Title
Meta-Learning Action Conventions in Ad-Hoc Hanabi

Author
Sarmasi, Aron

Publication Date
2021

Peer reviewed|Thesis/dissertation

Meta-Learning Action Conventions in Ad-Hoc Hanabi

By

ARON SARMASI
THESIS

Submitted in partial satisfaction of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Chair's Name, Chair (or Co-Chair)

Member's Name

Member's Name

Committee in Charge

2022

ACKNOWLEDGMENTS

I would like to thank my advisor Joshua McCoy for his support, all my coauthors for the long hours, and my parents for making it all possible.

ABSTRACT

The purpose of this thesis is to investigate how we can learn action conventions by observation in an ad-hoc context. We argue that the game Hanabi in particular is a promising application for studying this topic because it distills the problem to its core components, while presenting a small computational overhead. To facilitate research in this direction, we have compiled the Hanabi Open Agent Dataset (HOAD), consisting of neural replicas of the majority of contemporary Hanabi agents developed prior to this work. We first validate that HOAD is appropriate to use in meta-learning studies by demonstrating that HOAD agents use diverse, high quality strategies, and then we show that the popular meta-learning algorithm MAML can be used to train an ad-hoc learner that performs superior to random and naive baselines. Finally, we corroborate recent findings that MAML doesn't benefit from its inner learning loop after a sufficient number of training epochs.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Human beings regularly learn action conventions by observation in ad-hoc contexts. For example, every time we meet a new group of people, whether a new friend group or a new culture, there is always a period of adjustment, where we pick up on the unsaid conventions that underpin interactions within that group. Furthermore, for us this period can be remarkably short–a single "episode" to blend into a new friend group, a few tens to blend into a new workplace, and a few hundred to blend into a new culture. As various artificial intelligences (chat-bots, assistants, self-driving cars, game AIs, etc) become more prevalent in our lives, the need for these AIs to seamlessly blend into their social environments continues to grow. On the one hand, forcing users to adapt to the idiosyncrasies of the AI places a high burden on the individual. On the other, AI developers are often not a representative sample of the general public, and these undue burdens are likely to affect minority populations more significantly. An experience that can quickly tailor to the user will thus provide both a market advantage, and promote greater equity.

The game Hanabi is a particularly good application for studying this problem because it distills the problem of ad-hoc interaction to its core components, allowing researchers to focus on solving the problem in isolation. Games consist of a few tens of actions, and the game representation is fairly light, but the space of action conventions is huge. Moreover, there is a large subset of conventions which are substantially different from one another, but still lead to strong game-play. This particular feature underpins the core difficulty of

the problem, and is a direct analogy to how different groups of people have different (but still effective) conventions for interaction. As a result of these properties, research interest in Hanabi has spiked in recent years. Multiple groups have developed agents to play the game, and have provided valuable insights into various aspects of game-play (section 2.3.1). However, to get to the very heart of the ad-hoc problem, it is necessary for the agents from these disparate groups to be able to play with each other. This is the purpose of the Hanabi Open Agent Dataset (HOAD); to ease and simplify the problem of studying the ad-hoc learning of action conventions in Hanabi. Specifically, HOAD consists of neural replicas of the majority of contemporary Hanabi agents developed prior to this work. While there are a few agents developed concurrently that are not yet present, HOAD makes it easy to add additional agents. This extensibility is documented in section 3.5.

Although HOAD is flexible and can be used to study a wide array of problems involving Hanabi agents, in this work we focus specifically on the ad-hoc problem. We formulate this problem as "given 10 games of Hanabi played by an unfamiliar agent, learn the action distribution of that agent". Once an imitator is learned, it is used to play games with the unfamiliar agent; the resulting score average serves to evaluate the quality of the imitator. In section 3.6 we show that meta-learning is a promising approach for solving the ad-hoc problem. Since the agents in HOAD are both high quality and use diverse strategies (section 2.3.2), they can be used to meta-learn a generalist agent, which can then be specialized to imitate the previously unseen agent using 10 games of replay data. Finally, we conclude this study by corroborating the recent finding that the popular meta-learning algorithm MAML doesn't benefit from its inner learning loop after a sufficient number of training epochs. In other words, once the meta-learning has progressed far enough, the generalist agent no longer gains any advantage from being trained on the 10 games of the previously unseen agent.

The bulk of this thesis outside of this introduction is contained in two previously published articles. The first, titled "HOAD: The Hanabi Open Agent Dataset" is an exposition of

Hanabi and the eponymous dataset, and was published as an extended abstract in the 2021 Autonomous Agents and Multiagent Systems (AAMAS) conference. The second, titled "Meta-Learning a Solution to the Hanabi Ad-Hoc Challenge", expands on why HOAD is a good test-bed for meta-learning algorithms, and presents our findings after applying MAML to HOAD; it was published in the 2021 Foundations of Digital Games (FDG) conference. The majority of the work in both publications is due to the first author of the articles.

CHAPTER 2

HOAD: THE HANABI OPEN AGENT DATASET

2.1 Abstract

In this work we present the Hanabi Open Agent Dataset (HOAD)— meant to address the current lack of Hanabi datasets, HOAD is an easily extensible, open-sourced, and comprehensive collection of existing Hanabi playing agents, all ported to the Hanabi Learning Environment (HLE). We give a description and analysis of each agent's strategy, and we also show cross-play performance between all the agents, demonstrating both their high quality and diversity of strategy. These properties make HOAD especially well suited to studies involving meta-learning and transfer learning. Finally, we describe in detail an easy way to add new agents to HOAD regardless of the origin codebase of the agent and make our code and dataset publicly available at https://github.com/aronsar/hoad.

2.2 Introduction

Hanabi [5] is a tabletop card game for 2-5 players[1], notable for its unique combination of partial observability, cooperation, stochasticity, and implicit communication. Recently proposed as a challenge domain [4] to provide a sophisticated yet well-defined set of challenges for artificial intelligence practitioners, the game is of rising interest to the research community.

In a game of Hanabi, the players work together to assemble five piles of cards, where each pile is a different color, and consists of cards numbered 1 through 5, in that order.

[1] in this work we consider only the two-player scenario for simplicity, leaving 3-5 players to future work

The defining feature of the game is that players only see their teammates' cards, and not their own. On their turn, a player may either give a teammate a hint about the color or value of their cards (a limited resource), discard a card (doing so regains a hint, but there are a limited number of copies of each card), or play a card they believe is playable. See [30] for the complete rules. To be successful, a group of players must correctly interpret each others' explicit communication—through hints—as well as each others' implicit communication—through playing cards, discarding, or even hinting (e.g. the finesse play [40]). This results in a multiplicity of optimal and near-optimal strategies, each corresponding to a communication schema.

This aspect of the game is one of the most interesting from a research perspective. Indeed, it recently inspired the Ad-Hoc Challenge, [4], the crux of which is to figure out the communication strategy of a held-out agent given only ten of its games, and then successfully play games with it. However, to the best of our knowledge, there does not exist a dataset of Hanabi playing agents that can facilitate this kind of research. We thus propose the Hanabi Open Agent Dataset (HOAD), a compilation of different agents from different sources, most using conventions typically seen in human play, and all operating within the same environment and using a binarized game state representation specifically designed for neural learning. The dataset allows pairwise play between any two agents, offers an easy way to add additional agents from most codebases, and also gives access to the Dopamine reinforcement learning framework [7], which we plan to use in future work to further improve and extend the HOAD agents.

2.3 The Hanabi Open Agent Dataset

To create HOAD, we conducted a comprehensive review of existing implementations of agents that play Hanabi and ported those that met our criteria. We looked specifically for agents that had codebases available online, scored reasonably well in self-play (above 10 points), and employed strategies sufficiently different from one another, such that when two different

Table 2.1: We recorded 500,000 games of each original agent into the HLE representation, and then used them to learn multilayer perceptron (MLP) imitators of each agent.

Agent	Original Self Play Score	Imitator Self Play Score	Imitator Accuracy
Simplebot	16.9	16.8	99.7
Valuebot	19.8	18.0	92.0
Holmesbot	20.8	14.7	90.3
Outer	14.5	14.1	66.7
Iggi	17.0	16.2	90.9
Piers	17.3	15.9	85.8
Rainbow	18.5	18.1	77.5
Van-Den-Bergh	14.0	10.5	81.2

agents played one another they performed significantly worse than either did in self-play. As a result, the variation of strategy among the agents is ensured. This disqualified a number of derivate works that were slight modifications on existing agents, as well as the Actor Critic Hanabi Agent (ACHA) [23], which does not have a public codebase, and the agents proposed by [6], which released their codebase concurrently to this work. The FireFlower [38] and BAD [13] agents were also considered, but were not included due to technical difficulties; we plan to add them in future work.

Although some authors of Hanabi playing agents have published multiple agents using the same framework, in general, two arbitrary agents taken from different codebases will not be able to play one another because of differences in implementation in how the agents represent the game state. One naive way to enable agents from multiple different codebases to play one another is to rewrite the game logic of every agent using the same environment and game state representation. This however is prohibitively time-consuming, as some agents are comprised of several thousand lines of game logic. Another method would be to observe the game states of each agent in its native representation and save all the game states to some common representation. In our experience, this was as difficult as the first approach due to the variation in implementation among authors and the size and complexity of the game state (there are 658 binary variables in the game state representation of a 2-player game).

The workaround that made HOAD possible (and which is responsible for its ease of extensibility) is to observe the starting deck order and the actions taken by agents in their native environments. These observations are then used to recreate the games in the Hanabi Learning Environment (HLE). Observing the starting deck order and the actions taken is a much simpler task because the set of legal actions is small (≤ 20 in a 2-player game), and the ordering of the deck is typically known by the respective game engine at the start of the game. Once replay data has been gathered for all the agents in the HLE representation, it is possible to train a neural network to imitate each of the original agents. The accuracy of these imitators is presented in Table 2.1.

2.3.1 HOAD Agent Strategy Summaries

We present a summary of each agent in HOAD below, including only necessary detail. A compilation of commonly employed strategies, used both by human players and HOAD agents, can be found at [40].

Simplebot [25] – Plays only cards which it has enough knowledge about to know they are playable, and gives hints only about playable cards, preferring color hints over value hints. It uses an oldest first discarding strategy.

Valuebot [25] – Same as Simplebot, but before playing, checks to see if the next player is about to discard a valuable card (i.e. the last copy of a card).

Holmesbot [25] – Extends Valuebot by including the use of mulligans and by adding additional inference capabilities. Specifically, card knowledge from hints, the discard pile, and other players' hands are used to make deductions about the agent's hand.

Iggi [36] – Similar logic to Simplebot but prefers value hints over color hints and prefers discarding unplayable cards over oldest first.

Outer [36] – Similar to Iggi, but prefers discarding over hinting, and uses more randomness in its hinting and discarding logic; this results in significantly reduced imitation accuracy and is also likely the reason for lower published score.

Piers [36] – Extends Iggi by including the use of mulligans (but not as deterministically as

Table 2.2: Pairwise play scores are produced by playing each MLP imitator agent with every other for 500 games.

First player	Simplebot	Valuebot	Holmesbot	Outer	Iggi	Piers	Rainbow	Van-Den-Bergh
Simplebot	16.8	15.7	12.8	0.0	4.1	1.1	1.7	0.2
Valuebot	15.2	18.0	17.6	0.0	3.8	1.3	2.0	0.0
Holmesbot	11.2	18.3	14.7	0.0	1.4	0.6	0.5	0.0
Outer	0.0	0.0	0.0	14.1	1.0	4.1	6.2	9.0
Iggi	3.9	3.8	1.8	1.8	16.2	11.8	2.7	6.0
Piers	1.8	0.5	0.2	7.2	10.3	15.9	5.5	9.4
Rainbow	0.3	2.0	0.3	6.6	4.0	5.6	18.1	2.6
Van-Den-Bergh	0.0	0.2	0.0	10.6	4.7	8.3	2.7	10.5

Holmesbot), and some additional logic to avoid discarding valuable cards.

Van-der-bergh [36] – Makes some risky plays if they have high likelihood of success and there are remaining mulligans. Prioritizes discarding over hints, gives hints about useless cards, and attempts to maximize transmitted information.

Rainbow [4] – This agent tends to hint for rank instead of color. Conditional action probabilities may be found in the appendix of [4].

2.3.2 Discussion of Imitator Agent Pairwise Scores

Since the same observation can be passed in to any of the imitator agents, the imitator agents make it possible to play games with agents originating from two different codebases; this is necessary to evaluate how different the players are from each other. Table 2.2 shows the average score of each imitator agent playing 500 games with every other imitator agent. As expected, when an agent is paired with itself, it typically achieves a much higher score than if it were paired with any other agent—this corresponds to the high scores on the diagonals and the relatively low scores on the off-diagonals. This confirms the intuition that agents must use a similar strategy when paired with one another, else risk miscommunicating, losing

all three lives, and scoring zero points.

An interesting feature of Table 2.2 warranting discussion is the high scores achieved by certain combinations of agents. Two groups of agents that play relatively successfully with each other are Simplebot – Valuebot – Holmesbot, and Iggi – Piers – Outer – Van-der-bergh – Rainbow. All the agents in each of these two groups belong to the same codebase, so the high scores are likely due to the authors of the two codebases reusing logic between their agents. The exception to this is the Rainbow agent, which was produced using reinforcement learning, and so it is surprising that it performs so well with Walton-Rivers' agents. Our best explanation is that both agents reportedly prefer value hints and can presumably also respond well to game states where value hints have been given.

CHAPTER 3

META-LEARNING A SOLUTION TO THE HANABI AD-HOC CHALLENGE

3.1 Abstract

In this work we demonstrate that the First Order Model Agnostic Meta Learning (FOMAML) algorithm trained on the Hanabi Open Agent Dataset (HOAD) results in a model that is able to outplay both a naive MLP baseline, as well as a randomly selected partner in the Hanabi Ad-Hoc Challenge, in both low-shot and zero-shot setups. We first show that HOAD is well suited for the meta-learning task because its agents are high quality and utilize diverse strategies, thereby confirming that MAML is generalizing, and not memorizing agent strategies. We then detail our application of FOMAML to the cooperative decision making problem Hanabi entails, and we also provide evidence supporting recent results that the task update of MAML gives little to no test time performance boost. The pretrained models and game data are made available online at *https://github.com/aronsar/hoad*.

3.2 Introduction

Hanabi [5], a tabletop card game for 2-5 players[1], is a challenging application for agent modeling in the multi-agent domain due to its unique combination of partial observability, cooperation, stochasticity, and implicit communication. Recently proposed as a challenge

[1] in this work we consider only the two-player scenario for simplicity, leaving 3-5 players to future work

domain [4] to provide a sophisticated yet well-defined set of challenges for artificial intelligence practitioners, Hanabi is of rising interest to the research community. Of particular interest is the Ad-Hoc Challenge, [4] which aims to push research towards agents that are capable of cooperatively interacting with other agents or even humans with whom they are unfamiliar. These problems inherent to Hanabi have also been studied in other contexts, both commercial and industrial, such as self-driving cars, and human-robot cooperation in the manufacturing setting.

In a game of Hanabi, the players see the cards of all their teammates, but not their own. Thus, the defining element of the game is the communication of information about other players' cards either explicitly using hints (a limited resource), or implicitly through playing cards, discarding, or even hinting (e.g. the finesse play [40]). Because of this, a group's success or failure depends on that group's ability to communicate and to understand one another—e.g. if one player makes a finesse play, and another player doesn't pick up on it, the group loses one of their three lives. The result of this mechanic is that unlike other games such as Go [32] and StarCraft[21], Hanabi has a multiplicity of optimal and near-optimal strategies, each corresponding to a different communication schema. While a number of works have focused on creating agents that get high scores in self-play [23] [36] [8], here we instead focus on the more difficult Ad-hoc Challenge, which, to the best of our knowledge, has not yet been studied. In the Ad-hoc Challenge, an agent is asked to play successfully with a held-out (ie. previously unseen) agent after observing only 10 of its self-play games. We break this task down into two subtasks: 1) given 10 of an agent's games, build a model of that agent, and 2) use this model to search over possible future states and pick an action that maximizes the expectation of future reward. In this work we focus on the first of these two subtasks, leaving the second to future work.

Building a model of an agent-based on only 10 of its games is a very difficult task for traditional deep learning algorithms because the 10 games do not provide nearly enough data to cover every aspect of an agent's strategy, and they also don't necessarily provide an example for every game mechanic. For example, some agents in Hanabi Open Agent Dataset

[31] (HOAD) rarely or never give hints, while other agents may be too weak to reach late game, and so may not provide examples of late-game mechanics in action. As a lower bound, we present a naive multilayer perceptron (an MLP) which does exactly this: it is trained to imitate an agent based on only 10 of its self-play games.

Properly addressing the Ad-Hoc Challenge requires an a priori understanding of the game, and of the kind of communication strategies one might employ to play the game. The approach we take in this work is to learn such an understanding using a large number of games played by a variety of different agents (that we call "training agents"). To the best of our knowledge, HOAD is the only dataset of Hanabi playing agents that can facilitate this kind of learning. HOAD is a compilation of different agents from different sources, most using conventions typically seen in human play, and all operating within the same environment and using a binarized game state representation specifically designed for neural learning. The dataset allows pairwise play between any two agents, offers an easy way to add additional agents from most codebases, and also gives access to the Dopamine reinforcement learning framework [7], which we plan to use in future work to further improve and extend the HOAD agents.

To solve the Ad-hoc Challenge, we chose to implement the first-order approximation of Model Agnostic Meta-Learning (FOMAML) [11]. This is a highly generalizable technique that is well-suited to the low-shot learning scenario and has been successfully applied to several problems outside of image recognition, its original application [12] [39]. Despite its popularity however, this is the first time it is applied to the cooperative decision making problem. The goal of MAML is to train a meta-model that is close in parameter space to the optima of many different tasks. During inference, a few gradient updates (hence "low-shot") are all that is needed to obtain good performance on a previously unseen task. In our case, that means that given just a few games of a held-out agent, the meta-model can be updated to accurately imitate this agent. Recently, however, it has been shown that the meta-model itself is already quite strong on its own, without the task update [26]. Our results provide further evidence of this phenomenon, showing that it also occurs in the context of Hanabi,

and using the first-order approximation of MAML. Pseudocode, along with a more detailed explanation of our implementation is provided in Section 3.6.2.

The main contribution of this work is that we show that MAML can be applied to the cooperative decision making problem to train a meta-model on HOAD, which can then be updated on a small set of a held-out agent's games, and then played directly with that held-out agent to achieve superior performance to random partner selection and a naive MLP baseline. We also show that given enough training time, the meta-model actually performs better without the task update step, corroborating similar recent findings.

3.3 Related Works

Agent imitation is a well-established field of research; some early methods include behavioral cloning [28], which is quite brittle and does not easily adapt to new situations, and inverse reinforcement learning, which is very slow to run [27]. Despite increased attention to this field in the past decade [1] [3], few works study agent imitation under all the same constraints that apply to Hanabi—i.e. partial observability, cooperation, stochasticity, and implicit communication, all in the low-shot regime. As such, we discuss those that adopt a subset of these constraints. He et. al. [16] presents models that jointly learn a policy and the behavior of their opponents using deep reinforcement learning; however, they need access to the agent they model during training, and so are also not well suited to the ad-hoc situation. Le et. al. [19] proposes an approach to learning implicit coordination strategies that jointly learns a latent coordination model as well as the agents' individual policies, but this approach too is infeasible in the ad-hoc case, as it requires a large number of examples of agent interactions.

Several other works share our low-shot constraint but are overly specialized to their respective tasks. A seminal work in robotic agent imitation, Generative Adversarial Imitation Learning (GAIL), makes use of a generative adversarial technique that achieves good results on low-shot expert imitation; however, GAIL (and its class of solutions) is best suited to the continuous world of robotic movement, and so is difficult to adapt to Hanabi [17] [37]

[20]. Duan et. al. [9] proposes one-shot imitation learning to solve the task of robot block-stacking, but their neighborhood attention model is ill-suited to the Hanabi game representation, where there is little spatial information. Similarly, [14] [35] [34] [33] are all designed for the low-shot scenario, but their hand-crafted architectures are too focused on the image classification or robot motion applications to be easily adapted to Hanabi.

The most comparable approach to MAML in terms of generality is the Simple Neural AttentIve Learner (SNAIL) [22]; indeed, SNAIL performs comparably or slightly better than MAML when evaluated on Omniglot [18] and Mini-Imagenet [35], and we suspect that it could be a promising baseline to run on HOAD in future work. For this work, however, we preferred MAML's model agnosticism, which allowed us to use a small and simple fully connected network as the policy predictor. In a similar vein, we chose not to use any of the many MAML derivate works, many of which achieve improvements by introducing the kind of hand-designed architectural and algorithmic tweaks that specialize the approach to a specific domain, and which don't transfer well to Hanabi [20] [15] [29]. Of additional interest is Yu and Finn's more recent work using MAML for human imitation in the robot manipulation environment; this work most closely resembles the conceptual underpinnings of our use of MAML [39].

3.4 Rules of the Game

The base game considered here consists of the following materials:

- 50 cards; 10 each of blue, yellow, red, white, and green; of each color there are three 1 valued cards, two 2's, two 3's, two 4's, and only one 5 valued card
- 8 hint tokens, also referred to as information tokens; these have two faces to indicate whether they've been used already
- 3 lives, also referred to as mulligans or danger tokens; these too have two faces to designate their availability

The goal of the game is for the players to collectively form 5 piles of cards, one pile of each color. Each pile must be built in ascending order, starting with the 1 valued card on the bottom, and ending with the 5 valued card on top. At the start of the game, each player is dealt a hand of 5 cards, and the 8 hint tokens and 3 life tokens are all facing up. During play, a player may look at the cards of their teammates, but never their own. Players take legal actions in turns, going around the table in a clockwise fashion. The game ends when any of the below conditions occur:

- If the third life token is flipped face down, the game ends with a score of 0 (this is the most important factor in why it is difficult to play with unfamiliar agents)
- If the 5 piles are completed before the deck runs out, the game ends with a score of 25
- If the deck runs out before the 5 piles are completed, each player gets one last turn, including the one who drew the last card. The final score is equal to the count of the cards in the 5 piles.

According to the official ruleset, no communication may occur during the game other than through the use of hint tokens (explained below). However, the rules also recommend that players interpret this rule in a way that suits them best. Although there is some controversy in the research community over what exactly should be considered acceptable communication [10], in this work we have restricted the agents to only be able to pass information to their teammates through their choice of legal action on their own turn—the same as proposed in [4]. The possible actions in a game are:

- Giving a hint: a player can either give a color hint or a value hint. To give a color hint, pick another player, pick a color, and then point out all cards of that color in their hand. To give a value hint, do the same, but for a card value instead of a color. Giving a hint costs a hint token, and is only a legal action if there are remaining face up hint tokens. No empty hints may be given, i.e. one may not tell a player what is not in their hand.

- Playing a card: a player may choose a card in their hand and play it. To play a card, attempt to place it on one of the five piles. If successful, draw another card from the deck and pass the turn. If the card can't currently be placed on any of the piles, discard the card, flip a life token face down, and draw another card from the deck. Note: if a 5 is successfully played, flip a hint token face up if possible.

- Discarding a card: a player can choose a card in their hand and place it in the discard pile, flipping a hint token face up, and then drawing a card from the deck (note: players may look through the discard pile at any time). This is only a legal action if there are 7 or fewer hint tokens face up.

3.5 The Hanabi Open Agent Dataset

The Hanabi Open Agent Dataset consists of Hanabi playing agents that have codebases available online, score reasonably well in self-play (above 10 points), and employ strategies sufficiently different from one another, such that when two different agents play one another they perform significantly worse than either does in self-play. As a result, the variation of strategy among the agents is ensured. For this reason, some existing agents are excluded from the dataset. These include a number of derivate works that were slight modifications on existing agents, as well as the Actor Critic Hanabi Agent (ACHA) [23], which does not have a public codebase, and Caanan's agents [6], released concurrently to HOAD.

Canaan's agents specifically bear further discussion, as they too are motivated by the Hanabi Ad-Hoc Challenge. These rule-based, genetically evolved agents are created using MAP-Elites [24], a Quality Diversity algorithm that parameterizes the agents' risk aversion and communicativeness, allowing the user to generate agents with a wide array of different strategies. The approach isn't a panacea, however. Agents close in parameter space overlap significantly in action distribution, and achieve high scores in cross-play. Additionally, the current implementation of the approach doesn't explore the full space of possible Hanabi strategies (focusing only on risk aversion and communicativeness). Nevertheless, an imme-

Table 3.1: We recorded 500,000 games of each original agent into the HLE representation, and then used them to learn multilayer perceptron (MLP) imitators of each agent.

Agent	Original Self Play Score	Imitator Self Play Score	Imitator Accuracy
Simplebot	16.9	16.8	99.7
Valuebot	19.8	18.0	92.0
Holmesbot	20.8	14.7	90.3
Outer	14.5	14.1	66.7
Iggi	17.0	16.2	90.9
Piers	17.3	15.9	85.8
Rainbow	18.5	18.1	77.5
Van-Den-Bergh	14.0	10.5	81.2

diate next step for the HOAD project should be to add support for Canaan's agents, as the ability to create agents in a structured, parameterized way is very useful in the context of the Ad-hoc Challenge.

One of the advantages of using HOAD is its ease of extensibility. Although some authors of Hanabi playing agents have published multiple agents using the same framework, in general, two arbitrary agents taken from different codebases will not be able to play one another because of differences in implementation in how the agents represent the game state. To circumvent the burden of reimplementing every new agent, the authors of HOAD observe the starting deck order and the actions taken by agents in the native environments, and then use those observations to recreate the games in the Hanabi Learning Environment (HLE) (Section 3.5.2). Observing the starting deck order and the actions taken is a much simpler task because the set of legal actions is small ($\leq$ 20 in a 2 player game), and the ordering of the deck is typically known by the respective game engine at the start of the game. Once replay data has been gathered for all the agents in the HLE representation, it is possible to train a neural network to imitate each of the original agents. The self-play scores of these imitators, as well as their accuracies (ie. how well the actions of the imitators match up with the actions of the original agents), are presented in table 3.1.

3.5.1 Implementation Details

The workaround technique, described above, was used to generate 500,000 games per original agent, all in the HLE representation. These games were then used to train imitator agents for each original agent. The imitator architecture consisted of: layer sizes of 1024, 1024, 512, 512, 512, and 256, batch normalization at every layer, and a dropout rate of 0.5. Each network was trained for 5 – 10 epochs, until convergence.

3.5.2 Hanabi Learning Environment

The Hanabi Learning Environment (HLE) [4] is a framework published by DeepMind in 2019 consisting of a fast gameplay engine implemented in C, an easy to use Python API, and integration with the Dopamine reinforcement learning framework. During play, the game state (as seen by the observing agent without revealing hand information) can be accessed as a binarized vector. This is the representation used by all the imitator agents in HOAD, and all the ad-hoc agents described in this work. We use the Dopamine framework only to reproduce the Rainbow agent, although it does present an interesting opportunity for future research, specifically to improve and extend the other HOAD agents.

3.5.3 HOAD Agent Strategy Summaries

We present a summary of each agent in HOAD, including only necessary detail, and also offering explanation of the imitation accuracies and self-play scores in Table 3.1. A compilation of commonly employed strategies, used both by human players and HOAD agents, can be found at [40].

Simplebot [25] – Plays only cards which it has enough knowledge about to know they are playable, and gives hints only about playable cards, preferring color hints over value hints. It uses an oldest first discarding strategy. Due to its simplicity and high degree of determinism, Simplebot is easy to imitate.

Valuebot [25] – Same as Simplebot, but before playing, checks to see if the next player is

about to discard a valuable card (i.e. the last copy of a card). Also, it is deterministic, and easy to imitate.

Holmesbot [25] – Extends Valuebot by including the use of mulligans and by adding additional inference capabilities. Specifically, card knowledge from hints, the discard pile, and other players' hands are used to make deductions about the agent's hand. Still deterministic, and easy to imitate because all the information it uses is present in the HLE representation.

Iggi [36] – Similar logic to Simplebot but prefers value hints over color hints and prefers discarding unplayable cards over oldest first. Highly deterministic, easy to imitate.

Outer [36] – Similar to Iggi, but prefers discarding over hinting, and uses more randomness in its hinting and discarding logic; this results in significantly reduced imitation accuracy and is also likely the reason for lower published score.

Piers [36] – Extends Iggi by including the use of mulligans (but not as deterministically as Holmesbot), and some additional logic to avoid discarding valuable cards. The lower determinism explains the slight drop in imitation accuracy.

Van-der-bergh [36] – Makes some risky plays if they have high likelihood of success and there are remaining mulligans. Prioritizes discarding over hints, gives hints about useless cards, and attempts to maximize transmitted information.

Rainbow [4] – This agent tends to hint for rank instead of color. Conditional action probabilities may be found in the appendix of [4].

3.5.4 Discussion of Imitator Agent Pairwise Scores

Since the same observation can be passed in to any of the imitator agents, the imitator agents make it possible to play games with agents originating from two different codebases; this is necessary to evaluate how different the players are from each other. Table 3.2 shows the average score of each imitator agent playing 1000 games with every other imitator agent. As expected, when agents are paired with themselves, they achieve scores much higher than when paired with each other—this corresponds to the high scores on the diagonals and the

Table 3.2: Pairwise play scores are produced by playing each MLP imitator agent with every other for 500 games.

First player	Simplebot	Valuebot	Holmesbot	Outer	Iggi	Piers	Rainbow	Van-Den-Bergh
Simplebot	16.8	15.7	12.8	0.0	4.1	1.1	1.7	0.2
Valuebot	15.2	18.0	17.6	0.0	3.8	1.3	2.0	0.0
Holmesbot	11.2	18.3	14.7	0.0	1.4	0.6	0.5	0.0
Outer	0.0	0.0	0.0	14.1	1.0	4.1	6.2	9.0
Iggi	3.9	3.8	1.8	1.8	16.2	11.8	2.7	6.0
Piers	1.8	0.5	0.2	7.2	10.3	15.9	5.5	9.4
Rainbow	0.3	2.0	0.3	6.6	4.0	5.6	18.1	2.6
Van-Den-Bergh	0.0	0.2	0.0	10.6	4.7	8.3	2.7	10.5

relatively low scores on the off-diagonals. This confirms the intuition that agents must use a similar strategy when paired with one another, else risk miscommunicating, losing all three lives, and scoring zero points.

An interesting feature of Table 3.2 warranting discussion is the high scores achieved by certain combinations of agents. Two groups of agents that play relatively successfully with each other are O'Dwyer's Simplebot – Valuebot – Holmesbot, and Walton-Rivers' Iggi – Piers – Outer – Van-der-bergh. All the agents in each of these two groups belong to the same codebase, so the high scores are likely due to the authors of the two codebases reusing logic between their agents. The exception to this is the Rainbow agent, which was produced using reinforcement learning, and so it is surprising that it performs so well with Walton-Rivers' agents. Our best explanation is that both agents reportedly prefer value hints and can presumably also respond well to game states where value hints have been given. Even more interestingly, some agent combinations (e.g. Holmesbot – Valuebot) result in scores greater than either agent can achieve in self-play. We surmise this is a result of synergy between the two agents' strategies. Although such synergies could cast doubt on the ability of pairwise scores to tell whether agents' strategies are different, we argue that they are rare, and thus have an overall small effect on the trustworthiness of using the pairwise scores as a validation

metric.

Finally, we consider the phenomenon that for some agent combinations, it matters significantly who goes first. Specifically, for the combinations Simplebot–Holmesbot, Outer–Piers, and Outer–Van-den-bergh, if the first of the listed agents goes first, the combination gets around 2 points more on average than if the second listed agent goes first. This effect too is likely due to synergies between the agents, and additionally serves to show that in general, synergies have a fairly small effect on the final score.

3.6 Ad-Hoc Agent Modeling

In this work, we propose to apply MAML to the cooperative decision making problem of the ad-hoc challenge. The goal of this algorithm is to train a set of model parameters that are a single gradient update away from good generalization performance on a new learning task (imitating the strategy of a previously unseen agent), using only a small number of training examples—just 10 games in this case. Intuitively, the training process finds a meta-optimum that is close in parameter space to many other task-specific optima. Since MAML is model agnostic—meaning that the algorithm can be applied to any model that trains via backpropagation—we had to make a design decision on what model to choose. We decided not to use a recurrent neural network since none of the original agents consider a game's action history when making moves; rather they take moves considering only the information available in the current game state, as represented by the HLE. Preferring simplicity, we decided to make the MAML model a fully connected network.

We also train a set of "naive" MLPs, so called because they only use the test agent's 10 games to learn its strategy. This isn't meant to be a fair comparison to MAML, but simply a lower bound to demonstrate that it is necessary to use data from other agents to build up an inductive bias before tackling the Ad-hoc Challenge. As a point of clarity, the naive MLPs and the imitator MLPs have different architectures; the naive MLPs have much fewer parameters, because they are trained on so little data. Please refer to the end of section

3.6.2 for architecture details.

3.6.1 Summary of Experiments

First, the agent imitations from section 2.3 were used to generate 500,000 games per agent. Then, for each held-out agent, the MAML algorithm was run for 5 epochs on all the HOAD agents' games except for those of the held-out one, resulting in eight different meta-models. A checkpoint was taken after just 2 epochs for each meta-model.

Evaluation occurred in two scenarios: low-shot and zero-shot. The only difference between the two is that in the low-shot scenario each of the above models was updated with 10 games of their respective held-out agents, whereas this was not done in the zero-shot scenario. The now agent-specific models (or just the meta-models in the zero-shot scenario) were then played with their respective held-out agent imitations for 50 games each. We then repeated the above evaluation process 10 times, averaging scores across both games and trials.

Each naive MLP, on the other hand, was trained on only 10 games from each agent. The resulting model was then played directly with the corresponding agent imitator for 50 games each. This evaluation process too was repeated 10 times (including retraining each MLP on different games each time), and the scores averaged across both games and trials, for 500 total evaluation games per agent. The results for both of the above experiments may be found in table 3.3.

3.6.2 Implementation Details

The model used with MAML consists of 12 fully connected layers with hidden sizes 2048, 2048, 1024, 1024, 512, 512, 256, 256, 128, 128, 64, and 64. The model input is the binary vector representation of the game state for 2 players (length 658), as defined by the HLE, and the output is the argmax of the logits layer, which is of length 20 since there are 20 possible actions in a 2 player game. We use leaky ReLu with a negative slope of -0.3, batch normalization at each layer, and skip connections every other layer for increased performance.

Algorithm 1 FOMAML for HOAD

Require: M self-play games from each HOAD agent
Require: a differentiable model to use with FOMAML

Definitions:
k: number of agents to sample
d: number of ad-hoc games
α, β: task and meta learning rates
$\mathcal{L}$: cross entropy loss
θ: parameters of meta-model
f_θ: output of forward pass of meta-model
end Definitions:

Randomly initialize θ
while not done **do**
  sample k agents
  for each agent i **do**
    support $\leftarrow$ sample d games from agent i
    Compute $\nabla_\theta \mathcal{L}(f_\theta)$ using support
    Compute adapted model with gradient descent:
    $\theta'_i \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(f_\theta)$
    query $\leftarrow$ sample d more games from agent i
    Compute $\nabla_{\theta'_i} \mathcal{L}(f_{\theta'_i})$ using query
    Update meta-model using first order approximation:
    $\theta \leftarrow \theta - \beta \nabla_{\theta'_i} \mathcal{L}(f_{\theta'_i})$
  end for
end while

Table 3.3: Ad-hoc play scores for randomly selected partners, the naive MLP, and MAML methods. The naive MLP is trained on the 10 ad-hoc games of each held-out agent. The MAML models are trained on 500,000 games per each of the other seven HOAD agents (3.5M games total), and in the low-shot scenario are also updated with 10 ad-hoc games from the held-out agent. Evaluation is done by playing 500 games with the held-out agent.

			MAML 2 epochs		MAML 5 epochs	
Held-out Agent	Random Partner	Naive MLP	Zero Shot	Low Shot	Zero Shot	Low Shot
Simplebot	5.1	1.1	4.2	5.3	4.6	3.8
Valuebot	5.7	1.5	5.3	7.6	8.4	8.9
Holmesbot	4.6	0.5	6.2	6.8	9.4	7.9
Outer	2.9	2.5	3.0	3.6	3.4	3.9
Iggi	4.5	2.5	5.9	11.1	6.2	8.1
Piers	5.0	2.0	9.3	8.3	11.0	10.2
Rainbow	3.1	1.8	2.9	1.8	3.1	2.6
Van-Den-Bergh	3.8	2.2	4.5	4.7	6.7	5.4
Averages	4.3	1.8	5.2	6.2	**6.6**	6.3

In order to model previously unseen agents, it is necessary to modify the experimental setup typically used with MAML [35]. Typically, an N-way K-shot setup is used, where N unseen classes are selected and K examples of each class are provided, and then the model is expected to successfully classify among the N classes. In our implementation, we instead select one unseen agent, update the model with 10 of that agent's games (which is anywhere from 40 – 800 observation-action pairs depending on the length of the game), and then give the model an observation, and ask it to predict the action that the unseen agent would have taken. The technique and intuition behind MAML (i.e. training to find the meta-optimum) still hold; we have simply changed the objective.

The most significant difference between our implementation and the original MAML is that we're using the first-order approximation (FOMAML) described in [2] and [11]. In practice, the only difference between it and the original MAML is that the meta-model is updated with $\nabla_{\theta_i'}\mathcal{L}(f_{\theta_i'})$ instead of $\nabla_{\theta}\mathcal{L}(f_{\theta_i'})$ (please refer to our reproduction of FOMAML in Algorithm 1 for clarification on this notation). During each meta training step, 4 training agents are sampled. From each agent, we sample 10 games—this is the support data. Each agent's support data is used to make a gradient update with the SGD optimizer and a learning rate of 0.0001 (the objective is the imitation task); thus, we get an adapted model for each agent. Next, 10 more games are sampled from each agent—this is the query data.

We now find the loss using the adapted parameters and the query data, and then we compute the gradient with respect to this loss and the adapted parameters. This is the point where FOMAML diverges from MAML; in MAML, we would instead have computed the gradient with respect to this loss and the original parameters. Once we have computed this gradient, we use it to update the meta-model. As in the original MAML, this procedure optimizes the meta-model parameters such that a single gradient update on a new agent's games will produce a maximally accurate imitation of that agent. Finally, every 10,000 episodes the meta-learning rate is halved, until convergence.

The naive MLPs are a fully connected network with layer sizes 512, 512, 512, and 256, batch normalization, dropout = 0.5, and Adam optimizer with learning rate 1.5e-4. These parameters gave the best results after a thorough hyperparameter search.

3.6.3 Discussion

The first thing we notice in Table 3.3 is that the naive MLP gets fairly low scores across the board; this backs up our intuition that it is very difficult to learn the mechanics of the game (not to mention the specific strategy of the held out agent) from just 10 games of data. Comparing these scores to the case where we play the held-out agent with a random other HOAD agent, we can see that at least being familiar with the game mechanics helps to greatly boost the score, by more than a factor of 2. Playing the held-out agent with random HOAD agents also provides a good baseline for how similar the strategies of the various agents are. Whereas in self-play the average HOAD agent achieves around 15.5 points, paired with a randomly selected agent, a held-out agent can only expect to earn about 4.3 points. This baseline is also useful when measuring the performance of an ad-hoc learner—if the learner cannot outperform a randomly selected partner, then it is not a very competitive learner. In this case, we can see that the MAML approach outperformed the randomly selected partner in both the zero and low shot scenarios, by around 2 points.

An important caveat that bears mentioning is that the MAML results are affected to an unknown extent by the fact that some of the agents are drawn from the same codebase and

share some conventions. It is possible that some of the improvement seen in the MAML trials is due to agents copying conventions from other agents in the same family. Because diversity is an important aspect of the HOAD dataset, an essential piece of future work will be to develop a more principled approach to measuring similarity between bots–one that accounts for in-family similarities as well.

The most interesting result among the MAML runs is how the effect of adapting the parameters with the ad-hoc games changes as the model converges. Namely, in the early stages of training, the adaptation step seems to help, gaining about 1 point on average. However, once the model is converged, adapting to the held-out agent has no significant effect on performance (and indeed, seems to have a slightly deleterious one). This goes against the results of Raghu, who found that inner loop updates have negligible effect on learned representation at both early and late stages of training [26]. There are, however, some important differences between our work and theirs that may explain this discrepancy—namely, they performed this test using second-order MAML on Omniglot and MiniImageNet, whereas we use first-order MAML on Hanabi. Raghu does test first-order MAML as well, but only to compare the computational speed of FOMAML to their own algorithm ANIL (Almost No Inner Loop). They show that nearly eliminating the inner loop of MAML can speed up computation by about as much as using first-order MAML. They do not, however, show whether FOMAML benefits from an inner loop in the early stages of training, which in our case, it does.

3.7 Conclusion

In this work we demonstrated that the FOMAML algorithm can be applied to the Hanabi Ad-Hoc Challenge and the cooperative decision making problem it entails. We also argued that HOAD is a good benchmark for meta-learning based approaches, because of the quality and diversity of its agents. When FOMAML is trained on HOAD, the resulting meta-model was able to outperform both a naive MLP baseline, and a randomly selected partner.

Furthermore, we showed that after a sufficient number of training steps, the meta-model does not need to be updated on the 10 games of the held-out agent, and will actually perform better without it. This backs up recent evidence that MAML-type algorithms don't benefit from the task update step during test-time evaluation.

REFERENCES

[1] Stefano V. Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, may 2018.

[2] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. In *International Conference on Learning Representations*, 2019.

[3] Alexandre Attia and Sharone Dayan. Global overview of Imitation Learning. Technical report, jan 2018. unpublished.

[4] 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. Technical report, feb 2019.

[5] Antoine Bauza. Hanabi, 2010.

[6] Rodrigo Canaan, Julian Togelius, Andy Nealen, and Stefan Menzel. Diverse Agents for Ad-Hoc Cooperation in Hanabi. pages 1–8. Institute of Electrical and Electronics Engineers (IEEE), sep 2019.

[7] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, Marc G Bellemare, and Google Brain. Dopamine: A Research Framework For Deep Reinforcement Learning. In *unpublished*, 2018.

[8] Christopher Cox, Jessica De Silva, Philip Deorsey, Franklin H.J. Kenter, Troy Retter, and Josh Tobin. How to make the perfect fireworks display: Two strategies for Hanabi. *Mathematics Magazine*, 88(5):323–336, 2015.

[9] Yan Duan, Marcin Andrychowicz, Bradly Stadie, Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-Shot Imitation Learning. In *NIPS*, 2017.

[10] Markus Eger and Daniel Gruss. Wait a Second: Playing Hanabi without Giving Hints. 2019.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, 2017.

[12] Chelsea Finn, Tianhe Yu, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-Shot Visual Imitation Learning via Meta-Learning. In *CoRL*, 2017.

[13] Jakob N Foerster, H Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew M Botvinick, and Michael Bowling. Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning. In *Conference on Machine Learning*, 2019.

[14] Victor Garcia and Joan Bruna. Few-Shot Learning With Graph Neural Net-Works. In *ICLR*, 2018.

[15] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *ICLR*, 2018.

[16] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé. Opponent Modeling in Deep Reinforcement Learning. In *ICML*, sep 2016.

[17] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *NIPS*, jun 2016.

[18] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[19] Hoang M. Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated Multi-Agent Imitation Learning. In *ICML*, mar 2017.

[20] Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. Meta-SGD: Learning to Learn Quickly for Few-Shot Learning. Technical report, 2017. unpublished.

[21] Chris Metzen and James Phinney. StarCraft: Remastered, 1998.

[22] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A Simple Neural Attentive Meta-Learner. In *ICLR*, jul 2018.

[23] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P Lillicrap, David Silver, and Koray Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *JMLR*, 48, 2016.

[24] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[25] Arthur O'Dwyer. Github - quuxplusone/hanabi: Framework for writing bots that play Hanabi., 2018.

[26] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2020.

[27] Nathan D Ratliff, David Silver, and J Andrew Bagnell. Learning to search: Functional gradient techniques for imitation learning. *Auton. Robots*, 27:25–53, 2009.

[28] Stéphane Ross and J Andrew Bagnell. Efficient Reductions for Imitation Learning. *JMLR*, 9:661–668, 2010.

[29] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-Learning With Latent Embedding Optimization. In *ICLR*, 2019.

[30] Aron Sarmasi, Timothy Zhang, Chu-Hung Cheng, Huyen Pham, Xuanchen Zhou, Duong Nguyen, and Soumil Shekdar. Hanabi open agent dataset, 2020.

[31] Aron Sarmasi, Timothy Zhang, Chu-Hung Cheng, Huyen Pham, Xuanchen Zhou, Duong Nguyen, Soumil Shekdar, and Joshua McCoy. HOAD: The Hanabi Open Agent Dataset. In *AAMAS*, Montreal, Canada, 2021.

[32] 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, 2016.

[33] Jake Snell, Kevin Swersky, and Twitter Richard Zemel. Prototypical Networks for Few-shot Learning. In *NIPS*, 2017.

[34] Flood Sung, Yang Yongxin, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to Compare: Relation Network for Few-Shot Learning. In *CVPR*, pages 1199–1208, 2018.

[35] Oriol Vinyals, Google Deepmind, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching Networks for One Shot Learning. In *NIPS*, 2016.

[36] Joseph Walton-Rivers, Piers R. Williams, Richard Bartle, Diego Perez-Liebana, and Simon M. Lucas. Evaluating and Modelling Hanabi-Playing Agents. In *CEC*, pages 1382 – 1389, 2017.

[37] Ziyu Wang, Josh Merel, Scott Reed, Greg Wayne, Nando de Freitas, and Nicolas Heess. Robust Imitation of Diverse Behaviors. In *Conference on Neural Information Processing Systems*, jul 2017.

[38] D Wu. GitHub - lightvector/fireflower: A rewrite of hanabi-bot in Scala.

[39] Tianhe Yu, Chelsea Finn, Annie Xie, Sudeep Dasari, Tianhao Zhang, Pieter Abbeel, and Sergey Levine. One-Shot Imitation from Observing Humans via Domain-Adaptive Meta-Learning. Technical report, 2018. unpublished.

[40] James Zamiell. GitHub - Zamiell/hanabi-conventions: A list of Hanabi strategies.