement learning algorithms usually do not have outstanding performance. However, there are many previous works in this field proposing various SOTA multi-agent reinforcement learning algorithms. Based on these works, we improve the algorithm using multiple approaches and evaluate the performance of these approaches in the setting of Ad-hoc Teams.

1.2 Thesis Overview

This paper aims to evaluate the performance of several approaches under ad-hoc teams setting. Chapter 2 will describe the work related to these methods and the SOTA algorithms we will use. Chapter 3 will explain how we use these methods to improve our baseline algorithm on Hanabi games in ad-hoc teams. Chapter 4 introduces the Hanabi game, and the Hanabi Learning Environment built for the Hanabi challenge. In this chapter, we also describe how we evaluated the different algorithms and further analyzed them in the context of experimental results. The final chapter covers a discussion and analysis of the results and a conclusion.

Chapter 2

Related Works

2.1 Policy Gradient

In reinforcement learning, policy gradient methods typically focus on modeling and optimizing the policy. Recently, the popular on-policy policy gradient algorithms like Proximal Policy Optimization (PPO)[4] and Trust Region Policy Optimization (TRPO)[5], can converge very quickly to find the optimal policy in reinforcement learning problems. Compared to TRPO, PPO is more concise while still achieving the performance of TRPO. Engstrom *et al.*[6] suggest that the better performance of PPO over TRPO comes more from its code-level optimization than from macroscopic algorithmic differences. PPO is a policy gradient algorithm based on the Actor-Critic architecture, which guarantees the speed and convergence of KL, making it easier for us to experiment and adjust. Compared to other off-policy reinforcement learning algorithms like Rainbow[7] and SAC[8], PPO seems less sample efficient and does not necessarily cover all actions. Thus all actions need to be sampled by interacting with the environment.

2.2 Multi-Agent Reinforcement Learning

A fully cooperative stochastic game can be solved by maximizing the joint payoff. In other cases, however, determining a multi-agent learning goal is not easy because the payoff functions of the agents are interrelated with each other and difficult to maximize independently.

In general, there are two types of MARL framework: value decomposition (VD) and centralized training and decentralized execution (CTDE). However, A naive approach to solving a MARL problem using a centralized controller turns the problem into a single agent reinforcement learning

problem. Since the goal of the agents is to maximize their global rewards, it is straightforward to apply the single-agent reinforcement learning algorithms. Specifically, the actions of all agents are considered as one action vector, also known as joint action. The learning goal for this single agent algorithm is to learn a joint policy that can make the corresponding action vector in each state, such that the discounted cumulative reward is maximized. In this case, some value-based reinforcement learning algorithms like Q-learning can also be used in multi-agent systems. For example, Rainbow, an improved algorithm based on DQN[9], performed well in the Hanabi challenge.

In fully cooperative settings, the multi-agent system usually needs to maximize the global payoff. A centralized controller is not always feasible for decision-making in real-world situations. If a fully distributed approach is used, each agent will learn its value function and policy independently, without considering the influence of other agents and the non-stationarity of the environment. Using the characteristics of the actor-critic network in reinforcement learning, combined with the CTDE framework, the agent can learn the features of the whole system and have independent policies. Both MADDPG[10] and MAPPO[11] utilize the CTDE framework on top of the Actor-Critic network to achieve MARL. The MADDPG extends the DDPG[**?**] algorithm from single-agent reinforcement learning algorithms to a multi-agent environment. The MAPPO, on the other hand, uses the global value function based on PPO.

In the policy-based approach using the CTDE framework, the centralized value function is modeled directly using global information without considering the characteristics of individual agents. While a multi-agent system is composed of multiple individual agents on a large scale, such a value function is difficult to learn, and it is hard to derive the desired policy. When relying only on local observations, it is impossible to determine whether the current reward is obtained due to its action or the action of other teammates in the environment. The Value Decomposition Networks (VDN)[12] proposed by DeepMind decompose the global $Q(s, a)$ value into a weighted sum of every individual local $Q_i(s_i, a_i)$ values so that each agent has its local value function. In this case, the properties of individual actions in the joint action Q-value's structural composition make learning easier. On the other hand, we can also apply the centralized training approach on VDN, which can overcome the problem of non-stationary in multi-agent systems.

2.3 Meta Reinforcement Learning

Meta-learning gives the agent the ability to remember important historical information about a class of tasks and quickly establish correlations between states and rewards using (limited) samples. The

objective of meta-learning is to quickly learn how to respond based on previous experience and a small number of samples when exposed to new tasks that have not been seen before or when adapting to a new environment. There are three common approaches to meta-learning[13]: 1. Learn effective distance metrics, i.e., the relationship between things (Metric-based): Matching Networks[14], Prototypical Networks[15]; 2. Learn how to model using a (recurrent) network with external or internal memory storage (Model-based): Memory-Augmented Neural Network(MANN)[16]; 3. Finally, learn how to learn by training models that aim for fast learning (Optimization-based): MAML[17], Reptile[18].

On the other hand, meta-reinforcement learning is doing meta-learning on reinforcement learning. Simply put, making the RL agent be a meta-learning agent. In other words, instead of solving the problem of one environment or one task (usually represented by an MDP model), the meta-agent is trying to solve a series of tasks with the different environments (distribution of MDPs). Based on the fact that our environment is built in a multi-agent system and that the action states conducted by each agent in the system are continuous and in communication with each other. Among them, Wang *et al.*[19] and Duan *et al.*[20] mentioned that the hidden states of RNNs can be used to learn the relation between the present state and historical information to adapt to different tasks. We also use this method to improve the multi-agent reinforcement learning algorithm by adding a recurrent network for internal memory.

Chapter 3

Learning to Reinforcement Learn in Multi-Agent System

3.1 Preliminaries

The reinforcement learning problem is usually described as a Markov decision process(MDP). The system generally consists of an agent selecting actions over a specific environment. The environment will return a reward to the agent after each action step(Figure 3.1). The objective of the agent is to maximize the expected cumulative rewards in the system. When there are multiple agents interacting

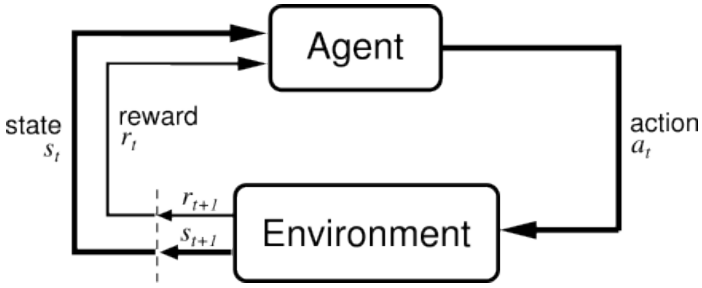


Figure 3.1: A framework for reinforcement learning (which also represents Markov decision processes)[21].

with the environment at the same time, the whole system becomes a multi-agent system. However, each agent still follows the goal of reinforcement learning, which is to maximize the cumulative reward that can be obtained, and the change in the global state of the environment is related to the joint action of all the agents. Therefore, the impact of joint action needs to be taken into account in the process of learning strategy(Figure 3.2). The Hanabi game we study is a fully cooperative multi-

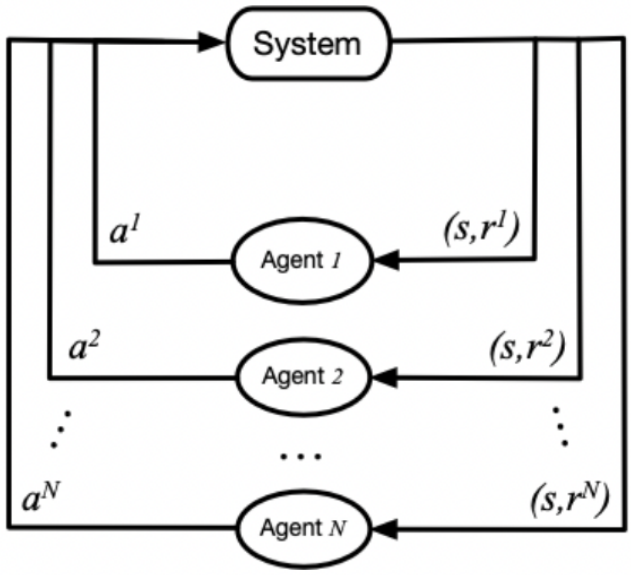


Figure 3.2: A framework for multi-agent reinforcement learning(Markov game process)[22].

agent reinforcement learning with imperfect information, which extends the MDP to a Decentralized Partially Observable Markov Decision Process(Dec-POMDP)[23]. We define a decentralized partially observed Markov decision processes (Dec-POMDP) with shared rewards by $M = (S, A, O, R, P, n, \gamma)$, where S is a state set. A_i is an action set for each individual agent i. $P(S'|S, A)$ is a transition probability distribution from S to S' with $A = \times_i A_i$ is the set of joint actions. O_i is the observations set for agent i. $R(S, A)$ is the shared reward function. $\gamma \in [0, 1]$ is the discount factor. The action-observation history of each agent can be presented as $\tau_i = (A_{i,0}, O_{i,1}, ..., A_{i,t-1}, O_{i,t})$, then the distributed policy for each agent is $\pi_i(\tau_i)$.

Dec-POMDP assumes that each agent can use its own local observations to maximize the global objective reward. However, the computational complexity is high for large-scale multi-agent problems, and it is often difficult to converge. We use neural networks to parameterize Actor and Critic with the centralized training distributed execution framework in order to reduce computational complexity.

3.2 Independent PPO and Multi-Agent PPO

Generally, the policy gradient algorithm learns the policy directly. The policy is represented as $\pi(a|s, \theta)$ with a parameter θ. We define the reward function as the expected reward, and the

objective of the algorithm is to maximize the reward function:

$$J(\theta) = E_{\tau\sim\rho_\theta}[R(\tau)] = E_{\tau\sim\rho_\theta}[\sum_{t=0}^{T}\gamma^t r(s_t, a_t, s_{t+1})] \tag{3.1}$$

where the trajectories are $\tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$. In this case, using the gradient boosting algorithm, we can find the optimal parameters θ. Independent PPO[24] and Multi-Agent PPO[11] both expand PPO(Proximal Policy Optimization)[4] to a multi-agent system, and PPO is also proposed mainly to solve the problem of policy gradient algorithm in which policy is updated too much at a time leading to policy gradient eventually unable to converge. In order to improve the stability of training, we should avoid updating the parameters that make the strategy change drastically in one step, so the clip function of PPO can make the new policy as close as possible to the old one to prevent the policy from updating too much or too little. The PPO algorithm is essentially a PG algorithm based on the Actor-Critic framework. The core optimization formula of PPO is:

$$L^{CLIP}(\theta) = \hat{E}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{3.2}$$

where the $r_t(\theta)$ is the ratio between the new policy $\pi_\theta(a_t|s_t)$ and the old policy $\pi_{\theta_{old}}(a_t|s_t)$:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \in [1-\epsilon, 1+\epsilon]. \tag{3.3}$$

PPO is a simplified version of TRPO, where the ratio of the new policy π_θ to the original policy π is clipped, forcing it to be limited to a certain range, and takes the minimum of the product of the ρ_t before and after the clipping and the Advantage function $A_\pi(s_t, a_t)$ for optimization. PPO applies a lot of tricks in the actual implementation to make the RL agent perform better. As mentioned in the previous works, The performance improvement for PPO is not brought about by clipping the ratio of the new policy to the original policy, but by code-level optimizations. In our approach, we also make use of these tricks of PPO. For individual agents, PPO uses Generalized Advantage Estimation (GAE)[25] to effectively reduce the estimated variance of the policy gradient, making the algorithm converge. The advantage estimation after using GAE for each agent a is:

$$A_t^a = \sum_{l=0}^{h}(\gamma\lambda)^l\delta_{t+l}^a, \tag{3.4}$$

where $\delta_t^a = r_t(z_t^a, u_t^a) + \gamma V_\phi(z_{t+1}^a) - V_\phi(z_t^a)$ is the TD error at step t.

In addition, PPO also uses value clipping method. Similar to the clipping function mentioned earlier, the main purpose of value clipping is to prevent the value function from being updated too much. The Value Clipping is defined as:

$$L^V = \min[(V_{\theta_t} - V_{targ})^2, clip(V_{\theta_t}, V_{\theta_{t-1}} - \epsilon, V_{\theta_{t-1}} + \epsilon) - V_{targ})^2] \tag{3.5}$$

where V_θ is clipped.

Independent PPO is trained and executed using a decentralized learning approach similar to independent Q-learning, this means that IPPO will have many separate execution units in a multi-agent environment since IPPO is also based on an actor-critic framework. In this case, each agent in the Hanabi game has an independent actor-critic network, and it does not share any information with others during training.

Previous work proposed MAPPO as a state-of-the-art(SOTA) MARL approach to Hanabi self-play challenge. However, we would like to take this a step further and extend this method to the ad-hoc setting. Inputting an observation of the complete global state on the value network can transform the POMDP into an MDP, thus making value learning easier. In MAPPO, instead of inputting the global state directly on the critic network, the local observation of each agent is concatenated together as the input to the value network. Each local agent receives a local observation *obs* and outputs an action probability, and each agent uses an actor network. While the critic network receives observation *obs* from all agents, the space of centralized observation is $cent\_obs\_space = n \times obs\_space$ where n is the number of agents and the output is a value V, which is used for the actor update. While in MAPPO for Hanabi game, we also include Action Masking for filtering out the restricted actions due to game rules in order to accelerate the training process.

3.3 Meta-Learning with PPO

MAPPO is proposed to solve the collaboration problems in multi-agent environments better, as the framework of CTDE can help agents establish communications with each other. However, since the CTDE framework only allows the agents to develop connections with each other during training, the actions can only be selected by a learned policy during the execution. In this case, the MAPPO agent must be trained with the corresponding agent in order to obtain better performance with that specific agent. We want to take MAPPO a step further to generalize the agent learning policy so that the learning agent can get better performance in ad-hoc settings. Usually, there are two

approaches to achieve this goal, either by online adaptation so that the agent can learn to interact with the other agent and update its own policy in the execution process after training, or by directly allowing the agent to learn more information about different tasks during the training process so that the policy can obtain some generalization ability. In our approach, we refer to the latter to implement the learning of policies, adding meta-reinforcement learning to MAPPO.

According to Wang *et al.*[19], the overall structure of Meta-RL and RL is nearly identical, except for the difference in the definition of the policy function(See figure 3.3).

- RL: $a_t = \pi_\theta(s_t)$
- Meta-RL: $a_t = \pi_\theta(a_{t-1}, r_{t-1}, s_t)$

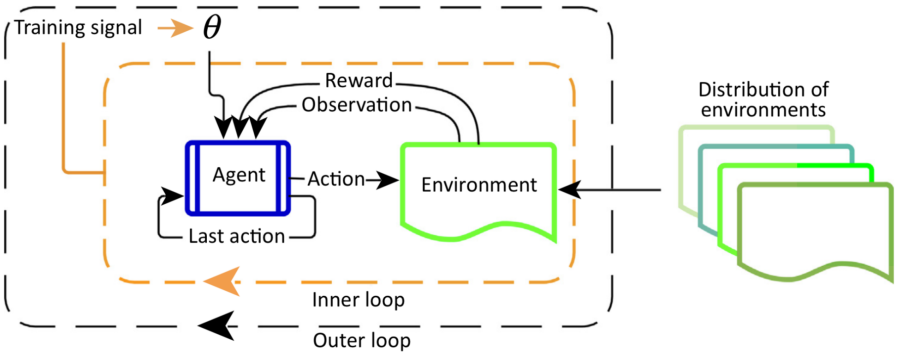


Figure 3.3: This figure shows the Meta-RL with two optimization loops. The system samples a new environment and updates parameters that define the agent's action at each environment step in the outer loop. The agent interacts with the environment in the inner loop to maximize the reward.[26]

This design introduces historical information so that policy can learn the dynamic relationship between the current MDP state, the previous reward, and the previous action. We apply the recurrent policy as the agent actor policy. The main architecture of our approach is to combine the recurrent network in our Actor-Critic network. Unlike optimization-based meta-learning methods like MAML, memory-based meta-learning methods using RNNs freeze the parameters of the model after training. The internal state (the pattern of activation over recurrent units of each agent) of the RNN is reset each time at the beginning of each episode during evaluation. In addition, the learned RNN policy is history-dependent and can generalize when exposed to new tasks.

3.3.1 Learning Architecture

In meta-reinforcement learning, the training procedure is also called meta-training. Hidden states of the recurrent network are the memory used to track the characteristics of the trajectories. In order to capture trajectories in different MDP tasks, our training process works as follows:

- Sample a new cross-play game with a different agent as a new task$_i$.
- Reset the hidden state of the model.
- Capture the trajectories and update the model weights.

We repeat this process when the learner agent is trained with the current agent for one cycle(Refer to Algorithm 1 for details). However, before the meta training process, we also pre-train the learner agent in the self-play setting. The final training process is shown in Fig. 3.4.

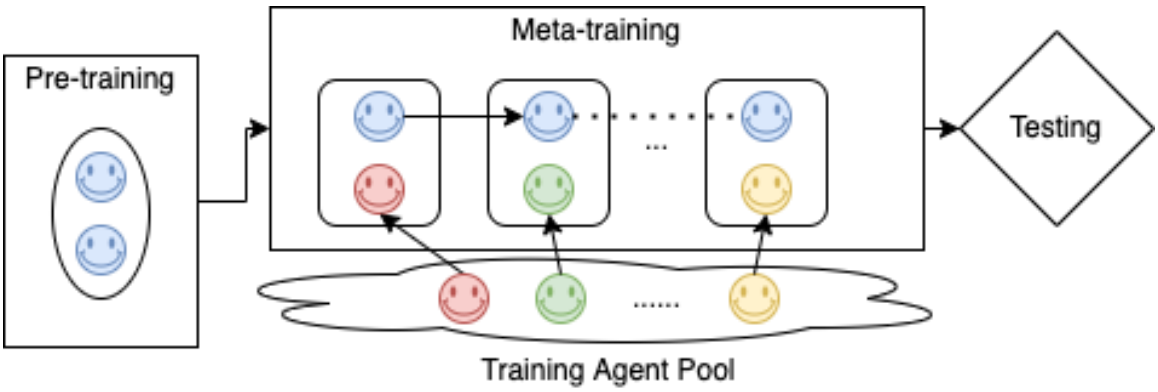


Figure 3.4: This figure shows the training process for Meta-RL agent: **1. Pre-training:** the learner agent trained in the self-play setting. **2. Meta training:** the learner is trained against the agent randomly taken from the training agents pool. Note that the agents in the training agent pool do not appear in the evaluating agents pool.

Algorithm 1 Meta-MAPPO Training Procedure

Require: Number of agents $N \geq 0$
Require: Learning rate α
  Initialize replay buffers $\mathcal{D}$;
  while not done **do**
    Randomly sample k partner agent from training agent pool;
    for partner $j = 1, ..., k$ **do**
      for each learner i **do**
        Select action $a_i \sim \pi_{\theta_i}(\cdot|o_i)$ based on local observation of learner agent;
        Execute a_i in the environment and observe o_i', r_i;
        Select action $a_j \sim \pi_{\theta_j}(\cdot|o_j)$ based on local observation of partner agent;
        Execute a_j in the environment and observe o_j', r_j;
        Store turn-based trajectory; $(o_i, o_j, a_i, a_j, r_i, r_j, o_i', o_j')$ in replay buffer $\mathcal{D}$;
        $o_i \leftarrow o_i'$;
        $o_j \leftarrow o_j'$;
      end for
    end for
    for learner agent $i = 1, ..., N$ **do**
      Randomly sample s samples from $\mathcal{D}$;
      Compute $\nabla_\theta L^{CLIP}(\theta)$ using s based on Equation 3.2;
      Update model parameters with gradient descent: $\theta_i \leftarrow \theta - \alpha \nabla_\theta L^{CLIP}(\theta)$;
    end for
  end while

Chapter 4

Experiments

4.1 Hanabi Environment

As we described in the introduction section, the Hanabi game is designed as a cooperative game. All players have a common goal that needs to be achieved by collaboration. Players are required to build up the perfect fireworks by placing the cards on the table in the correct order to earn points.

4.1.1 Game Rules

The Hanabi game supports 2 to 5 players, and each player draws 5 hands when there are only 2-3 players in the game and 4 hands when there are 4-5 players in the game. In our experiments, however, we only tested the performance of the various approaches in 2 players Hanabi game.

Players can only see their teammates' cards, but not their own. Hence, for each player, he only has partial observation of the environment. In addition, players are only allowed to establish communication through the 'cue' action, where one player tells another player about his cards, so they know what to play or discard. Since a limited number of cues can be provided, good players can communicate strategically and utilize conventions such as "discard the oldest card first".

The basic components in the Hanabi game: * 50 Hanabi cards with 5 colors (white, red, blue, yellow, green), the values on the cards to be dealt are three **1**s , two **2**s, **3**s, **4**s, and one single **5** for each color (The basic setup of a 4-player Hanabi game is shown in the figure 4.1).

- 8 Blue tokens represent the number of clues that remain.
- 3 Red tokens represent the number of lives remaining.

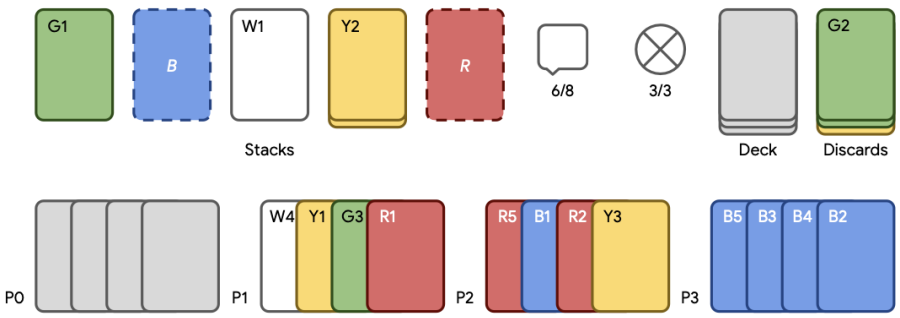


Figure 4.1: Example of a four-player Hanabi game from the point of view of the first player 0[3].

The ultimate goal of the game requires the players to collect five-color decks of cards, each consisting of numbered cards valued from 1 to 5. And each sequence of the deck has to be arranged in ascending order starting from value 1. Each player will be randomly assigned 5 cards to their hand(two-player Hanabi game) at the beginning of the game, 8 blue tokens are placed in the box, and the 3 red tokens are out of the box. For example, in a 2-player game, each player starts with 5 cards, which he cannot view, but can only be viewed by teammates. During the play, you need to play 1-5 of each color in order onto the table. Because the game is a cooperative game, the total score is calculated based on the number of cards played. With a score of 25, the higher, the better. When it is a player's turn, he must complete one of the following three actions(without skipping the turn):

- Giving a piece of information: pick any player, cue him a value hints or color hints, and then point out all the eligible cards in that player's hand, without any sequential information. Each time the action is performed, a clue token is spent, and if all clue tokens are used, the current player will not be legal to perform this action. Note: Each player cannot provide both color and value information in one turn.

- Discarding a card: place this card openly in the discard pile and recover a blue clue token and draw a card, keeping that card invisible to you. If the clue tokens are full (all 8 are unused), then this action will be illegal to select. After this action, the player needs to add a card to his hand.

- Playing a card: the action is considered successful if the card can be successfully placed after any existing cards sequence according to the rules, then the card is added to the sequence to form a new card deck, and the player needs to add a card. Otherwise, if the card cannot be attached to any sequence of cards, the action is considered a failure. Then a live token is

consumed, and the card is discarded. After this action, the player also needs to add a card to his hand.

The game is turn-based, with players playing in a clockwise direction, and each player can only perform an action on their turns and cannot communicate in any way on other players' turns. Each player has to play the cards in their hands and form sequences of 5 cards in ascending order from 1 to 5 by different colors in order to score as many points as possible. There are 3 ways to end the game of Hanabi:

- If the third red token is consumed, the game is lost. Therefore, the game ends immediately, and the players will receive a score of 0.
- The game ends immediately, and it is a stunning victory if the players manage to make the 5 fireworks(5 color card sequences) before the cards run out. The players are then awarded the maximum score of 25 points.
- The game ends if a player takes the last card from the pile. Then each player plays one more time, including the player who picked up the last card. The players cannot pick up cards as the pile is empty during this final round. Then the final score is the total number count of cards in all 5 sequences.

Communication between the players is essential to Hanabi Game. If you strictly follow the rules, the communication between you and your teammates is limited. However, we can analyze the intention of our teammates to provide information by looking at all known cards on the field and the previous hints.

4.1.2 Hanabi Learning Environment

DeepMind Proposed Hanabi Learning Environment(HLE) along with Hanabi Challenge in 2019[3]. HLE is an open-sourced code framework written in Python and C++. It provides an interface for agents to interact with the environment. The environment class consists of state, rewards generation for each agent. By taking one-step action, each agent can observe the environment state partially. HLE provides different types of Hanabi games from "Small" to "Full". However, we only consider 2-player "Full" Hanabi environment in our experiments.

HLE uses a python API similar to OpenAI's gym[27], providing state class in the environment. The state for each step is a combination of observations from all players on the field. For each player,

the observations only have information about all other players but not their own. This player observation consists of human-readable observation and a binary vectorized observation. We modified the original environment so that each step state function would return not only the observations for each player but also a global observation for a centralized critic network to learn from. In addition, each step also returns the available actions in the current state for action masks. HLE formulates the actions into four types: PLAY, DISCARD, REVEAL RANK, REVEAL COLOR. Each executed action must be legal for step function in the environment class. For example, a player cannot perform REVEAL RANK or REVEAL COLOR action if the number of remaining clue tokens is equal to 0. The reward for each step is calculated by the score difference in HLE. If the player plays a card in each step, the total score will be increased by one point, and the returned reward will be 1. If the player performs other actions in each step, the score will not be increased, and the reward will be 0. If the player fails the game in this step, the total score will be 0, and the reward may be negative.

4.2 Implementations

Hanabi is a turn-based board game, and all players are not acting simultaneously. For both IPPO and MAPPO, they update the policy after each round of the game. Only then MAPPO can get the global information. Therefore, we saved various information for each round in training, including the global observations, the actions selected by all players, and the turn-based rewards after executing the actions. Since Independent PPO uses a decentralized framework, we extend the original PPO implementation for a multi-agent environment so that each agent has a separate independent policy-value network for training and execution. Similar to vanilla PPO, however, in the IPPO trainer, each agent only receives the state in each turn as the player's observation since each agent uses an independent PPO strategy. The overall underlying framework of MAPPO and IPPO is the same. The only difference is that MAPPO is trained with the value network receiving global observation information.

We used the actor-critic framework for both IPPO and MAPPO, where actor-network and critic-network are separate. In addition, we use an MLP policy with two fully-connected layers for the policy network and include a value clipping method to calculate value function loss (see Figure 4.2).

In the section of mete reinforcement learning, we use the GRU[28] policy based on the original actor-critic network, keeping a fully connected layer as the encoding layer. We also concatenate the actions of the previous step, the previous rewards, and the observation encoding as the input of GRU.

In this operation, we also use one-hot encoding on the previous action to make the representation of the action to be more expressive. The overall structure is in Figure 4.3. In our neural network implementation schematic diagram, the input units encode current observations. Previous actions and rewards are all connected to hidden units, which are fully connected to the GRU units.

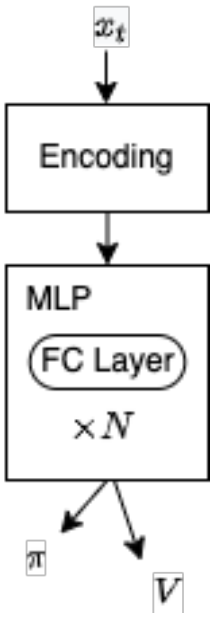


Figure 4.2: Actor-critic with MLP policy. x_t =state of current step t; π =policy; V =value function.

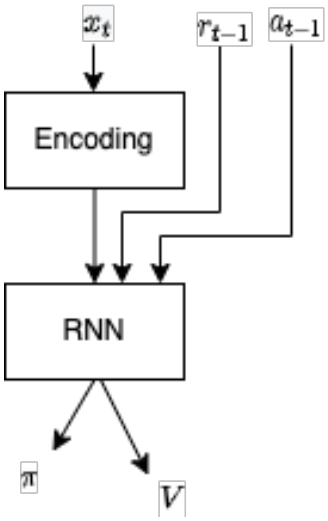


Figure 4.3: Actor-critic with recurrent policy. x_t is the state for current step t, r_{t-1} is the reward for previous step, and a_{t-1} is the one-hot representation of previous action; π =policy; V =value function.

4.3 Evaluation

The agents in the Hanabi game use many different approaches. Since our experiments focus on evaluating the performance of different learning agents in the ad-hoc teams setting, we need to evaluate multiple games for each agent cooperating with different teammates. In this case, we collected various agents and put them into an agent pool to evaluate each of them in a cross-play manner. According to Canaan *et al.*[29], we re-implemented all the rule-based agents the author collected, including IGGI, Internal, Outer, Van den Bergh, and Piers. These rule-based agents are introduced by Walton-Rivers *et al.*[30]. Internal agent and Outer agent are both presented by Osawa[31]. The Internal rule remembers the information it gets from other players about its own hand but not the information other players received. And the outer agent is able to remember the hand information obtained by other players. IGGI only plays a card that can be guaranteed to play and prefers to discard the oldest card in hand. On the other hand, Piers uses an ifRule strategy based on IGGI. In this case, it has a probability of 0.6 to play a safe card when the number of live tokens is greater than 1. At the same time, it is more inclined to give the other players information about dispensable cards when the number of clue tokens is less than 4. The Van den Bergh rule is introduced by Van den Bergh *et al.*[32], and it uses a similar ifRule as IGGI. However, it prefers to provide information about the playable cards that your teammates can play. It also provides information about the dispensable cards if there are no playable cards to hint. In addition, we also implemented an improved simple agent. These agents are all rule-based, and they will follow a hard-coded strategy to play the game. Moreover, we also added several learning agents as baselines in the test agent pool, including Rainbow[7] agent. The Rainbow baseline is based on Google's Dopamine framework. Other learning agents include IPPO, Meta-IPPO, MAPPO, and Meta-MAPPO.

For reference, we also evaluated the average score of each agent in the self-play setting. Finally, we compared the average score in the ad-hoc setting to calculate the score difference. In this case, we can determine how much the teammate affects the agent in the ad-hoc team setting.

In addition, since Hanabi games are turn-based games, there are both early-hand and late-hand players in each round. Therefore, we also have to consider the difference between different agents as early hand players and late hand players in the ad-hoc setting. Particularly, in the case of evaluating IPPO agents, as IPPO is a decentralized algorithm, each agent has an independent policy. Then the policies for early hand and late hand are likely to be different in this case. Therefore, we need to eliminate this error in the evaluation and record the scores in both early-hand and late-hand cases.

Because we evaluate many different agents, covering rule-based agents and learning agents. These agents all perform differently in the self-play setting, especially some learning agents perform well in the self-play setting, but not in the ad-hoc setting. Therefore, in order to investigate the performance of different agents in the ad-hoc setting, we also reduced the weight of the self-play score in the total score.

4.4 Results

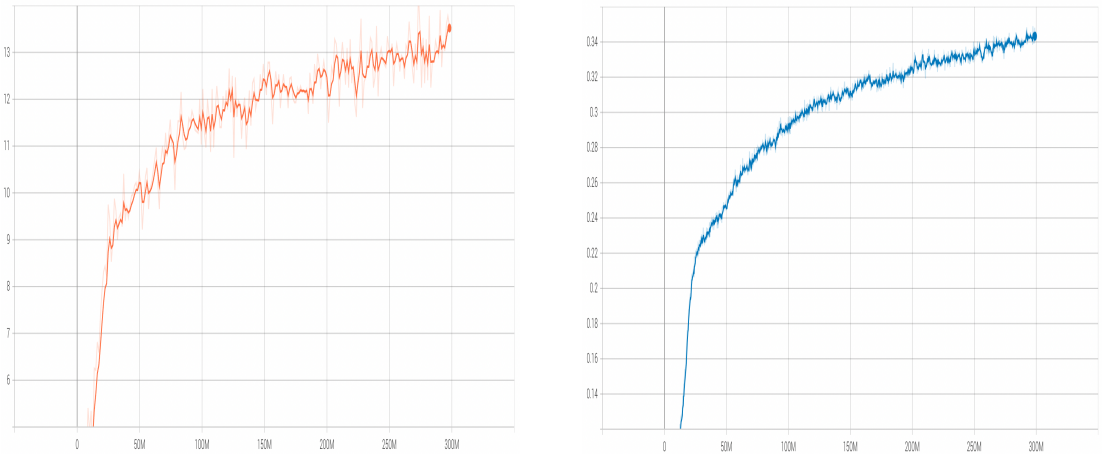
In the self-play setting, the agent uses the exact same policy to play with its clone. The experimental results for all agents in the self-play setting are presented in Table 4.1. Among learning agents, Rainbow, IPPO, and MAPPO all outperform the best baseline hard-coded agents in our experiments. On the other hand, the two improved algorithms Meta-IPPO and Meta-MAPPO, do not perform better than most hard-coded agents in the self-play setting. However, Yu *et al.* mentioned that the best average score of MAPPO can reach 23.85 due to the different parameters used in our experiments and the training method[11]. Moreover, MAPPO was trained for 10B environment steps in the original paper. In our experiments, all learning agents were trained for only 400-500M environment steps due to the limitation of the equipment.

MAPPO has the highest score among all the learning agents because it uses the CTDE architecture, which allows the agents to obtain global information of environmental state and use shared policy during training, enabling communication between agents. Even during the execution, it can cooperate with the teammate agent's actions. IPPO, on the other hand, is completely opposite to MAPPO. It uses an independent policy in the training process and does not share information between agents. Therefore it receives a lower score than MAPPO, which uses the CTDE method. As for Meta-IPPO and Meta-MAPPO, they receive less than 13 points of performance in the self-play setting, which we believe that using the RNN policy makes the network more difficult to train. In fact, in our training(as figure 4.4 shows), Meta-IPPO and Meta-MAPPO with the RNN policy have not converged to a global minimum at all.

To investigate the learning agents in the ad-hoc team setting, we evaluate agents by cross-play, which means each agent has to play in a team with every agent in the pool to create a matchmaking cross-table. The figure 4.5 shows a heatmap table of agents' performance in the ad-hoc setting. In the cross-table, we can find that the score obtained by the Advanced Simple Agent in team play with Meta-IPPO Agent is 0. This is because the Advanced Simple Agent and Meta-IPPO will report errors in 100k episodes of the evaluation process. As a result, we only get 2.46 points before the

Agent	2-Player Self-Play
IGGI	16.189
Internal	10.189
Outer	13.878
VanDenBergh	15.924
Piers	17.027
AdvancedSimple	15.147
Rainbow	19.08
IPPO	17.06
MAPPO	20.54
Meta-IPPO	12.82
Meta-MAPPO	12.72

Table 4.1: The average score was evaluated for 100k episodes in the self-play setting.



(a) Average evaluating scores of Meta-MAPPO. (b) Average step rewards of Meta-MAPPO.

Figure 4.4: Training procedure of Meta-MAPPO.

evaluation fails, which we consider an invalid score. This error is due to the fact that the Advanced Simple Agent cannot respond effectively to Meta-IPPO policy actions when it is the early hand player.

We calculated the average scores in the ad-hoc team setting for different conditions based on the figure 4.5 and got the table 4.2. As we can see in the table 4.2, the 'Ad-hoc Teams' column indicates the average of all scores from all 2-player cross-play Hanabi game evaluations that include this agent. The column 'without self-play' shows the average score of all evaluations that include this agent by removing the self-play score. Finally, the 'only rule-based agents' column is the average of all scores obtained by this agent play with all other rule-based agents during cross-play evaluations. For example, for the IGGI Agent, the 'Ad-hoc Teams' score is the average of all scores IGGI obtained in Figure, while the score 'without self-play' is the average of all IGGI scores without IGGI self-play

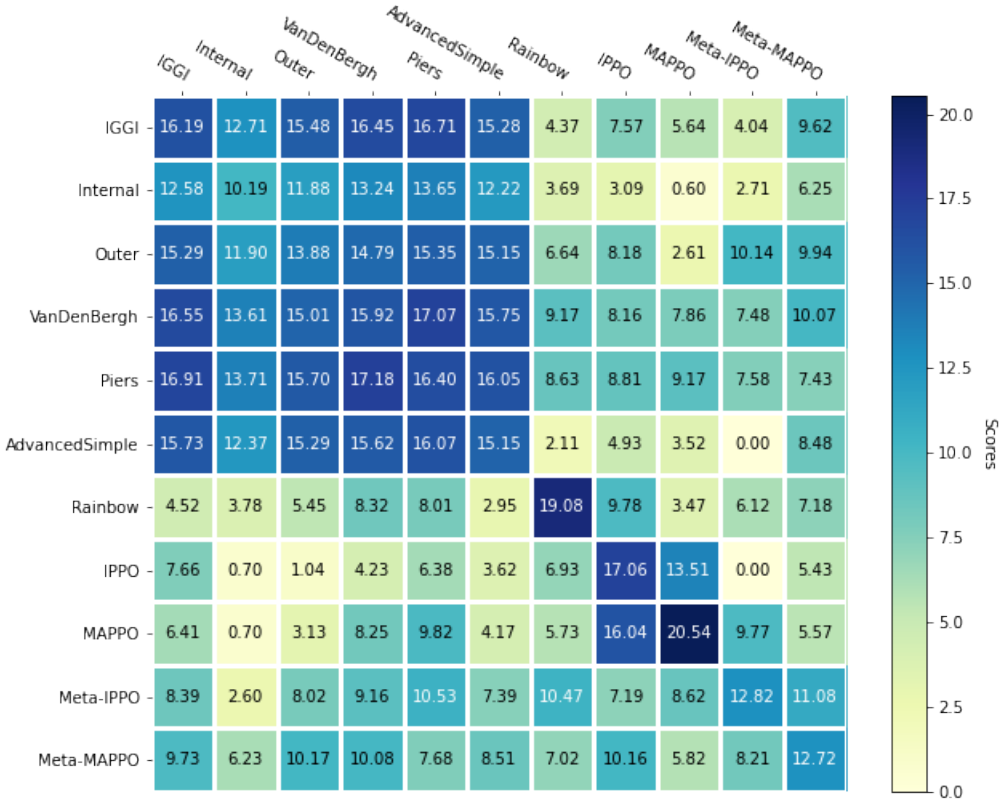


Figure 4.5: Ad-hoc results for two players.

score. And 'only rule-based agents' score for IGGI is the average of the scores of all rule-based agents that played with the IGGI agent.

Agent	Ad-hoc Teams	without self-play	only rule-based agents
IGGI	11.325	11.082	15.444
Internal	8.02	8.10	12.551
Outer	10.716	10.558	14.52
VanDenBergh	12.094	11.903	15.563
Piers	12.326	12.122	15.891
AdvancedSimple	10.017	9.761	14.971
Rainbow	6.825	6.212	5.628
IPPO	7.165	6.671	5.364
MAPPO	7.188	6.521	5.157
Meta-IPPO	7.616	7.342	7.095
Meta-MAPPO	8.446	8.233	8.682

Table 4.2: The average score was evaluated for 100k episodes in the ad-hoc setting.

In the contrast experiments, we trained multiple groups of learning agents based on IPPO and MAPPO algorithms. We use different numbering to distinguish between agents using the same strategy trained with different hyper-parameters and methods. In this case, we want to compare the performance of various agents using the same strategy in the Ad-hoc setting. We also used the

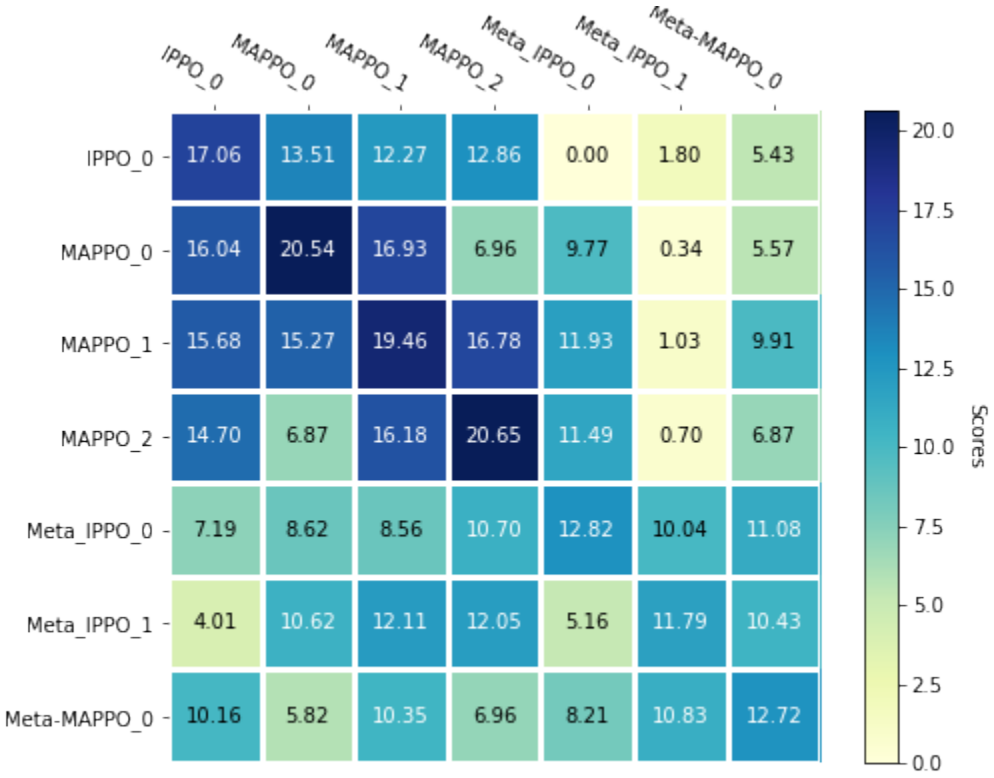


Figure 4.6: Ad-hoc cross-table for learning agents.

cross-play method to evaluate all seven learning agents, and the final results are shown in the figure 4.6.

Agent	Ad-hoc Teams	without self-play
$IPPO_0$	10.054	9.470
$MAPPO_0$	10.528	9.694
$MAPPO_1$	12.805	12.250
$MAPPO_2$	11.059	10.260
Meta-$IPPO_0$	8.889	8.562
Meta-$IPPO_1$	6.992	6.592
Meta-$MAPPO_0$	8.796	8.469

Table 4.3: The average score for learning agents in the ad-hoc setting.

Similarly, on the basis of the results in Figure 4.6, we calculated the average scores of Ad-hoc Teams and the average scores of ad-hoc teams without self-play and obtained the results as shown in the table 4.3. Again, the table indicates that IPPO and MAPPO score higher without meta-learning than the agents with the meta-learning method.

Chapter 5

Discussion and Future Work

5.1 Discussion

In the 2-player Hanabi game, existing reinforcement learning-based learning agents can easily outperform the rule-based agent in the self-play setting. In the table 4.1, we find that most learning agents have the potential to exceed 20 points in the self-play setting. However, as we mentioned before, both algorithms using meta-learning, meta-IPPO, and meta-MAPPO, did not meet our expected performance in the self-play setting. Possible causes of this problem are:

- Meta-RL uses a recurrent policy to memorize. It is hard to train and get the recurrent policy to converge. (auxiliary losses)
- Training recurrent policy is time-consuming, and we only trained it for 300M environment steps for almost a week.
- The meta-RL methods mentioned by Wang *et al.*[19] are not necessarily perfectly applicable to the on-policy PPO algorithm, and it is very restrictive to the specific environment.
- In the learning phase for meta-RL in both meta-IPPO and meta-MAPPO, after pre-training in the self-play setting, we also performed meta training. The only two agents in the training agents pool for meta-training were a trained MAPPO agent and a trained IPPO agent.

By analyzing the Table 4.2, we found that the learning agents in the experiment still could not outperform the rule-based agents in the ad-hoc setting. The rule-based agent performed very consistently, reaching about 15 points when the teammate agent was also rule-based. However, in analyzing the learning agents, the score difference between IPPO and MAPPO in the ad-hoc setting

is minimal, which indicates that MAPPO's CTDE framework does not have any advantage over IPPO for playing with unseen agents. Compared to the Rainbow agent, we found that the score advantage of both IPPO and MAPPO mainly comes from playing with other learning agents. As for the two agents using the meta-learning method, the scores are lower in the self-play setting, while it performs better than the other learning agents in playing with other agents. Especially in the case of playing only with rule-based agents, the score is significantly higher than the other three learning agents(Rainbow, IPPO, MAPPO).

In the case of the analysis on all learning agents, it is combined with the figure 4.6 and table 4.3. We find that IPPO and MAPPO can obtain higher scores when playing with agents using the same strategy, while the agents using the meta-learning method perform similarly when playing with any agent.

In both experiments, we found that MAPPO performs very similarly as an early hand player and a late hand player because MAPPO uses a shared policy for each actor in training. In contrast, since the policy for each actor is independent in IPPO, the policy used as an early hand player and late hand player may be completely different.

5.2 Conclusion

In this work, we evaluate several different algorithms in the ad-hoc setting of the Hanabi challenge. We also propose several approaches to improve the current reinforcement learning algorithms and obtain better performance when collaborating with unseen agents. Using a meta-reinforcement learning approach, we present meta-MAPPO and meta-IPPO. Although it is still not possible to outperform the best rule-based agent in the ad-hoc teams setting, this approach still shows the improvement of the learning agent in the ad-hoc setting relative to its baseline algorithm.

As we evaluated all the agents, we have shown that the learning agents using the *centralized-training-decentralized-execution* framework do not perform better than those using independent strategy. In other words, CTDE can only improve the performance when the learning agent is playing with other agents with pre-coordination. When cooperating with unseen agents, the communication between the agents established by the CTDE approach does not improve performance. Although the difference in scores between IPPO and MAPPO in ad-hoc teams setting is not significant. Through detailed score statistics, we can also find that the score difference between IPPO's early-hand and late-hand policies in the cross-play situation is substantial. Therefore, MAPPO with

a shared policy is more stable in an actual deployment environment. This conclusion also applies to IPPO and MAPPO using meta-learning.

5.3 Future Work

Our ultimate goal is to find an approach for the learning agent to outperform the best rule-based agent in the ad-hoc setting and be capable of playing with human players. However, there are still many points that can be improved based on our experiments:

- Our trained meta-learning model is not the best, and we need more parameter tuning and training time to get a better performing meta-RL agent.
- In the continual training, we used only one MAPPO agent and one IPPO agent to train with. We need more different agents for continuous training to generalize the Meta-RL model.
- Model-based meta-learning may not be optimal for the on-policy policy gradient algorithm. Instead, we can use the optimization-based meta-learning(e.g., Model-Agnostic Meta-Learning[17]) approach on the off-policy algorithms, such as Soft Actor-Critic[8] and Deep Deterministic Policy Gradient[33].

In addition, due to the unique property of the Hanabi game, we can further look into the ad-hoc teams challenge when the number of players increases to 3-5. However, we think it is feasible to expand the 2-player Hanabi game to 3-5 players. Suppose an approach proves to perform well on a 2-player Hanabi game. In that case, the more significant challenge in a 3-5 player Hanabi game is the increased computational complexity due to the exponential growth of agent interactions and larger observation space. Therefore, researching a 2-player Hanabi game in the ad-hoc setting can be more applicable than the Hanabi game with 3-5 players.

References

[1] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 7587 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Computational science;Computer science;Reward Subject_term_id: computational-science;computer-science;reward.

[2] Johannes Heinrich and David Silver. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. *arXiv:1603.01121 [cs]*, June 2016. arXiv: 1603.01121.

[3] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi Challenge: A New Frontier for AI Research. *Artificial Intelligence*, 280:103216, March 2020. arXiv: 1902.00506.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, August 2017. arXiv: 1707.06347.

[5] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. *arXiv:1502.05477 [cs]*, April 2017. arXiv: 1502.05477.

[6] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation Matters in Deep RL: A Case Study on PPO and TRPO. September 2019.

[7] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *arXiv:1710.02298 [cs]*, October 2017. arXiv: 1710.02298.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, August 2018. arXiv: 1801.01290.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. Bandiera_abtest: a Cg_type: Nature Research Journals Number: 7540 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Computer science Subject_term_id: computer-science.

[10] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *arXiv:1706.02275 [cs]*, March 2020. arXiv: 1706.02275.

[11] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games. *arXiv:2103.01955 [cs]*, March 2021. arXiv: 2103.01955.

[12] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv:1706.05296 [cs]*, June 2017. arXiv: 1706.05296.

[13] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-Learning in Neural Networks: A Survey. *arXiv:2004.05439 [cs, stat]*, November 2020. arXiv: 2004.05439.

[14] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching Networks for One Shot Learning. *arXiv:1606.04080 [cs, stat]*, December 2017. arXiv: 1606.04080.

[15] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical Networks for Few-shot Learning. *arXiv:1703.05175 [cs, stat]*, June 2017. arXiv: 1703.05175.

[16] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot Learning with Memory-Augmented Neural Networks. *arXiv:1605.06065 [cs]*, May 2016. arXiv: 1605.06065.

[17] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. *arXiv:1703.03400 [cs]*, July 2017. arXiv: 1703.03400.

[18] Alex Nichol, Joshua Achiam, and John Schulman. On First-Order Meta-Learning Algorithms. *arXiv:1803.02999 [cs]*, October 2018. arXiv: 1803.02999.

[19] Jane X. Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z. Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv:1611.05763 [cs, stat]*, January 2017. arXiv: 1611.05763.

[20] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. RL$ˆ2$: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv:1611.02779 [cs, stat]*, November 2016. arXiv: 1611.02779.

[21] 3.1 The Agent-Environment Interface.

[22] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *arXiv:1911.10635 [cs, stat]*, April 2021. arXiv: 1911.10635.

[23] Frans A. Oliehoek. Decentralized POMDPs. In *Reinforcement Learning: State of the Art*. Springer, 2012.

[24] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is Independent Learning All You Need in the StarCraft Multi-Agent Challenge? *arXiv:2011.09533 [cs]*, November 2020. arXiv: 2011.09533.

[25] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, October 2018. arXiv: 1506.02438.

[26] Matthew Botvinick, Sam Ritter, Jane X. Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement Learning, Fast and Slow. *Trends in Cognitive Sciences*, 23(5):408–422, May 2019.

[27] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[28] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv:1412.3555 [cs]*, December 2014. arXiv: 1412.3555.

[29] Rodrigo Canaan, Haotian Shen, Ruben Rodriguez Torrado, Julian Togelius, Andy Nealen, and Stefan Menzel. Evolving Agents for the Hanabi 2018 CIG Competition. *arXiv:1809.09764 [cs]*, September 2018. arXiv: 1809.09764.

[30] Joseph Walton-Rivers, Piers R. Williams, Richard Bartle, Diego Perez-Liebana, and Simon M. Lucas. Evaluating and Modelling Hanabi-Playing Agents. *arXiv:1704.07069 [cs]*, April 2017. arXiv: 1704.07069.

[31] Hirotaka Osawa. Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information. page 7.

[32] Mark van den Bergh, W.A. Kosters, and F.M. Spieksma. *Hanabi, a co-operative game of fireworks*. PhD thesis, Universiteit Leiden, July 2015.

[33] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, July 2015. arXiv: 1509.02971.

Appendix A

Implementation Details

A.0.1 Implementation

As Hanabi is a turn-based card game, all of our trajectories that need to be stored in the memory buffer are turn-based. This means that all agents get the same accumulated reward $R_i = r_t^1 + r_{t+1}^2 + \cdots + r_{t+i-1}^i$ in one turn, where t is the total timestep and i is the agent number. In the trainers of both MAPPO and IPPO, we collected the turn-based information, including reward, action, observation, and value for each agent. Our MAPPO implementation is based on the original MAPPO implementation. We not only implemented the meta-learner and the meta-training method but also implemented our evaluation method in the ad-hoc setting.

As for the meta reinforcement learning method, we implemented the recurrent layers, including the GRU and LSTM. However, we only used the GRU policy in the experiments since the LSTM policy did not perform as well as the GRU policy in our experiments and could not converge.

A.0.2 Hyperparameters

The table A.1-A.7 describes in detail the hyperparameters of all learning agents trained in our experiments.

hyperparameters	value
batch size	num envs × buffer length × num agents
hidden size	512
number of MLP layers	2
use centralized value function	true

Table A.1: Hyperparameters used in MAPPO_0.

hyperparameters	value
batch size	num envs × buffer length × num agents
hidden size	128
number of MLP layers	2
use centralized value function	true

Table A.2: Hyperparameters used in MAPPO_1.

hyperparameters	value
batch size	num envs × buffer length × num agents
hidden size	128
number of MLP layers	1
use centralized value function	true

Table A.3: Hyperparameters used in MAPPO_2.

hyperparameters	value
batch size	num envs × buffer length × num agents
hidden size	512
number of MLP layers	2
use centralized value function	false

Table A.4: Hyperparameters used in IPPO_0.

hyperparameters	value
batch size	num envs × buffer length × num agents
RNN hidden size	512
number of MLP layers	2
use centralized value function	true

Table A.5: Hyperparameters used in Meta-MAPPO_0.

hyperparameters	value
batch size	num envs × buffer length × num agents
RNN hidden size	512
number of MLP layers	1
use centralized value function	false

Table A.6: Hyperparameters used in Meta-IPPO_0.

hyperparameters	value
batch size	num envs × buffer length × num agents
RNN hidden size	128
number of MLP layers	1
use centralized value function	false

Table A.7: Hyperparameters used in Meta-IPPO_1.