university of groningen / faculty of mathematics and natural sciences / artificial intelligence

Theory of Mind for Multi-agent Coordination in Hanabi

Masters Thesis
(Artificial Intelligence)

Nicholas Kees Dupuis (s2843013)

August 9, 2022

Internal Supervisor(s): Prof. Rineke Verbrugge, Dr. Harmen de Weerd

Artificial Intelligence / Human-Machine Communication
University of Groningen, The Netherlands

Acknowledgements

First, I'd like to thank Rineke and Harmen for their patience with my project's many hiccups, and their advice on overcoming them. I'd also like to thank Nicole Nohemi and Akash for the many hours we played Hanabi together, and all they taught me about the game. Nicole also helped me come up with the original research direction, for which I'm really grateful. Thanks as well to Milan for talking through my RL troubleshooting with me and to Akash again for staying up all night helping me debug the MCTS algorithm. I'm also extremely thankful to Varun for taking the time to help me understand Tensorflow, and always answering any and all of my random programming questions. Lastly, I would like to thank Hadasa for her amazing photography skills and Dave for his beautiful hands.

THE UNIVERSITY OF GRONINGEN

Abstract

Master of Science

Theory of Mind for Multi-agent Coordination in Hanabi

by Nicholas Kees Dupuis

In order to successfully coordinate in complex multi-agent environments, AI systems need the ability to build useful models of others. Building such models often benefits from the use of theory of mind, by representing unobservable mental states of another agent, including their desires, beliefs, and intentions. In this paper I will show how theory of mind affects the ability of agents to coordinate in the cooperative card game Hanabi. The ability to play Hanabi well with a wide range of partners requires reasoning about the beliefs and intentions of other players, which makes Hanabi a perfect testbed for studying theory of mind. I will use both symbolic agent-based models designed to play a simplified version of the game which explicitly engage in theory of mind as well as reinforcement learning agents which use meta-learning to play the full version of the game. Both methods were used to build models of other agents and thereby test how theory of mind can both promote coordination as well as lead to coordination failure. My research demonstrates that the effect of theory of mind is highly variable, and depends heavily on the type of theory of mind reasoning being done by the partner. The empirical results of the agent-based models suggest that theory of mind is best applied when the joint policy produced without theory of mind is far from optimal, in which case second-order theory of mind appears to offer the most significant advantage. Zeroth-order agents are able to play quite well partnered with other zeroth-order agents, but when the environment contains agents using theory of mind, it pays to use theory of mind as well, particularly second-order theory of mind or a mixture of second-order and first-order. The results of the reinforcement learning agents show no advantage of theory of mind reasoning in either self-play or when paired with rule-based agents.

Contents

Abstract **v**

1 Introduction **1**
1.1 Cooperative AI . . . 1
1.2 Theory of Mind . . . 2
1.2.1 Intentional Systems . . . 2
1.2.2 Optimization and Intention . . . 4
1.2.3 Intentional Theory of Mind . . . 6
1.3 Hanabi . . . 8

2 Agent-based Theory of Mind **11**
2.1 Introduction . . . 11
2.1.1 Related Work . . . 12
2.1.2 Monte Carlo Tree Search . . . 12
2.2 Methods . . . 13
2.2.1 Simplified Hanabi . . . 13
2.2.2 Zeroth-order Agent (ToM0) . . . 13
Imperfect Information . . . 14
Value and Policy . . . 15
Training with MCTS . . . 16
2.2.3 Theory of Mind Agents . . . 16
First-order agent (ToM1) . . . 16
Second-order agent (ToM2) . . . 17
Mixed-Order Agents . . . 17
ToM0+ . . . 18
2.3 Results . . . 19
2.3.1 Training the Zeroth-order Agent . . . 19
2.3.2 Testing Theory of Mind . . . 20

3 Cooperative Reinforcement Learning **27**
3.1 Introduction . . . 27
3.1.1 Related Work . . . 28
3.1.2 Machine Theory of Mind . . . 28
3.1.3 Reinforcement Learning with RainbowDQN . . . 29
Q-learning . . . 29
Deep Q-learning . . . 30
Rainbow . . . 31
3.2 Methods . . . 33
3.2.1 Hanabi Learning Environment . . . 33
Reinforcement Learning Environment . . . 34
3.2.2 RainbowToM . . . 34
RainbowZero . . . 36
3.2.3 Troubleshooting . . . 37

3.2.4 Rule-based Agents . . . 40
3.2.5 Ad-hoc Environment . . . 42
3.3 Results . . . 42
3.3.1 Rule-based Agents . . . 42
3.3.2 Self-play . . . 42
3.3.3 Ad-hoc Experiments . . . 43

4 Discussion and Conclusion **49**
4.1 General Findings and Future Work . . . 49
4.1.1 Interpretable AI . . . 49
4.1.2 Limitations . . . 50
4.1.3 Concluding Remarks . . . 50

Bibliography **51**

List of Abbreviations

DQN	**D**eep **Q**-**N**etwork
LDA	**L**inear **D**iscriminant **A**nalysis
MCTS	**M**onte **C**arlo **T**ree **S**earch
RL	**R**einforcement **L**earning
TD	**T**emporal **D**ifference
ToM	**T**heory **o**f **M**ind

Chapter 1

Introduction

1.1 Cooperative AI

Our civilization is characterized by groups of individuals interacting in a wide variety of multi-agent settings. The ability to cooperate and coordinate is central to the ability of an individual to navigate those settings, and it is thus critical for artificially intelligent agents to also share in that ability.

Taking effective action in an environment often requires having a useful model of that environment, and so in the case of multi-agent environments, agents must be able to build useful models of other agents. The term "Theory of Mind" (ToM) refers to the particular ability of modeling the unobserved inner state of other agents (or agent-like systems), such that those models allow for better predictions of that agent's behavior. A thorough explanation and definition of ToM will be given in Section 1.2.

This ability can be essential to the ability of individuals to coordinate. Even in purely cooperative settings, the ability of a group of individuals to reach their common goals can hinge on their ability to correctly interpret each other's actions, and to correctly predict how actions will be interpreted (Blokpoel et al., 2012).

To study the effect that using different kinds of theory of mind has on the ability of agents to cooperate and coordinate toward a common purpose, I will be testing the application of ToM in the context of the card game Hanabi. Hanabi is a fully cooperative multi-player game with imperfect information, where players must manage to coordinate their actions to score points. Players are allowed only very limited explicit communication, and so must rely on implicit communication to share knowledge. The game rules will be fully explained in Section 1.3.

Because it is deliberately designed as a challenging environment for players to coordinate in, Hanabi makes a good test bed for the study of coordination, and in particular, the role that ToM has in facilitating that coordination. I will be studying the application of ToM from two angles:

Agent-based Theory of Mind First, I will be studying theory of mind and coordination in the context of a simplified version of Hanabi (see Chapter 2). This will allow me to study highly idealized types of symbolic reasoning, and to perform experiments which would not be computationally tractable in the full version of the game. The advantage of such as simplified context is that experiments can be set up to limit confounding variables, and I can exactly specify how an agent will apply ToM. The disadvantage of a simplified context is that it might not generalize as well to other more complex cooperative tasks.

Cooperative Reinforcement Learning Second, in Chapter 3 I will also be performing experiments with reinforcement learning agents on the full version of the game. I will be using state of the art machine learning methods, in particular neural network architectures, which will allow me to study coordination in a more complex and practical setting. The disadvantage of using neural networks is that the internal representations and decisions of the agents are latent and inaccessible, though I will do some work to extract features from this latent space in Section 3.3.3.

1.2 Theory of Mind

Theory of Mind (ToM) is the ability of one mind to model the unobservable inner state of another mind, including their beliefs, intentions, and values. As a mind goes about modeling their environment, ToM is critically about modeling those parts of the environment which are also minds, and might also be occupied with modeling and reasoning about their environment.

This partitioning of the environment, however, into the mental and the non-mental, deserves significant scrutiny. Unless we take a dualist view, minds are a product of the same physical laws as everything else in the universe, and are not fundamentally distinct from other matter. And yet, as pointed out by Schrödinger (2012), life, and in particular mental life, behaves as if bound by a different set of laws, and its behavior cannot be well understood from a physical understanding alone.

Theory of Mind, which is the focus of this report, is exclusively related to the study of what are called "intentional systems", and how those intentional systems attempt to model each other.

1.2.1 Intentional Systems

We live in a world containing minds, each with their own beliefs, desires, and limited rationality. While it's clearly very important to develop a theory of how these minds interact with each other and the rest of the world, terms like "mind", "intention", or "belief" can often seem vacuous and not concrete enough for a scientific discipline. Behaviorists in psychology deliberately avoided these concepts at any cost in order to maintain scientific rigor, and while the need for concreteness is very real, many phenomena simply cannot be explained without engaging constructively with these terms. [1] For example, if a person spends time in a room without being told why, and is then later asked to describe contents of the room, their ability to do so cannot be well explained without supposing that during that time they formed a mental model of the room.

Instead of getting hung up on defining the true boundaries between the mental world and the non-mental world, Dennett (1981) proposes to consider how a system can be usefully modeled, or what kinds of models are capable of explaining and predicting a system's behavior. He offers three different approaches or stances to building such models:

[1] Behaviorism's refusal to examine mental processes lead to the *cognitive revolution*, which took place in the 1950s. Important works like Chomsky (1959) lead to the establishment of *cognitive science* which has since largely eclipsed behaviorism. Cognitive science tries to understand behavior by trying to model unobservable mental processes and developing theories of cognition.

Physical Stance A physical model is one which takes the full physical state of a system and the rules of nature to predict the behavior of that system. This model is fully reductionist, and explains a system by the interaction of its physical parts. For example, to predict how a set of billiard balls will react to being struck, it might be useful to consider the exact positions of each billiard ball, the friction of the table, the laws of motion, etc.

Functional Stance A functional model abstracts away the exact details of how a system's physical components interact, and focuses instead on the purpose of the system, or what function it implements. Such a model takes the input given to the system and a functional relationship between input and behavior to predict how the system will react. For example, to predict how a car will react when the gas pedal is pressed, a physical model would consider how each mechanical part of the car interacts and distributes the forces applied, while a functional model would consider the design of the car, and the expected relationship between the position of the pedal and the car's acceleration.

Both the physical and the functional stances are fundamentally non-mental, and attempt to understand a system without assuming any beliefs, goals, or rationality.

Intentional Stance An intentional model directly uses beliefs, desires, and an assumption of rationality to predict behavior. This is the default stance we take when discussing complex decision-making systems like animals and AI. For example, to predict the behavior of a chess-playing computer, a physical model might consider the exact voltage of all of its circuits, and how those circuits allow current to move through the machine. A functional model might represent the complete function directly mapping board states to actions. An intentional model, however, considers the computer as taking rational action in order to move towards winning the game. An intentional model here might be more useful, because it abstracts away a lot of very complex detail that might be hard to uncover or make one's model unmanageable large, but it is still able to assist in predicting the actions such a chess computer will take.

It might feel quite suspect to talk about such a simplistic system as having "beliefs" or "desires", but the point Dennett makes is that it's not about whether the system truly "thinks" or "wants" anything, but whether the intentional stance is useful in predicting its behavior.

Clearly systems can be modeled in many different ways. I could explain my thermostat as a rational agent trying with its limited knowledge to maintain a desired temperature, or I could, like the behaviorists, insist on modeling animal behavior as entirely a function of stimulus and reward.

Dennett defines a class of "intentional systems" as systems which can sometimes most usefully be modeled with the intentional stance. This quality of "intentionality" is also sometimes called "agency", referring to properties associated with the mathematical object of an "agent" as used in economics. This definition crucially does not assume that other stances cannot be useful in understanding such systems, nor that they are idealized agents with full rationality, only that the intentional stance has epistemic value in understanding such systems.

I will be taking a strict interpretation of the term "intentional stance" to refer to modeling systems as if they have three main properties:

1. **Beliefs**: An internal model/representation of the world.

2. **Goals**: Some state or set of states the system is working to bring about.

3. **Rationality**: The system is taking actions and updating its beliefs in order to accomplish its goals.

This will help to distinguish the intentional stance from stances which might consider some but not all of those three properties.

1.2.2 Optimization and Intention

In Dennett's framing as described above, a system is considered intentional if it is more usefully modeled with the intentional stance than the physical or the functional stance. There are many systems, however, which do seemingly intelligent things in the world and have powerful effects not explainable with the physical or functional stance, and yet are sometimes but not always usefully modeled as having internal mental states. These systems are called "optimization processes" and goal-directed agent-systems are but a subset of this class of systems. I propose adding an "optimization stance" distinct from the "intentional stance", and to clarify the definition of intentional systems to describe systems which are more useful to model with the intentional stance than either the physical, functional, or optimization stances.

Optimizing Systems Yudkowsky (2008) observes that there is a class of systems which can be understood by considering the target they are optimizing for, without necessarily considering any details about their internal state. They powerfully shape the world by aiming for configurations of atoms which, a-priori, are vanishingly unlikely to appear on their own. Flint (2020) provides a more concrete definition of optimizing systems as systems which robustly tend toward a target configuration from a basin of attraction. This could range from an algorithm estimating an irrational number, to a set of agents accomplishing a goal, to natural selection crafting a species to survive its environment (see Figure 1.1).

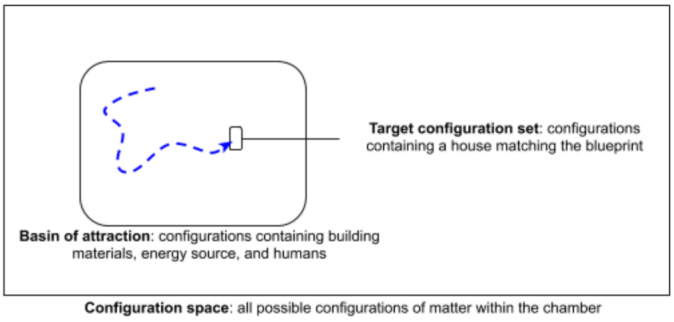
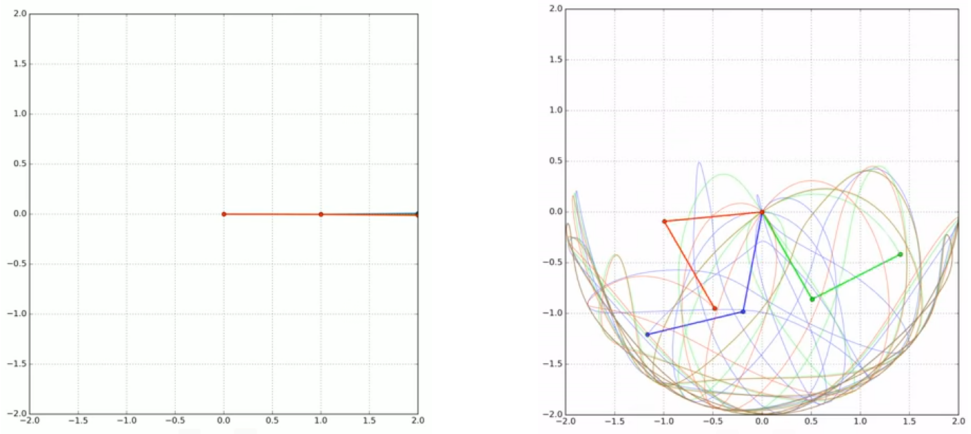


FIGURE 1.1: From Flint (2020) with permission. The diagram models a group of humans trying to build a house as an optimization process. Despite not considering the details of how the system will evolve, we can predict the system will eventually end up in one of the target configurations.

What makes these processes distinct from other dynamical processes is that they are robust to perturbations. Many complex systems are inherently chaotic, and small perturbations result in huge changes in their final states. A common example of this is the double pendulum (see Figure 1.2).



(A) Initial state of three double pendulums, one placed horizontally to the right with the other pendulums set at 0.5 degree deviations.

(B) Small initial differences lead to totally different trajectories. Predicting the final state requires knowing the exact initial state.

FIGURE 1.2: Chaotic motion of a double pendulum. Screenshots are taken from a video demonstration by Weiss (2016) which can be found on YouTube.

Systems undergoing optimization tend toward a set of target configurations from anywhere within a broad basin of attraction, and that process is robust to certain changes in the state of the system (see Figure 1.3).

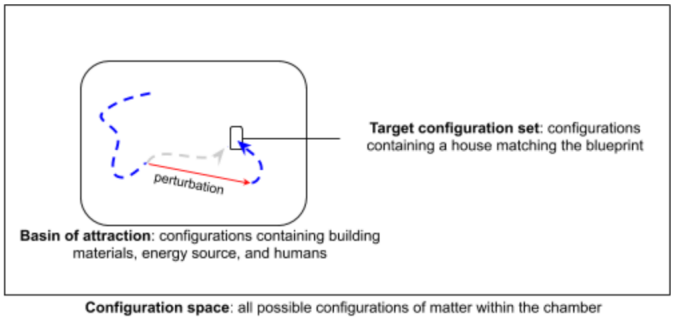


FIGURE 1.3: From Flint (2020) with permission. Here a perturbation is made to a system undergoing optimization, and while it changes the trajectory of that system's evolution, it still predictably ends up moving to the target configuration.

This does not imply that such a system needs to be infinitely robust to all changes to its state. If the configuration is sufficiently changed, the optimization process can be destroyed, and the target configuration will never be reached (see Figure 1.4). For example, natural selection produces species which are highly adapted to their environments, and the process is robust to the death of individual organisms or even species extinction. If the atoms of the world were rearranged sufficiently, however, all life would be destroyed, and with it natural selection.

These types of systems, just like intentional systems as defined by Dennett (1981), are not usefully modeled with the physical stance or the functional stance. They are not, however, always usefully modeled in terms of beliefs, goals, and rationality. While natural selection might very abstractly be thought of as having a goal, in terms of the states of the world it robustly brings about, it is nonsensical to try to predict how evolution will bring about those states in terms of beliefs and rational choice.

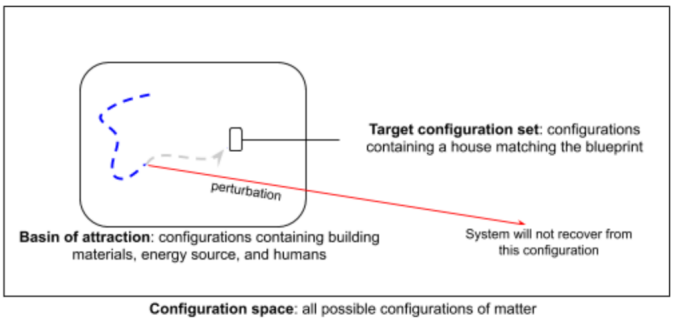


FIGURE 1.4: From Flint (2020) with permission. Here a perturbation is made to a system undergoing optimization which puts it outside the basin of attraction, thereby ending the process.

From this perspective, intentional systems are better thought of as being a subclass of the broader set of optimizing systems (see Figure 1.5).

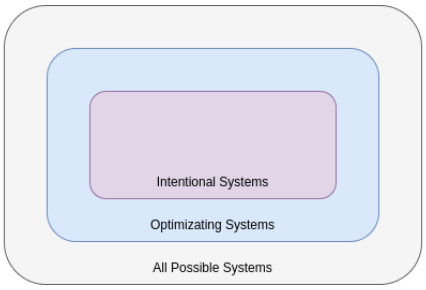


FIGURE 1.5: Intentional systems as a subclass of optimizing systems.

Intentional systems can be modeled merely as optimization processes, but they can also be explained and predicted with much more detail by attributing beliefs and assuming limited rationality (see Figure 1.6).

Optimization Stance The intentional stance considers three main things: belief, goals, and rationality. Combined, they can be used to predict the behavior of an intentional system. I propose, for the purposes of disambiguation, to also consider a more abstract optimization stance: an approach which considers only a target configuration, or in more agentic language, a goal.

It can be tempting to assign intention to optimizers because they share qualities with intentional systems, particularly because we spend so much time interacting with minds that behave according to beliefs and intention, but doing this may not always lead to better predictions.

An intentional system is one which is more usefully modeled with the intentional stance than just the optimization stance.

1.2.3 Intentional Theory of Mind

Earlier I defined Theory of Mind as the ability of one mind to model another, but I shall now make this more concrete using definitions I've laid out above:

Theory of Mind is the ability of intentional systems to build intentional models of other intentional systems.

Critical to this definition is that those models are made with the intentional stance as defined by Dennett (1981). If I model other people around me using the

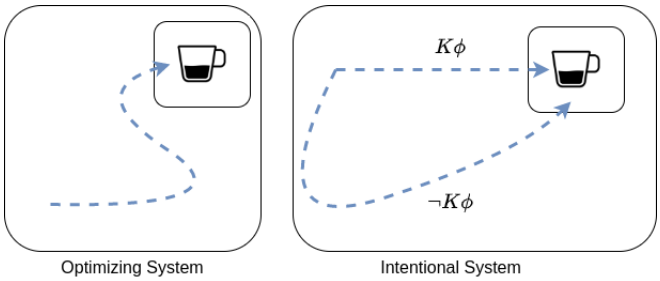


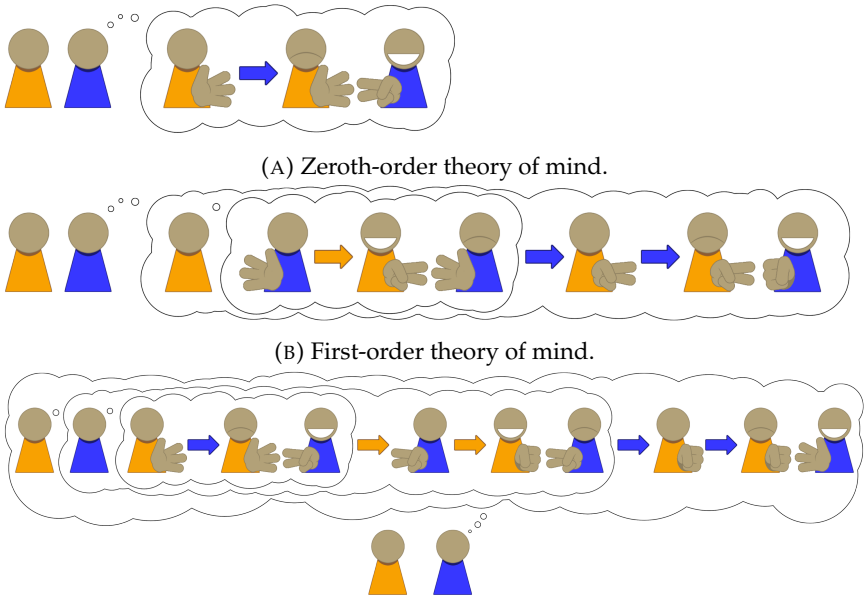
FIGURE 1.6: Here I model my path toward obtaining coffee. If the scenario is modeled as an optimizing system which robustly results in me having coffee, much of the detail of my trajectory is abstracted away. For example suppose there is a cafe close to me where I prefer to go which is either closed ϕ or not closed $\neg\phi$. If I know that the cafe is closed, I will chose a different trajectory. An intentional model is capable of capturing that level of detail, and thereby better predicting how I will behave.

functional stance, as if they were finite state machines like my thermostat, this would not qualify as theory of mind, neither would it it qualify to model them as just optimizers, not considering their beliefs or rationality. Theory of mind is the ability to consider other minds as being rational, intentional beings pursuing goals based on their own models of the world.

This can become recursive, when the system being modeled is also occupied with building intentional models of other systems. The maximum recursive depth used by an agent to understand a situation is referred to the "order", where a mind using Nth order ToM may include in their model minds using up to $(N-1)$th order ToM. For example, Figure 1.7 shows an agent using zeroth, first, and second order theory of mind to select an action in the game rock paper scissors. The zeroth-order agent does not model their opponent using the intentional stance, and rather just tries to find patterns in their behavior. The first-order agent, however, models their opponent as a zeroth-order agent, and likewise the second-order agent models their opponent as a first-order agent.

Studies in humans have been able to assess what order of ToM participants use by constructing puzzles which require a certain order to solve correctly, and are incorrect when attempted with a lower order of ToM. Most humans fully develop the ability to use first order ToM by around the age of 3-4, and continue to develop higher order ToM as they age. (Liddle and Nettle, 2006; Wellman, Cross, and Watson, 2001) Fully grown adults have been tested by researchers with puzzles requiring varying orders of ToM, and their accuracy on these tasks reliably decreases as the order increases. (Kinderman, Dunbar, and Bentall, 1998) Individuals on the autism spectrum often struggle with ToM at a young age, and many must put in more time and effort to develop these abilities (Baron-Cohen, 2001).

While ToM is studied in psychology to better understand how humans go about building these mental models, in the field of artificial intelligence there is also a lot to study. In small models, ToM is studied from the perspective of idealized rational agents. For example De Weerd, Verbrugge, and Verheij (2017) study how different orders of ToM interact with each other in a controlled setting, where artificial agents take actions based on symbolic beliefs. ToM is also studied from the perspective of designing machine learning systems capable of ToM, like Rabinowitz et al. (2018) who demonstrate the use of neural networks to build mental models of simple agents.



(A) Zeroth-order theory of mind.

(B) First-order theory of mind.

(C) Second-order theory of mind.

FIGURE 1.7: An agent using various orders of theory of mind to play rock paper scissors. The zeroth-order (A) agent models their opponent by trying to predict patterns in their behavior, but does not ascribe belief or intention to them. The first-order (B) and second-order (C) agents model their opponent by considering them as taking action based on their own personal model of the game. Figures taken with permission from De Weerd, Verbrugge, and Verheij (2013).

Better understanding ToM both in practice and in theory can help us think about how to build AI systems capable of coordinating and cooperating with other AI systems and the human beings who operate them. It will become significantly more important to understand the phenomenon of ToM as AI systems become more powerful and start to have a larger effect on the trajectory of our civilization.

1.3 Hanabi

Hanabi is a cooperative card game in which 2-5 players must coordinate to play a series of ranked and colored cards in the correct order to earn points. The game contains:

- 50 colored and numbered playing cards
- 8 "hint" tokens
- 3 "disaster" tokens

Each player holds a hand of five cards (or four cards if the game is played with more than three players), which they must hold facing away from themselves such that they can only observe the cards of their partner(s) (see Figure 1.8). Each card has a rank from 1 to 5, and one of five colors. Players work as a group to play cards to form five stacks in increasing order (one stack for each color), and cards may not be played out of order. If a stack is empty, the only playable card of that color is a card of rank 1, otherwise the only playable card is of rank $N + 1$ where N is the rank of the card at the top of the stack.

FIGURE 1.8: Hanabi being played by two players. Each player must hold their cards facing away from themselves, such that only their partner(s) can see them.

When it is a player's turn they have three possible actions, of which they must perform exactly one:

1. Play a card. If the card is playable, it is added to the stack matching its color. If not, the card is removed from the game, and one of three disaster tokens is flipped. If all three disaster tokens are flipped, the game is ended as a loss (zero points).

2. Offer a hint. A player may communicate information to another in the form of a hint. A hint must communicate a single property (either a rank or a color) and must identify each and every card in the receiving player's hand which has that property. A hint may not communicate any other information, and crucially, all other communication is banned from the game. The group starts with exactly eight hint tokens, and offering a hint costs one token.

3. Discard a card. A player may discard a card from their hand in exchange for the group regaining a hint token. This discarded card is removed from the game and is never played. If the group already has the maximum of eight hint tokens, the discard action is illegal, and the player must instead choose to play a card or offer a hint.

If a stack has been completed (one of each rank has been successfully played), upon successfully playing the final card (rank 5), the group regains a hint token. If all eight hints are already available, this bonus is lost.

After playing or discarding a card, players draw a new card to replace it. When the last card is drawn, all players are allowed one more turn. After this, the game is over. If all three disaster tokens have been flipped, the players obtain zero points. Otherwise, they obtain points for the number of successfully played cards, with a maximum score of 25.

There is a limited number of each card type, making playing and discarding cards dangerous. If there are no remaining cards of a required rank and color, then the stack of that color cannot be completed. Players must try to sufficiently inform each other which cards can be safely played or discarded. There are few hint tokens

in the game, and it is highly improbable to achieve a good score while only communicating explicitly. It is necessary for players to communicate implicitly, which is done by relying on the player receiving a hint to reason about the intentions of the sender. This requires players to model their partner(s)' mental states, making Hanabi a game with Theory of Mind baked into its basic strategy.

Chapter 2

Agent-based Theory of Mind

The purpose of this chapter is to explore the effect of theory of mind (ToM) on the ability of agents to coordinate by running experiments with agents which use symbolic ToM reasoning in a simplified version of Hanabi. In Section 2.1 I will define what I mean by "symbolic", describe other research with symbolic ToM agents, and explain Monte Carlo Tree Search which I will use as the backbone of my agents' internal representation of the game. In Section 2.2 I will explain the simplified game of Hanabi, and how the seven different ToM agents that I use are constructed. Finally in Section 2.3 I will describe the results of all the different configurations of agent pairs playing the game.

2.1 Introduction

In this chapter I will be running experiments with symbolic agents using ToM. By "symbolic" I am referring to agents who function on the basis of entirely explicit properties and mechanisms, including:

- **Explicit beliefs**: Beliefs about the world/environment which can be either enumerated or described in a formal logic.
- **Explicit goals**: Either a utility function or a clear goal state (or set of states).
- **Transparent reasoning**: A clear and human repeatable process for updating beliefs or selecting actions based on beliefs. For example, logical deduction, Bayesian updating, or expected utility maximization.

The definition of symbolic agents is set up in contrast to neural network agents produced by gradient descent. Those agents may also have beliefs, goals, and reasoning processes, but they are all *latent* within the neural network and are therefore incredibly difficult to directly observe or control for in experiments. [1]

By studying symbolic agents, I can more easily control for the exact beliefs and reasoning processes that I want to test, and design experiments which are as free from confounding variables as possible. More specifically, I can build agents which explicitly use various types of ToM reasoning, and then empirically test in simulation how those types of ToM affect coordination.

[1] The terms *latent* or *latent space* refer to how each layer of a neural network maps its input into a high dimensional space which is generally not human understandable. Developing better understanding of these latent representations is open area of study (Nguyen, Yosinski, and Clune, 2016; Olah, Mordvintsev, and Schubert, 2017), as is the modification of neural networks and their training loss to make them easier to interpret Zhang, Wu, and Zhu (2018) and Filan et al. (2021). Currently, however, neural networks remain incredibly opaque.

2.1.1 Related Work

It is not obvious that using ToM reasoning would necessarily provide an advantage in all situations. For example, in one-shot scenarios with a single pure-strategy Nash Equilibrium, an agent already playing the equilibrium strategy would gain no advantage from reasoning about the mental states of others.[2] For this reason, some have studied the relative advantage different orders of ToM have in a wide variety of multi-agent settings.

In four different purely competitive games, De Weerd, Verbrugge, and Verheij (2013) show that first-order and second-order ToM are generally quite advantageous, but that using higher-order ToM has diminishing returns beyond second-order. In mixed-motive settings, De Weerd, Verbrugge, and Verheij (2017) find that ToM can also provide a significant advantage even when agents don't have a completely accurate model of their opponent/partner. Furthermore, they show that humans are more likely to engage in ToM when faced with an opponent using higher-order ToM.

In a purely cooperative game called the Tacit Communication Game, where players must coordinate with limited explicit communication, findings by Blokpoel et al. (2012) suggest that human players do use ToM to predict how their partner will interpret their actions (as opposed using heuristics). In an agent-based model of the same game, De Weerd, Verbrugge, and Verheij (2015) show that ToM can be very advantageous, but most interestingly, find that having a lower order of ToM can sometimes also be an advantage depending on the order of ToM of the partner. It would be interesting to see if the same is true in Hanabi.

De Weerd, Verbrugge, and Verheij (2022) also study how the type of environment affects the relative advantage of higher-order ToM, finding that higher-order ToM is particularly useful in highly complex and dynamic environments.

2.1.2 Monte Carlo Tree Search

The models used by the agents I am constructing will be developed using Monte Carlo Tree Search (MCTS), and all my agents will, using different kinds of ToM reasoning, build their beliefs about expected value from those models.

MCTS is a method for searching through a game tree in order to produce a good policy for a player to follow. It balances exploration and exploitation to intelligently search through the tree, and is most commonly used to develop software to play board games (Silver et al., 2017). MCTS explores the tree by iteratively performing the following steps:

1. **Selection**: Starting from the root node, select child nodes until a leaf node (a node with unexplored children) is reached. The selection is done by picking the child with the highest value:

$$S_i = \frac{w_i}{n_i} + c\sqrt{\frac{ln(N_i)}{n_i}} \tag{2.1}$$

   Where for a node i, n_i is the number of times that node has been visited, N_i is the number of times the parent node has been visited, w_i is the number of times the game has been won after visiting the node i, and c is an exploration parameter.

[2] In cooperative or mixed-motive multi-shot scenarios, agents can use ToM to coordinate a joint policy which is better for both parties.

2. **Expansion**: When a leaf node is reached, explore one of the unexplored children.

3. **Rollout**: Using a rollout policy (for example, choosing uniform random moves), select actions until the game is decided.

4. **Backpropagation**: Use the game outcome to update the value of explored nodes visited during selection and expansion.

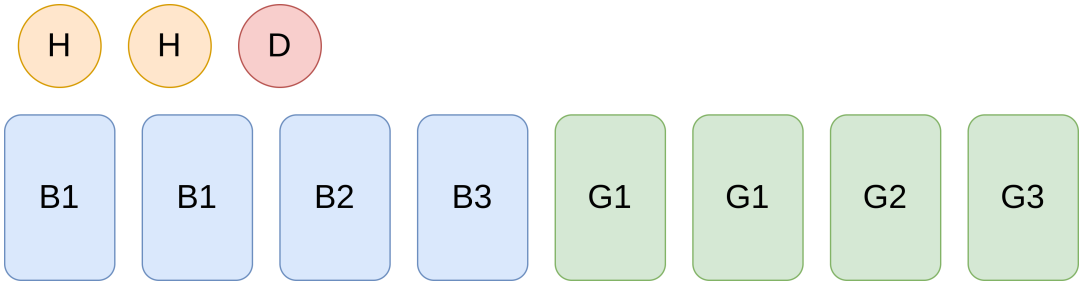
Rounds of search are done for as long as time allows. When MCTS has finished, the final policy is obtained from the empirical values w_i and n_i to maximize the expected outcome.

2.2 Methods

2.2.1 Simplified Hanabi

To make the simulations tractable, I use a simplified version of two player Hanabi.

In this game there are only eight cards, two hint tokens, and a single disaster token:



Each player starts with a random hand of four cards, and no cards are left undrawn. The game continues until a player has no actions they can perform. Instead of eight hint tokens there are only two, and instead of three disaster tokens there is just one. Furthermore, the scoring is simplified such that successfully completing both stacks (which would be worth 6 points) is considered a "win", and failing to do so is considered a "loss".

Each agent maintains a set of possible worlds, each representing a possible configuration of the cards. Upon receiving a hint, or observing a card after playing or discarding it, the agent becomes more certain about their cards, and the size of this set shrinks.[3]

From this set of possible worlds, an agent can obtain a set of possible actions and they must decide which action has the highest expected value. This is done by consulting an internal model the agent has of the game, which differs from agent to agent.

2.2.2 Zeroth-order Agent (ToM0)

The most basic agent I will use is the zeroth-order agent. This agent develops a model of the game in a training phase by searching through the game tree via Monte

[3] I've chosen to represent beliefs as a set rather than with accessibility relations in a Kripke model, as is typically done in epistemic logic. I did this because the simplified game does not involve the drawing of new cards, which means that it lends itself well to a finite set of states, and this made the implementation of my later algorithms much simpler (Meyer and Van Der Hoek, 2004).

Carlo tree search (MCTS). The final tree is consulted during testing both as a value function as well as a policy for selecting actions in each state.

Imperfect Information

Hanabi is not a perfect information game and so the MCTS algorithm must reflect that. Each agent has an observation and a set of actions they can take, but because they do not know the full state of the game (i.e. their own cards), the game tree contains unknown and stochastic transitions. An example of the search tree this causes is given in Figure 2.1.

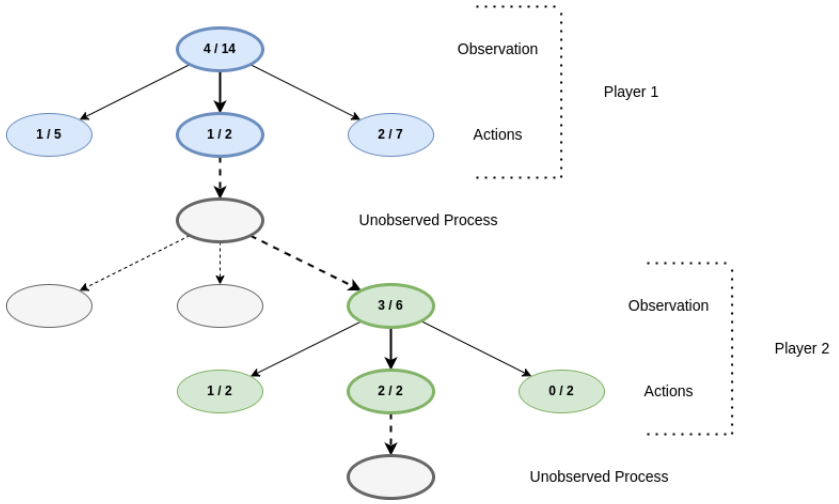


FIGURE 2.1: MCTS with an unobserved process translating actions to partner observations.

Because there is an unknown transition which takes place, states are stored in a lookup table rather than a tree, where each element of the lookup table contains:

- Key: A string uniquely identifying the observation. This includes what the player directly observes and what they know explicitly from previous hints.
- Actions: List of legal actions $\{a_0, a_1, a_2, \dots\}$
- N: Number of times this state has been visited
- $\{n_0, n_1, n_2, \dots\}$: Number of times each action has been visited
- $\{w_0, w_1, w_2, \dots\}$: Number of times the game has been won after taking each action

In order to update the lookup table, a full game is played between two agents. First, during the selection phase, observations are used to generate unique keys which are then used to look up states from the lookup table. Actions are chosen by selecting the action with the highest value[4]: according to Equation 2.1. Once an action is selected, the agent takes that action, and their partner is presented with

[4]The variable c is the exploration parameter. Kocsis and Szepesvári (2006) showed that a value of $c = \sqrt{2}$ ensures asymptotic optimality when the reward is in the range $[0, 1]$, which is true for simplified Hanabi where all outcomes are either 0 (loss) or 1 (win). In practice, c is often set empirically to improve the quality of the search results, but for the purposes of my experiments the quality of search does not matter, and therefore the selection of c is entirely arbitrary. For this reason I've left it at the theoretical value of $c = \sqrt{2}$.

a new observation. That observation is then used to lookup a new state from the lookup table. This repeats until either:

1. A state is reached with an unexplored action. In this case, that unexplored action is taken (expansion), and MCTS enters the rollout phase.

2. A state is not found in the lookup table. In this case, that state is added to the lookup table (expansion), and MCTS enters the rollout phase.

Once the search process is in the rollout phase, uniform random actions are taken until the game ends. Once the game ends, all states visited in the selection and expansion phases are updated with the result of the game. As many games are played as training allows. The final lookup table represents everything that has been learned during the training process.

Value and Policy

The lookup table is used to generate both a value function and a policy. The value of a state is the probability of winning the game from that state, assuming the agent takes the action which maximizes the probability of winning. The policy is the action which maximizes that probability.

During training, the MCTS algorithm balances exploration and exploitation by trading off between an exploration term and an exploitation term in the calculation of node priority (shown in Equation 2.1).

During testing, however, an agent is only interested in exploitation. Naively then, the empirical result $\frac{w_i}{n_i}$ serves as a good estimate for the probability of winning the game with a particular action. This estimate, however, does not distinguish between actions which are well explored and those which are not. To correct for this, I make the simplifying assumptions that each outcome reached after a state-action pair is sampled from a fixed distribution. Assuming a uniform prior over each distribution, I use the rule of succession discovered by De Laplace (1794) to obtain the probability of winning the game after a particular state-action pair to be the following:

$$P(s,i) = \frac{w_i + 1}{n_i + 2}$$

If the value of a win is set to be 1 and the value of a loss is set to be 0, then this directly translates into a Q-value giving the expected value of a state-action pair;

$$Q(s,a_i) = \frac{w_i + 1}{n_i + 2}$$

From this, a zeroth-order agent can derive both a policy as well as a value function:

$$\pi_0(s) = \arg\max_i \left(\frac{w_i + 1}{n_i + 2}\right)$$

$$V_0(s) = \max_i \left(\frac{w_i + 1}{n_i + 2}\right)$$

This also implicitly deals with the problem of comparing unexplored states where $n_i = 0$ to explored states where $n_i > 0$.

Training with MCTS

Monte Carlo Tree Search is a method for more intelligently searching a game tree. Run for long enough (with enough memory), it will explore the entire game tree and is proven to converge to perfect play[5] (Kocsis, Szepesvári, and Willemson, 2006). Instead of building a model capable of perfect play in simplified Hanabi, I will be logging models at different stages in training in order to study more broadly how theory of mind affects the ability of agents to cooperate with each other.

2.2.3 Theory of Mind Agents

I define a range of agents using different levels of theory of mind. Each agent has a value function $V(s)$ which estimates the expected value of the current state and a policy $\pi(s)$ which maximizes expected value. The way that a theory of mind agent selects actions and estimates expected value depends on the assumptions that they make about their partner. Let the assumed value function of the partner be $V_p(s)$.

In Hanabi, each agent is not able to directly view their own cards, and therefore the complete configuration of the cards is not known. Let $W(s) = \{w_0, w_1, \dots\}$ be the set of possible worlds, or configurations of the cards, which are not ruled out by what the agent has observed so far. Furthermore, let $A = \{a_0, a_1, \dots\}$ be the set of possible actions, and $T(w, a)$ be the transition function which takes a world and an action, and generates an state s_p which the partner will observe in the next turn.

I define my theory of mind agents such that they take action in order to maximize the expectation over the expected value estimated by their partner in the next turn:

$$Q(s,a) = \frac{1}{\|W(s)\|} \sum_{w \in W(s)} V_p(T(w,a))$$

$$V(s) = \frac{1}{\|W(s)\|} \max_{a \in A} \sum_{w \in W(s)} V_p(T(w,a))$$

$$\pi(s) = \frac{1}{\|W(s)\|} \arg\max_{a \in A} \sum_{w \in W(s)} V_p(T(w,a))$$

A visual representation of this calculation is given in Figure 2.2.

First-order agent (ToM1)

A first-order agent in my framework is an agent which assumes that $V_p = V_0$, or that their partner is a zeroth-order agent. Let this agent's value function be $V_1(s)$.

[5] Perfect play here does not mean winning all of the time, rather it means pursuing an optimal policy. I'm interested in observing how ToM reasoning affects play both positively and negatively, but if the zeroth-order agents are already pursuing optimal policies I cannot study how ToM might be an improvement. For this reason, I choose to use models from across the spectrum of competency.

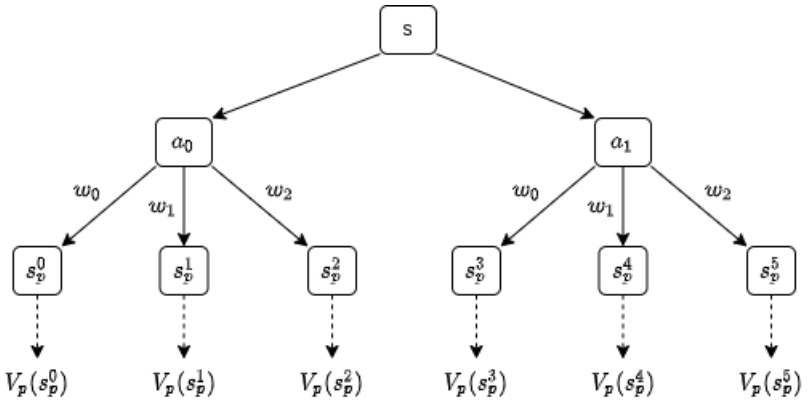


FIGURE 2.2: One level of ToM reasoning. The expected value of the state s is estimated by calculating the value of $V_p(s_p)$ for all possible values of s_p

Second-order agent (ToM2)

Similarly, a second-order agent in my framework is one which assumes that $V_p = V_1$. This level of theory of mind has one level of recursion, and so each leaf in the search tree shown in Figure 2.2 corresponds to a tree of first-order reasoning.

Mixed-Order Agents

In the paper by De Weerd, Verbrugge, and Verheij (2015) the authors implement mixed-order ToM by means of a confidence term representing an agent's belief that their partner is using a particular order of ToM. Many trials of the game are played with each pair of agents, and between trials this confidence shifts as the behavior of the partner either conforms or conflicts with expectations.

I will also study mixed-order ToM, but instead of a shifting weighting between different orders I will be using fixed mixtures. This is in part because I wish to eliminate confounding variables, and just observe how various strategies interact, and also because I am only studying the purely ad-hoc setting of Hanabi where players attempt to coordinate with entirely unknown opponents. The games don't last very long, so this wouldn't leave much time for deducing the order of a partner.

I will be studying the following three agents:

- ToM0+ToM1: For each action, this agent averages the estimate from zeroth-order and first-order reasoning, and selects the action which maximizes this average. Let $Q_0(s,a)$ and $Q_1(s,a)$ be the expected value of a state-action pair according to ToM0 and ToM1 respectively.

$$V_{(0+1)} = \max_{a \in A}\left(\frac{Q_0(s,a) + Q_1(s,a)}{2}\right)$$
$$\pi_{(0+1)} = \arg\max_{a \in A}\left(\frac{Q_0(s,a) + Q_1(s,a)}{2}\right)$$

- ToM1+ToM2: For each action, this agent averages the estimate from first-order and second-order reasoning, and selects the action which maximizes this average. Let $Q_1(s,a)$ and $Q_2(s,a)$ be the expected value of a state-action pair

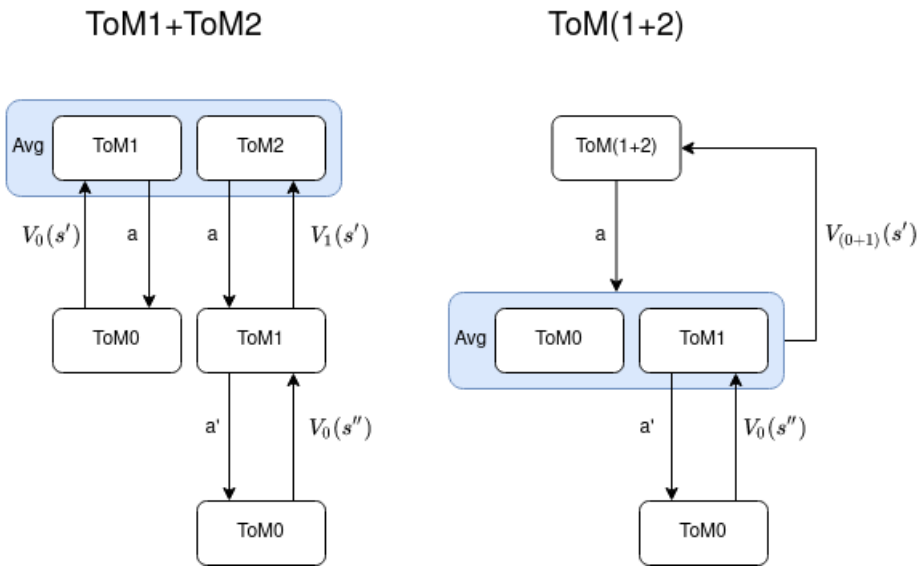


FIGURE 2.3: ToM1+ToM2 vs ToM(1+2). Both agents use a mixture of first-order and second-order reasoning, but differ in how they are combined. ToM1+ToM2 has two partner models which it references in order to build two independent estimates of the value of each state-action pair, which are then averaged. By contrast, ToM(1+2) only has a single partner model, namely that of a ToM0+ToM1 agent.

according to ToM1 and ToM2 respectively.

$$V_{(1+2)} = \max_{a \in A}\left(\frac{Q_1(s,a) + Q_2(s,a)}{2}\right)$$
$$\pi_{(1+2)} = \arg\max_{a \in A}\left(\frac{Q_1(s,a) + Q_2(s,a)}{2}\right)$$

- ToM(1+2): This agent assumes that $V_p = V_{(0+1)}$. Like ToM1+ToM2, this agent combines first-order and second-order reasoning, but instead of using two partner models, it uses a single model of a partner which is itself using mixed-order ToM. The differences between how these types of reasoning are aggregated is illustrated in Figure 2.3.

ToM0+

The ToM agents I describe search forward through the game tree and predict what they think their partner will do in the next turn. By searching through the game tree, however, they are also able to uncover other information without needing any theory of mind at all. "Dead ends", or parts of the game tree which contain only losses, can be avoided by looking ahead without making any assumptions about the behavior of the other player. For example, in a sample situation shown in Figure 2.4, an agent capable of looking one move further in the game tree is able to determine that one of their actions would lead to a certain loss without considering at all what their partner might do. To disambiguate the advantage a ToM agent has due to being able to avoid dead ends from the advantage from having a model of their partner, I am also including a ToM0+ agent, which despite having no model of their

partner, is able to search one move into the future to avoid dead ends, making no other assumptions about the behavior of their partner.[6]

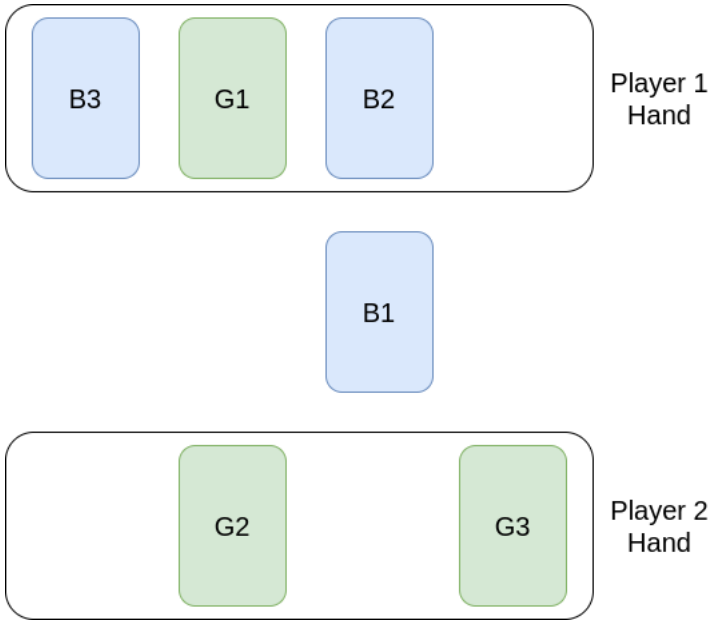


FIGURE 2.4: Sample situation where it is player 1's turn, there are no hints left, and it is assumed that both players have come to know the identity of their cards. If player 1 chooses to play B2, all of player 2's actions will cause a loss. Knowing this does not require theory of mind, but it does require being able to search one move further in the game tree.

2.3 Results

2.3.1 Training the Zeroth-order Agent

Training was done using MCTS to build a model. The quality of this model was evaluated by measuring the win rate achieved by two zeroth-order agents using the model to play the game. The training trajectory can be seen in Figure 2.5.

Four models were generated with different win rates (see Table 2.1). Each of these models is a tree generated by MCTS.

	Win Rate	Iteration (in thousands)	Nodes expanded
Model 1	0.3	1000	148547
Model 2	0.566	2000	200952
Model 3	0.629	3000	210786
Model 4	0.66	4000	213620

TABLE 2.1: Versions of the model logged for experiments. The number of nodes expanded represents the size of the tree generated by MCTS.

These models are zeroth-order models of the game, and are used to build all other ToM models of higher orders. The exact way the higher-order models are built is explained fully in Section 2.2.3.

[6] ToM0+ assumes the rules of the game are definite, and that the partner cannot take illegal moves.

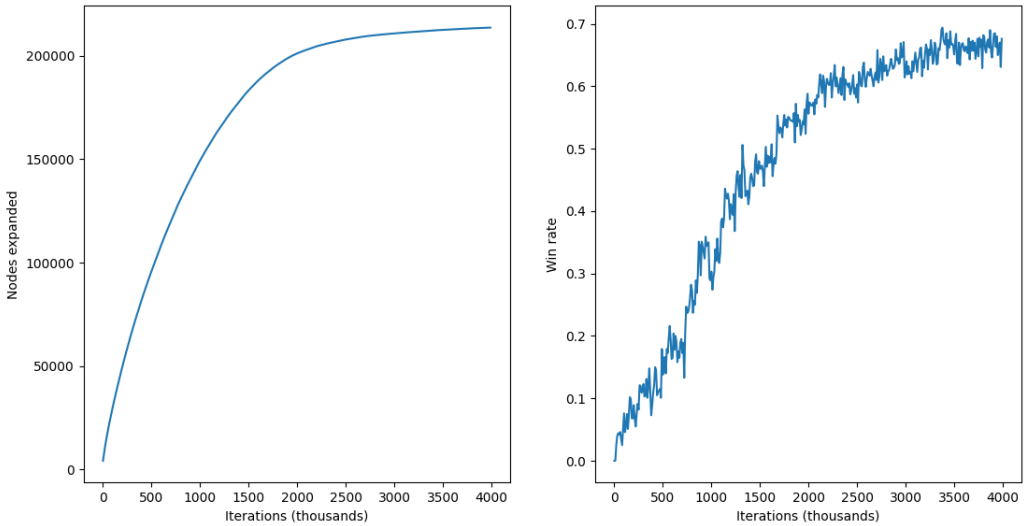


FIGURE 2.5: Results from training with MCTS for 4 million iterations. Iterations are represented here in thousands. The win rate every thousand iterations was calculated by taking average of 1000 games played by two zeroth-order agents using the model at that stage.

2.3.2 Testing Theory of Mind

To test the effect of ToM on gameplay, I tested every combination of the seven agent types for each of the four models produced by MCTS. Each configuration of agents played simplified Hanabi 10,000 times, and I recorded their win rates in Table 2.2. The coordination ability of the agent pairs varies a lot from the worst win rate of 0.31 to the best win rate of 0.96.

One of the first things to notice is that the win rate of the agent pairs does not always go up as the size of the zeroth-order model goes up. In a full 23 out of 49 pairs, the win rate using model 4 is actually *lower* than the win rate using model 1 (shown in Figure 2.2). The most extreme example is the pair where ToM0+ToM1 starts the game with ToM2 as their partner, where the accuracy drops by 16.4% between model 1 and model 4. The difference between the win rates is shown in Table 2.3.

Zeroth-order agents are always able to play better with larger models, because those models have better explored the game tree, and they directly query those models to determine which action has the highest expected value. Higher order agents do not directly query the model, but rather reason under the assumption that their partner will query the model or engage in their own ToM reasoning. As a result, the effect of a larger and more thorough model will not necessarily be straightforward. MCTS builds the estimates of expected value under an assumption that the game tree will continue to be explored in a certain way. When this assumption is violated, it is not necessarily the case that the estimates of expected value continue being as accurate as they were in the zeroth-order setting.

Another result is the difference in win rates which depends on which agent starts the game. This can be seen in the asymmetry along the diagonals in Table 2.2, or in Table 2.4. Specifically, in the games with either ToM0 or ToM0+ToM1 this difference is especially extreme. When these agents play against each other the difference stays below 0.06, but when they play against other agents the difference becomes quite large, as shown in Figure 2.6. Both ToM0 and ToM0+ToM1 do much better when the agent they are paired with starts the game and this difference grows for larger

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.3061	0.4714	0.4082	0.4464	0.4588	0.448	0.4577
ToM0+	0.4723	0.7675	0.7413	0.8414	0.7997	0.7704	0.8131
ToM1	0.4836	0.8209	0.8082	0.8796	0.7783	0.7774	0.9006
ToM2	0.4703	0.8945	0.8765	0.9617	0.8462	0.9527	0.9478
ToM0+ToM1	0.5184	0.7809	0.7518	0.8543	0.7383	0.8377	0.8256
ToM1+ToM2	0.5074	0.9098	0.8942	0.9302	0.8672	0.9392	0.9616
ToM(1+2)	0.5552	0.9081	0.8787	0.928	0.8756	0.9357	0.9599

(A) Model 1

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.57	0.6171	0.5477	0.5603	0.6274	0.5323	0.5927
ToM0+	0.7818	0.8597	0.7861	0.8007	0.8233	0.799	0.8283
ToM1	0.7426	0.8405	0.8871	0.8987	0.8323	0.8957	0.9388
ToM2	0.6281	0.8396	0.8752	0.8697	0.8871	0.8936	0.9151
ToM0+ToM1	0.657	0.7455	0.7119	0.73	0.7183	0.7344	0.7576
ToM1+ToM2	0.7551	0.8817	0.885	0.9019	0.8738	0.8981	0.9199
ToM(1+2)	0.7341	0.8364	0.8804	0.9074	0.8602	0.8876	0.9252

(B) Model 2

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.628	0.6471	0.6008	0.5851	0.6406	0.6331	0.6688
ToM0+	0.8521	0.9112	0.797	0.7823	0.845	0.8177	0.8276
ToM1	0.7825	0.8514	0.8928	0.849	0.8534	0.8787	0.9021
ToM2	0.7113	0.8405	0.8881	0.8654	0.8892	0.9305	0.9197
ToM0+ToM1	0.6882	0.716	0.657	0.6867	0.7158	0.7044	0.7372
ToM1+ToM2	0.7743	0.8377	0.88	0.8554	0.8351	0.8719	0.894
ToM(1+2)	0.7729	0.8227	0.8753	0.8746	0.8535	0.8707	0.8917

(C) Model 3

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.661	0.6747	0.6216	0.5937	0.6573	0.628	0.6713
ToM0+	0.8686	0.9093	0.8319	0.8139	0.8717	0.8471	0.9222
ToM1	0.7847	0.8366	0.9029	0.8484	0.8318	0.8836	0.8913
ToM2	0.707	0.8389	0.8885	0.8601	0.879	0.9185	0.9294
ToM0+ToM1	0.6699	0.68	0.6511	0.6903	0.6771	0.7006	0.7134
ToM1+ToM2	0.7821	0.8364	0.8782	0.8615	0.8367	0.8604	0.8889
ToM(1+2)	0.7989	0.8574	0.8985	0.8904	0.8798	0.909	0.9206

(D) Model 4

TABLE 2.2: Win rate with various pairs of agents. Rows represent the agent who starts the game.

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.3549	0.2033	0.2134	0.1473	0.1985	0.1800	0.2136
ToM0+	0.3963	0.1418	0.0906	**-0.0275**	0.0720	0.0767	0.1091
ToM1	0.3011	0.0157	0.0947	**-0.0312**	0.0535	0.1062	**-0.0093**
ToM2	0.2367	**-0.0556**	0.0120	**-0.1016**	0.0328	**-0.0342**	**-0.0184**
ToM0+ToM1	0.1515	**-0.1009**	**-0.1007**	**-0.1640**	**-0.0612**	**-0.1371**	**-0.1122**
ToM1+ToM2	0.2747	**-0.0734**	**-0.0160**	**-0.0687**	**-0.0305**	**-0.0788**	**-0.0727**
ToM(1+2)	0.2437	**-0.0507**	0.0198	**-0.0376**	0.0042	**-0.0267**	**-0.0393**

TABLE 2.3: Difference between the win rate with model 4 and the win rate with model 1 (both show in Table 2.2) In 23 out of 49 cases the win rate is lower with model 4. Negative results are shown in bold.

models. In nearly all other pairs of agents, the difference in win rate stays below 0.1 with the following exceptions where ToM1+ToM2 does significantly better if it starts the game:

- ToM1+ToM2 with ToM0+: The difference only appears for model 1 (0.1394) and disappears for the larger models.
- ToM1+ToM2 with ToM1: The difference only appears for model 1 (0.1168) and disappears for the larger models.

It's not entirely clear why ToM0 and ToM0+ToM1 play so much worse when they start the game. Games of simplified Hanabi typically last about 10 turns (in fact games which last longer than 10 turns must necessarily end in a loss), and so by starting the game an agent will likely end up taking one turn more than their partner, and if their decisions are higher quality than their partner this would be a good thing. It could also be that some early moves happen to critically affect the later stages of the game, and that both ToM0 and ToM0+ToM1 make poor early game decisions. Perhaps also worth noting is that ToM0 and ToM0+ToM1 are the only two agents which are capable of getting caught in a "dead end" as described in Figure 2.4.

Table 2.5 shows the results where this asymmetry is removed by averaging the win rate between games where either agent starts. This allows us to directly see, for each agent, which agent is the best partner. The top performing partner for each agent is shown in bold.

It appears that in general, ToM0+ is the best partner for zeroth-order models. This is interesting, because it suggests that when partnered with a zeroth order agent, ToM reasoning provides no advantage, and changes in behavior caused by ToM are actually a disadvantage. One exception is the case with the smallest model, in which higher-order agents significantly outperform ToM0+. Perhaps as the models grow, they eventually define a joint policy which is near-optimal, and deviating from that policy in any way is consistently worse. However, ToM0+ playing against itself with the largest model (Model 4) does not achieve the highest win rate of all pairs, which should cast some doubt on this claim. If the theory about zeroth-order agents forming a near-optimal policy were correct, I would have expected that ToM0+ playing against itself would eventually overtake all other agent pairs as model size increased. Future research could continue MTCS beyond 4000 thousand iterations to see if this does happen.

For larger models, ToM1 seems to be able to play quite well with itself, and is its own best partner. This is surprising, because in this case both agents are modeling

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2
ToM0+	0.009					
ToM1	0.0754	0.0796				
ToM2	0.0239	0.0531	-0.0031			
ToM0+ToM1	0.0596	-0.0188	-0.0265	0.0081		
ToM1+ToM2	0.0594	**0.1394**	**0.1168**	-0.0225	0.0295	
ToM(1+2)	0.0975	0.095	-0.0219	-0.0198	0.05	-0.0259

(A) Model 1

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2
ToM0+	**0.1647**					
ToM1	**0.1949**	0.0544				
ToM2	0.0678	0.0389	-0.0235			
ToM0+ToM1	0.0296	-0.0778	**-0.1204**	**-0.1571**		
ToM1+ToM2	**0.2228**	0.0827	-0.0107	0.0083	**0.1394**	
ToM(1+2)	**0.1414**	0.0081	-0.0584	-0.0077	**0.1026**	-0.0323

(B) Model 2

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2
ToM0+	**0.2050**					
ToM1	**0.1817**	0.0544				
ToM2	**0.1262**	0.0582	0.0391			
ToM0+ToM1	0.0476	**-0.1290**	**-0.1964**	**-0.2025**		
ToM1+ToM2	**0.1412**	0.0200	0.0013	-0.0751	**0.1307**	
ToM(1+2)	**0.1041**	-0.0049	-0.0268	-0.0451	**0.1163**	-0.0233

(C) Model 3

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2
ToM0+	**0.1939**					
ToM1	**0.1631**	0.0047				
ToM2	**0.1133**	0.0250	0.0401			
ToM0+ToM1	0.0126	**-0.1917**	**-0.1807**	**-0.1887**		
ToM1+ToM2	**0.1541**	-0.0107	-0.0054	-0.0570	**0.1361**	
ToM(1+2)	**0.1276**	-0.0648	0.0072	-0.0390	**0.1664**	0.0201

(D) Model 4

TABLE 2.4: Difference in win rate resulting from which agent starts the game. Each entry is a pair where the agent given by the row starts, and the agent given in the column plays second. What is measured is the win rate for that pair minus the win rate where the order of the agents is reversed. Cases where the difference is greater than 0.1 are in bold.

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.3061	0.4719	0.4459	0.4584	0.4886	0.4777	0.5065
ToM0+	0.4719	0.7675	0.7811	0.8680	0.7903	0.8401	0.8606
ToM1	0.4459	0.7811	0.8082	0.8781	0.7651	0.8358	0.8897
ToM2	0.4584	**0.8680**	0.8781	**0.9617**	0.8503	0.9415	0.9379
ToM0+ToM1	0.4886	0.7903	0.7651	0.8503	0.7383	0.8525	0.8506
ToM1+ToM2	0.4777	0.8401	0.8358	0.9415	**0.8525**	0.9392	0.9487
ToM(1+2)	**0.5065**	0.8606	**0.8897**	0.9379	0.8506	**0.9487**	**0.9599**

(A) Model 1

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.5700	0.6995	0.6452	0.5942	0.6422	0.6437	0.6634
ToM0+	**0.6995**	**0.8597**	0.8133	0.8202	0.7844	0.8404	0.8324
ToM1	0.6452	0.8133	0.8871	0.8870	0.7721	0.8904	0.9096
ToM2	0.5942	0.8202	0.8870	0.8697	0.8086	0.8978	0.9113
ToM0+ToM1	0.6422	0.7844	0.7721	0.8086	0.7183	0.8041	0.8089
ToM1+ToM2	0.6437	0.8404	0.8904	0.8978	0.8041	0.8981	0.9038
ToM(1+2)	0.6634	0.8324	**0.9096**	**0.9113**	**0.8089**	**0.9038**	**0.9252**

(B) Model 2

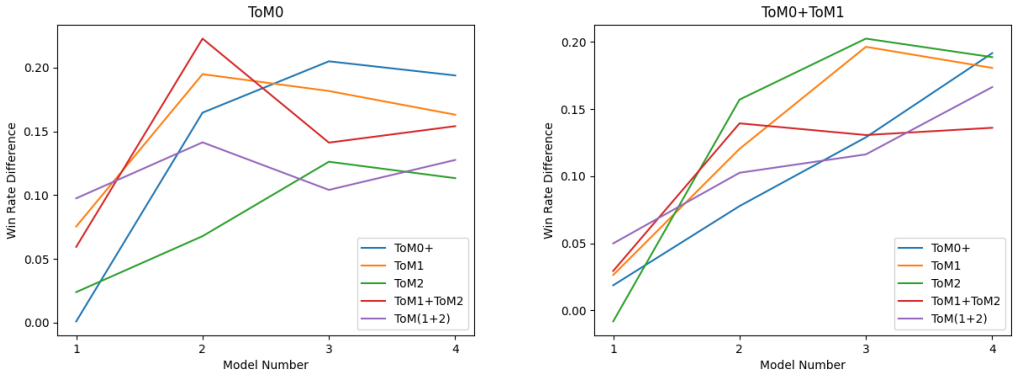
	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.6280	0.7496	0.6917	0.6482	0.6644	0.7037	0.7209
ToM0+	**0.7496**	**0.9112**	0.8242	0.8114	0.7805	0.8277	0.8252
ToM1	0.6917	0.8242	**0.8928**	0.8686	0.7552	0.8794	0.8887
ToM2	0.6482	0.8114	0.8686	0.8654	0.7880	**0.8930**	**0.8972**
ToM0+ToM1	0.6644	0.7805	0.7552	0.7880	0.7158	0.7698	0.7954
ToM1+ToM2	0.7037	0.8277	0.8794	0.8930	0.7698	0.8719	0.8824
ToM(1+2)	0.7209	0.8252	0.8887	**0.8972**	**0.7954**	0.8824	0.8917

(C) Model 3

	ToM0	ToM0+	ToM1	ToM2	ToM0+ToM1	ToM1+ToM2	ToM(1+2)
ToM0	0.6610	0.7717	0.7032	0.6504	0.6636	0.7051	0.7351
ToM0+	**0.7717**	**0.9093**	0.8343	0.8264	0.7759	0.8418	0.8898
ToM1	0.7032	0.8343	**0.9029**	0.8685	0.7415	0.8809	0.8949
ToM2	0.6504	0.8264	0.8685	0.8601	0.7847	0.8900	0.9099
ToM0+ToM1	0.6636	0.7759	0.7415	0.7847	0.6771	0.7687	0.7966
ToM1+ToM2	0.7051	0.8418	0.8809	0.8900	0.7687	0.8604	0.8990
ToM(1+2)	0.7351	0.8898	0.8949	**0.9099**	**0.7966**	**0.8990**	**0.9206**

(D) Model 4

TABLE 2.5: Win rates with various pairs of agents. Symmetry has been established by averaging the win rates between games where one or the other agent starts. For each column, the agent with the highest win rate is bolded.



(A) Difference in win rate between cases where ToM0 does not start the game and cases where ToM0 does start the game. In all cases ToM0 does better when the other agent starts the game.

(B) Difference in win rate between cases where ToM0+ToM1 does not start the game and cases where ToM0+ToM1 does start the game. In all cases but one, ToM0+ToM1 does better when the other agent starts the game.

FIGURE 2.6: Both ToM0 and ToM0+ToM1 perform best when the other agent starts the game. This difference grows as model quality improves.

the other as a zeroth-order agent, and are therefore both making incorrect assumptions about how the other will behave. What is also odd is that the same is not true for ToM0+ToM1, which uses a mixture of the expected value estimates produced by ToM0 and ToM1, as it plays relatively poorly with itself as a partner.

Another interesting result is that the top average score of 0.9617 is achieved by ToM2 playing itself with the smallest model, but that this high win rate monotonically decreases as the model size increases, bottoming out at 0.8601. A similar thing is true for ToM1+ToM2, and those results are shown in Figure 2.7. This matches the previous trend that ToM is less useful for larger models, but what's surprising about the result is that ToM2 playing against itself gets not only a higher win rate than any other pair using Model 1, but a higher win rate than any pair using any model. Just like before with ToM1, both agents have a completely incorrect model of their partner, and are reasoning based on incorrect assumptions. Somehow, both agents reasoning under totally incorrect assumptions leads to behavior which is near optimal.

The agent which is most often the best partner is ToM(1+2). ToM(1+2) is either the best partner, or close to the best partner, for all agents which are not zeroth order. This is true for both smaller and larger models. This suggests that using mixed-order strategies might be really advantageous, but the precise nature of that mixed-order strategy matters a great deal. ToM(1+2) almost always outperforms ToM1+ToM2, despite both using a combination of first-order and secord-order reasoning (differences highlighted in Figure 2.3). Furthermore, ToM0+ToM1 does quite poorly, nearly always getting a worse win rate than either ToM0+ or ToM1, often by more than 0.1.

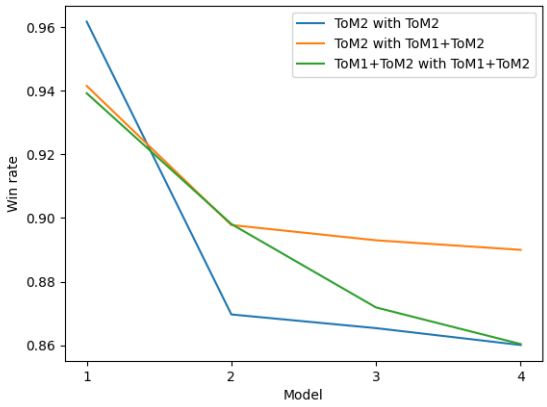


FIGURE 2.7: Win rate of ToM2 and ToM1+ToM2 decreasing monotonically as model size increases as it plays with either ToM2 or ToM1+ToM2 as a partner.

Chapter 3

Cooperative Reinforcement Learning

In this chapter I will explore the effect of ToM reasoning on the full version of Hanabi by empirically testing an RL agent augmented to use ToM. In Section 3.1 I will explain in further detail the challenge of designing such agents, the past work in this area, and the architecture ToMNet developed by Rabinowitz et al. (2018) which I will use as inspiration for my own architecture, RainbowToM, a modification of the RainbowDQN algorithm explained in full detail in Section 3.1.3. In Section 3.2 I give an overview of the Hanabi Learning Environment in which my RL agents will play and obtain reward, describe RainbowToM in detail as well as a baseline architecture called RainbowZero, as well as a series of rule-based agents which will be use be used to produce a multi-agent environment for training and testing. Lastly, in Section 3.3 I will present the results of the experiments, as well as do some data analysis to probe into the latent representations formed by RainbowToM.

3.1 Introduction

Bard et al. (2020) divides the challenge of training RL agents to play Hanabi into two parts:

- **Self-Play Learning**: The agent plays games against copies of itself. Training optimizes a joint policy to maximize reward.
- **Ad-hoc Teams**: The agent learns to play against a wide range of agents and plays no more than ten games against the same opponent. Training optimizes a general policy which is able to coordinate with broad set of possible partners.

There exist near-optimal heuristic policies for self-play. Cox et al. (2015) designed a heuristic strategy which is able to achieve a perfect score of 25 points 76% of the time with an average score of 24.68 points in self-play. On the other hand, ad-hoc play remains very difficult for artificial agents (Walton-Rivers, Williams, and Bartle, 2019). In the ad-hoc case an agent must tailor its strategy to the particular partner it's playing with, which requires developing a model of its partner.

In the self-play RL, the training regime is a process which searches for a joint policy, or protocol, which the agents can follow. Such highly specialized protocols only work when both agents are following the protocol, and so agents trained in self-play don't do well in the ad-hoc setting (Bard et al., 2020). I expect that ToM is useful primarily in the ad-hoc case, where an agent can't just follow a fixed protocol. For this reason, I will be training agents in self-play in order to establish a baseline and to check that the architectures I am using work, but my main experiment will be investigating ad-hoc play against a set of different agent types.

For my experiments I will be comparing a baseline agent based on the RL agents used by Bard et al. (2020), which I will call RainbowZero, with an agent that has been augmented in order to do ToM reasoning in an ad-hoc regime, which I will call RainbowToM.

3.1.1 Related Work

In Bard et al. (2020), the authors train reinforcement learning agents in self-play, achieving an average score of 20.64 in two-player Hanabi. They also run some ad-hoc experiments demonstrating that agents trained in self-play perform very poorly when placed in an ad-hoc regime. Ad-hoc play is more extensively tackled by Canaan et al. (2020) by training reinforcement learning agents in environments with different combinations of rule-based agents, and evaluating how well different agent-types can coordinate with one another. None of these agents, however, explicitly use theory of mind.

In order to directly implement first-order ToM in reinforcement learning agents playing Hanabi, Fuchs et al. (2021) design an algorithm which builds a Monte-Carlo approximation of the belief of the partner by sampling hands from the possible hands the agent might be holding (and thus the partner might be observing), and thereby estimating what the partner might believe about their own cards. In self-play they demonstrate that their method improves on the standard RL agents trained by Bard et al. (2020), but they do no experiments in an ad-hoc setting.

Another relevant implementation of first-order ToM in RL, though not on Hanabi, is a paper by Nguyen et al. (2022). The authors augment an RL agent by adding a ToM model which was used to predict the goal, intention, and next action of the other agent, which are all then used as features by an actor-critic RL learner. They then show that their augmented agent, Trait-ToM, is able to use ToM to better assist other agents in achieving their goals.

3.1.2 Machine Theory of Mind

In order to study ToM in the context of reinforcement learning I will be building an RL agent which has been augmented such that a part of its own internal state will directly model the internal state of its partner. This augmented agent will be called RainbowToM, the design of which will be laid out thoroughly in Section 3.2.2. The design of RainbowToM is based on an architecture by Rabinowitz et al. (2018) called ToMnet.

ToMnet was designed to learn to predict the behavior of a family of partially observable Markov decision processes (POMDPs) by modeling them in real time. ToMnet did this by learning to produce, for an arbitrary POMDP, a *character embedding* representing the general unchanging properties of a particular POMDP as well as a *mental embedding* representing the changing internal state of the POMDP. These embeddings were then used, in conjunction with the current state, to predict the behavior of the POMDP. The architecture, shown in Figure 3.1, uses the character embedding to inform the mental embedding, and then uses both to inform the prediction. The authors then show that these embeddings contain meaningful structure, and that ToMnet was able to pass the "Sally-Anne" test (Wimmer and Perner, 1983), accurately modeling others as holding false beliefs, a classic ToM task .

ToMnet was designed to make predictions about the behavior of agents, but not to take action itself. RainbowToM will build both a character and a mental embedding analogous to that of ToMnet, but will then also use those embeddings to help

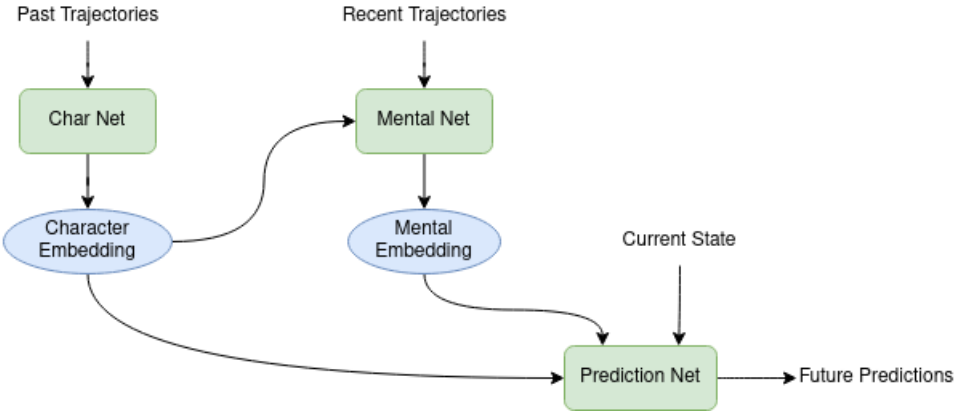


FIGURE 3.1: ToMnet architecture from Rabinowitz et al. (2018). The figure has been simplified slightly from the original paper for clarity.

make decisions about which action to take. In this way, I hope to build an RL agent capable of using first-order beliefs to better cooperate with its partner.

3.1.3 Reinforcement Learning with RainbowDQN

To perform experiments in reinforcement learning I will be using a variation of the RainbowDQN algorithm (Hessel et al., 2018), which is a state of the art variation on the DQN algorithm (Mnih et al., 2015), which is a method for doing Q-learning (Watkins and Dayan, 1992).

Q-learning

In reinforcement learning, an agent must be taught to take actions that, in expectation, obtain high reward. This can be framed as the search for an optimal policy, where a policy is the function that takes in the current state, and returns an action. There are two main types of functions which can be learned to attempt to achieve this:

- A policy function $\pi : S \to A$ directly maps states to actions.
- A value function $V : S \to \mathbb{R}$ maps to states to real numbers representing the expected discounted reward achieved from that state.

A value function can be used to generate a policy by selecting the action which results in the state with the highest value. This requires an agent know the transition function given an action and a state in order to determine the expected value of a particular action. This is called "model based" learning. A policy function does not require any knowledge about the transition function, and so is a kind of "model free" learning.

Q-learning is a kind of model free learning which combines the policy function and the value function into one function called the Q function. The function $Q : (S, A) \to \mathbb{R}$ estimates the expected discounted reward of state action pairs:

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma R(s_{t+1}, a_{t+1}) + \gamma^2 R(s_{t+2}, a_{t+2}) \dots$$

Here $R(s, a)$ represents the reward obtained from taking action a in state s, and $\gamma \in (0, 1)$ is the discount factor. This infinite sum can be more compactly written as:

$$Q(s_t, a_t) = R(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+1})$$

To estimate the Q values, Q-learning uses a variation of the Bellman equation. Supposing that an agent was in state s_t, took action a_t, and ended up in state s_{t+1}, they can use what they observe to update their Q function:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha(R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$

Here $\alpha \in (0, 1)$ is a learning rate, and the term $R(s_t, a_t) + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)$ represents the TD error of the initial estimate $Q(s_t, a_t)$. By iteratively updating the value of Q as an agent takes actions in an environment, the function should eventually converge to a stable point.

A policy π can be obtained from a Q-function by selecting at every state the action which maximizes the Q-function:

$$\pi(s) = \arg\max_a Q(s, a)$$

Deep Q-learning

Mnih et al. (2015) developed an algorithm used to do Q-learning which produces a neural network commonly referred to as a DQN. The DQN takes in a vector representing the state of the environment and returns a vector representing the Q value for each action. The training process has two main features distinguishing it from other Q-learning algorithms:

1. **Experience Replay** The DQN is trained online, which means that the network is being used to generate new data while simultaneously being trained on that data. The agent uses the DQN to select an action in the environment, and then records the transition resulting from that action in memory. Batches of transitions are then sampled randomly from memory and used to train the DQN. This allows the DQN to learn from past experiences and avoids the problem of experiences being too heavily correlated with each other. The replay memory is finite, so old experiences are eventually forgotten, but typically the replay memory is quite large such that the agent has a rich and varied experience to sample from.

2. **Target Network** The Bellman equation used to update the Q function is modified to make use of a target network Q_T:

   $$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha(R(s_t, a_t) + \gamma \max_a Q_T(s_{t+1}, a) - Q(s_t, a_t))$$

   The target network is an old version of the DQN which is only periodically synchronized with the current parameters. Using the target network to estimate the TD error provides stability, because it prevents oscillations of the Q function resulting from a moving target.

By implementing this method of Q-learning, Mnih et al. (2015) demonstrated that a DQN was capable of playing a wide range of games, and was at the time of publication a state of the art reinforcement learning method.

Rainbow

While the DQN was state of the art for its time, a number of improvements were made since then:

1. **Double DQN**: In the Bellman equation used by the DQN, the same function is used to select the next action $a_{t+1} = \arg\max_a Q(s_{t+1}, a)$ as well as estimate the value of taking that action $Q(s_{t+1}, a_{t+1})$. This can lead to systematic overestimation of the actual discounted reward. Van Hasselt, Guez, and Silver (2016) solve this problem by using the online DQN to select the next action, but using the target DQN to estimate the value of that action:

$$Q'(s_t, a_t) = Q(s_t, a_t) + \alpha(R(s_t, a_t) + \gamma Q_T(s_{t+1}, \arg\max_a Q(s_{t+1}, a)) - Q(s_t, a_t))$$

2. **Prioritized Experience Replay**: In the standard experience replay, experiences are sampled randomly according to a uniform distribution. Some experiences are more informative or surprising than others, and Schaul et al. (2016) reasoned that by prioritizing those experiences which result in the largest TD error, the DQN can be trained more efficiently. They do this by sampling experiences with the following probability:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

   The priority of each experience i is given by $p_i = |\delta_i| + \epsilon$ where δ_i is the TD error and ϵ is a small positive constant. A constant α represents how much prioritization is used, where $\alpha = 0$ causes a uniform distribution. [1]

   A side effect of doing this is that now experiences will become more correlated, which was the initial reason to sample uniformly from memory. To counteract this, the authors use importance-sampling weights:

$$w_i = \left(\frac{1}{N}\frac{1}{P(i)}\right)^\beta$$

   Instead of updating the DQN based on the TD error δ_i, each experience is instead updated based on $w_i\delta_i$ to scale the size of the update, where $\beta = 1$ fully corrects for the the correlation.

3. **Categorical DQN**: Implicitly, there exists some distribution of possible future rewards, and the Q function estimates the expectation over that distribution. The Q function is therefore unable to distinguish between distributions which have the same expectation. For example, suppose that future reward is bimodally distributed such that when the agent takes an action, it either results in a lot of reward, or very little. Contrast that to a situation where the reward is

[1] Note that this term α is a separate hyperparameter than the learning rate, which is also given by α. I chose to use α for both to stay consistent with the original papers, and because both improvements can be understood separately.

normally distributed such that it generally gets a moderate amount of reward. To a Q function, these distributions are identical if the expectation is the same.

To allow an agent to model the full distribution of possible future reward, Bellemare, Dabney, and Munos (2017) develop what is called Distributional Reinforcement Learning. Instead of asking the network to predict a single Q value for each state action pair, the distribution is partitioned into bins, and a vector is output representing the probability of the total discounted reward being in each bin.

TD error is redefined to be the KL-divergence[2] between the distribution that the network outputs and a target distribution. The target distribution is formed by scaling the distribution over the next state s_{t+1} by the discount factor γ, shifting it by the observed reward $R(s_t, a_t)$, and then projecting that distribution onto the fixed support which defines the bins.

4. **N-step Learning**: In learning to estimate the value of a state action pair, there exists a tradeoff between two approaches. On the one hand, to obtain the most accurate target possible, one could wait until a full trajectory has been completed in order to calculate the true value of that trajectory. This is advantageous because the signal being trained on is maximally accurate. The disadvantage is that for longer trajectories consisting of many actions, it becomes difficult to assign credit to any particular action for causing the final outcome. The solution to this is TD learning, where in each step the agent learns to improve its current estimate $Q(s_t, a_t)$ by bootstrapping from a more accurate estimate it obtains in the next timestep:

$$Q'(s_t, a_t) \leftarrow R(s_t, a_t) + \gamma Q(s_{t+1}, a_{t+a})$$

This is how a Q function is typically trained. While TD learning can assign credit directly to each action, it does so at the expense of using a less accurate target.

A middle ground between these two extremes is to use N-step learning, where the Q function is still trained using TD learning, but a more accurate estimate is used from N time steps in the future:

$$Q'(s_t, a_t) \leftarrow \sum_{i=0}^{N} \gamma^i R(s_{t+i}, a_{t+i}) + \gamma^N Q(s_{t+N}, a_{t+N})$$

By selecting an appropriate value for N, one can obtain benefits of both approaches (Sutton, 1988).

Each of these alterations of the DQN solves a unique problem and offers some improvement, and the insight of Hessel et al. (2018) was to combine all these variations into one algorithm that they call the RainbowDQN, which is currently a state of the art Q-learning algorithm. [3]

[2] Kullback-Leibler (KL) divergence, also called "relative entropy", is a method to compare two probability distributions introduced by Kullback and Leibler (1951).

[3] In the paper by Hessel et al. (2018) they include two other improvements, DuelingDQN and NoisyDQN, into their implementation of Rainbow. Not all versions of Rainbow include all DQN variations, and because neither I nor Bard et al. (2020) use DuelingDQN or NoisyDQN, I omitted them here.

3.2 Methods

3.2.1 Hanabi Learning Environment

For all experiments using the full version of Hanabi, I will be using the Hanabi Learning Environment developed by Deepmind (Bard et al., 2020). This is a game environment inspired by OpenAI Gym which allows agents to interact with the game through a clearly defined input-output channel (Brockman et al., 2016). The environment contains the full state of the game as well as rules for how that state is allowed to change, and can be updated by means of the reset and step.

environment.reset() When the environment is reset, it starts a new game, initializing the game state. For Hanabi, this means resetting the hint tokens, disaster tokens, shuffling the deck, and dealing cards to each of the players. The function then returns what each player is allowed to observe, which player's turn it is, as well as which actions that player can legally take.

environment.step(action) The environment can be changed by calling the step function, providing the action taken by whichever player is in their turn. The environment once again returns the observations, which player's turn it is, and the legal actions from the new game state. The environment also returns a reward, which can be used to train an RL agent, as well as a whether or not the game has finished. This function can be called until the game ends.

Actions For two-player Hanabi, there are twenty different actions an agent could take, of which only a subset are legal in each turn. These correspond to five playing actions, five discarding actions, and ten hinting actions. To select an action, an agent must provide to the environment an integer between one and twenty which is within the subset of legal moves.

Observations The Hanabi Learning Environment provides an object containing observations in a variety of formats and levels of detail, but for the purpose of Reinforcement Learning, the relevant observation they provide is an observation vector, consisting of (for two-player Hanabi) 658 ones and zeros, which describes everything a player is able to observe. This includes, it should be noted, not only what is technically visible, but also a record of which hints have been received by that player in past turns, as well as the last action taken by their partner. It is from this vector that an RL agent must decide which action to take.

Reward In Hanabi there is a maximum score of twenty-five points and a minimum score of zero. As the game progresses the group collects points by successfully playing cards, but each time they unsuccessfully play a card they use up a disaster token. If all disaster tokens are used up, they lose everything and finish the game with zero points. The Hanabi Learning Environment deals with this by providing one unit of reward for every card successfully played, and a negative reward equal and opposite to the total reward collected so far if and when the final disaster token is used up. This causes the game to finish with zero total reward, but localizes the reward to the exact moments where points are gained or lost. This means that in any given turn reward can only range from -24 to +1.

Reinforcement Learning Environment

The Hanabi Learning Environment also contains an implementation of the RainbowDQN learning algorithm based on the Dopamine framework by Castro et al. (2018), which is a standardized framework meant to help users quickly prototype reinforcement learning algorithms and test ideas. I will be using a version of this code which I have modified in order to run my experiments.

3.2.2 RainbowToM

Q-learning is model free, and so the agent does not explicitly model their environment, nor other agents present in their environment. A Q-function only predicts the value achieved by taking an action from a particular state. In order to study Theory of Mind in the full Hanabi game, I have designed an algorithm which is explicitly asked to build a model of their partner, which I call RainbowToM.

The RainbowDQN receives observations as input, does some computation with some number of hidden layers, and finally outputs a vector which is used to define the Q-function, predicting how much value each move will produce. The activation pattern present in each hidden layer of the network describes what features the agent has extracted from the observation it receives, or rather, the agent's "belief" about the true state of the game. RainbowToM splits this belief into three parts with three unique tasks:

- Zeroth-order Embedding (B0): This vector has the same purpose as a hidden layer in RainbowDQN, to extract useful information from the observation to be used in computing Q values. This embedding will be of length 512, which is the size of the hidden layer used by Bard et al. (2020) for their RainbowDQN agents.

- Character Embedding (B1-C): This vector's purpose is to describe the agent's partner in order to better predict their behavior. The "true" character of the partner is assumed to be unchanging, and so the character embedding is intended to approximate and predict the true character of the partner. This embedding will be of length 8, which is the size of the embedding space used by Rabinowitz et al. (2018).

- Mental Embedding (B1-M): This vector is intended to capture information about the partner in order to predict their behavior, but is not assumed to be unchanging. This vector represents the "mental state" of the partner, and is expected to change as the game is played. This embedding will be of length 8, which is the size of the embedding space used by Rabinowitz et al. (2018).

RainbowToM takes in the current observation (Obs) as well as the beliefs from the previous time step, and computes new beliefs (B0, B1-C, B1-M). These beliefs are then used to produce Q-values (Q), as well as predict the partner's next action (A). The architecture of RainbowToM is shown in figure 3.2.

In order to train the three belief embeddings, the algorithm makes use of three different loss functions:

1. Q-loss: This loss is used to train B0, and is identical to the loss function used to train RainbowDQN. This is the KL-divergence between the target distribution and the distribution of Q-value given by the network. This is called distributional RL, a visual of which is shown in Figure 3.3. The target distribution is

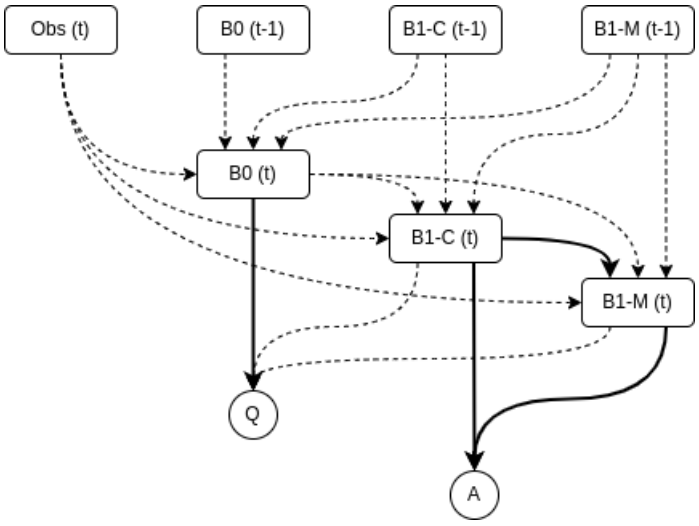


FIGURE 3.2: The RainbowToM network. Nodes of the graph represent vectors, and the lines between them represent fully connected neural networks. If the line is solid, during backpropagation the gradient continues back through the input, but if the line is dotted, backpropagation stops at the input.

formed by taking the distribution produced by the target network, shifting it by the reward obtained, and then projecting it onto the original support.

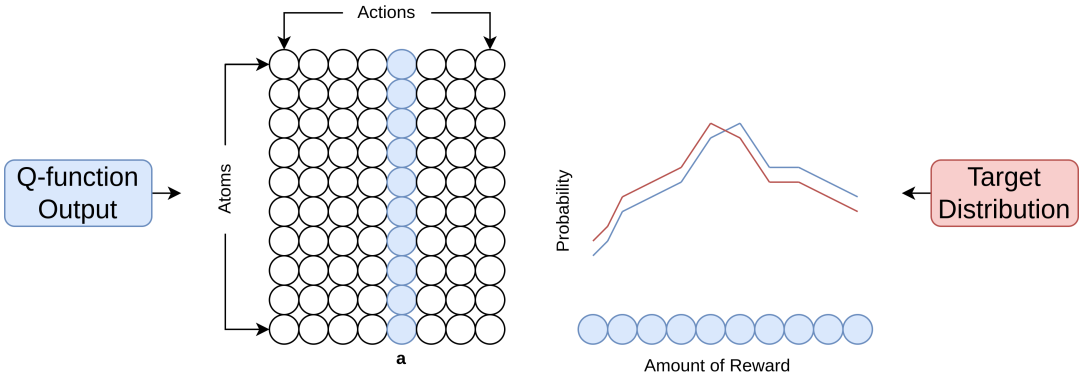


FIGURE 3.3: Q-loss: A distribution over possible future reward is formed from the vector corresponding to the action taken by the agent. This distribution is compared to the target distribution, and the KL-divergence between the two is used as a signal to train the network.

2. A-loss: This loss is used to train B1-C and B1-M. The network is asked to predict the partner's next action, and does so by outputting a 20 by 20 matrix representing the probability distribution over each of the partner's possible actions, conditional on each of the current player's possible actions. The length 20 vector corresponding to the action actually taken by the current player is the prediction distribution, and the loss is equal to the categorical cross entropy between that distribution and the one-hot encoding of the true action taken by the partner. A visual of this loss is given in Figure 3.4.

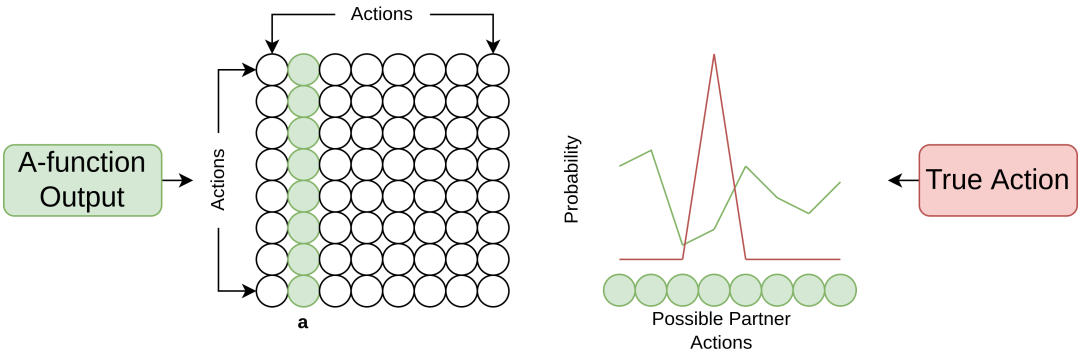


FIGURE 3.4: A-loss: A distribution over the possible actions the partner might take in the next turn is formed, conditional on the action actually taken by the current agent. This distribution is compared to the one-hot encoding of the true action taken by the partner in the next turn.

3. C-loss: This loss is used to train B1-C, and is used to disambiguate it from B1-M. While both B1-C and B1-M are used to predict the partner's next action, B1-C is assumed to be approximating something which is unchanging. C-Loss is the TD error between the current character embedding, and the character embedding given in the next timestep. If RainbowToM is playing multiple games against the same opponent, the character embedding may be preserved between games, and the C-loss also applied between the embeddings produced during the last step of a game and the first step of the next game. This also disambiguates the B1-C from B1-M in training, because both B0 and B1-M are always reset at the end of every game.

The network is used to produce a policy in the same way as the RainbowDQN, by selecting the action which obtains the highest value according to the Q-function. This means that the act of predicting the partner's action does not have any direct impact on the behavior of the agent. Instead, it forces the network to learn a representation of the partner, which will indirectly affect how the Q-function is calculated. Furthermore, by measuring the prediction accuracy, I can gauge the quality of that representation.

RainbowZero

I will be doing experiments with the model described in Figure 3.2 which I am calling RainbowToM. As a baseline I will also be training a model I am calling RainbowZero (shown in Figure 3.5), which is the same as RainbowToM, but without the character and mental embeddings. This architecture is still distinct from a standard RainbowDQN because of the zeroth-order embedding which may be used by the agent in the next timestep, but like the RainbowDQN, it will only be estimating the Q-function and nothing else.

The more thorough scheme for both model architectures is given in Figure 3.6. Because the character embeddings and mental embeddings are so short, the number of parameters for each architecture is still reasonably similar despite the difference in complexity. The hyperparameters used for both RainbowZero and RainbowToM are as follows[4]:

[4]These parameters are based on those used by Bard et al. (2020).

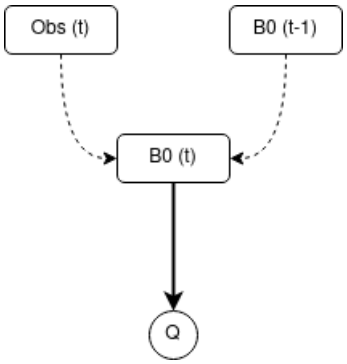


FIGURE 3.5: Architecture of RainbowZero, which is RainbowToM with the character and mental embeddings removed. All other details are kept identical.

- **Replay memory size**: 50000 samples
- α **(priority experience replay)**: 1
- β **(priority experience replay)**: 1
- **N-step horizon**: 1
- **Epsilon decay period**: 1000 steps
- **Target update period**: 500 steps
- **Distributional RL support**: 51 atoms
- **Time discount factor (γ)**: 0.99
- **Optimizer**: RMSProp
- **Learning rate**: 0.0025
- **L2 regularization**: 0.001

3.2.3 Troubleshooting

During the early experimentation with RainbowToM, I had a lot of trouble getting the architecture to learn properly. With only the Q-loss applied, RainbowToM was able to learn to achieve an average score during self-play of more than 15 points, but when A-loss was applied it was only able to achieve an average score of 4 points, and when C-loss was applied the network collapsed completely and was unable to learn anything at all. RainbowZero was able to learn just fine in self-play, and so I tried to investigate what was causing RainbowToM to fail.

The first thing I considered was how I was combining the loss functions. Each loss function has a very different magnitude.

- Q-loss: Here the magnitude should be small. This loss is the KL-divergence between two highly similar probability distributions.
- A-loss: This loss can have an arbitrarily high magnitude. The target is always a one-hot encoding, but the output will be a distribution of probability spread out over multiple possibilities. If the output assigns a very small probability to the action taken, the loss could be extremely large.

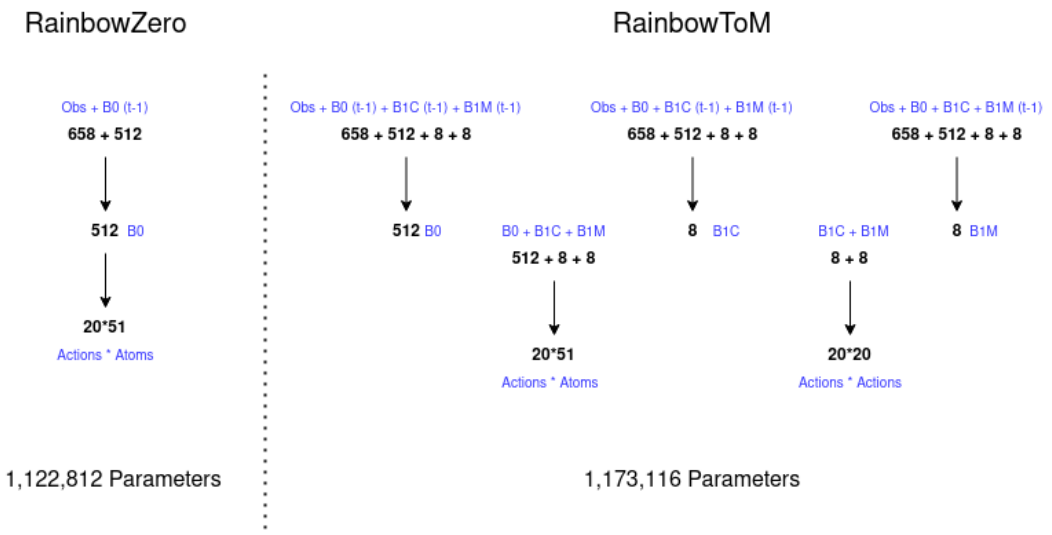


FIGURE 3.6: Scheme for both model architectures. The observation size is 658 for a two-player game in the Hanabi Learning Environment. Both architectures use distributional reinforcement learning, and so the Q-function output is divided not only the number of possible actions, but by the number of atoms used to describe the distribution of predicted reward. The number of atoms was chosen as 51 to stay in line with the paper by Bard et al. (2020).

- C-loss: This loss could also have a high magnitude, but it's not entirely clear what to expect. This depends on how much the character embedding changes from one time-step to the next. I would expect, certainly at first, that the embeddings are completely different, because they are features extracted from entirely different input. On the other hand, because the embedding is of length 8, this might lead the magnitude to remain reasonably low.

I was adding the losses together as a weighted sum: $L = w_q L_q + w_a L_a + w_c L_c$

I knew when w_a and w_c were both set to zero, RainbowToM was able to learn, so I figured that if the issue was the magnitude of the losses, then it should be possible to set w_a and w_c to exceptionally small values. The worst-case scenario of setting these values too small would be that the network would not be able to learn to predict the partner. What I found instead, was that any value greater that zero caused the network to fail just as before.

I next attempted to dynamically normalize the losses. Each loss was itself a sum across a batch of 32 samples, and so I could normalize according to either the average or the maximum loss across a batch. All attempts to dynamically normalize the loss in this either caused no change, or caused learning to catastrophically collapse.

I next began to explore deactivating parts of the network to discern where the problem was happening exactly. I started by turning off the C-loss and just exploring how I could get the network to learn with the A-loss applied. I knew that if A-loss was deactivated too that the network would learn just fine (see Figure 3.7). I first tried deleting the character and mental embeddings from the input by setting them to all zeros (see Figure 3.8). This architecture still failed to do significantly better than averaging 4 points per game.

I next tried completely severing the part of the network being trained by the Q-loss from the rest of the network by both setting the character and mental embedding inputs to zero as well as preventing the net from using the new embeddings in the

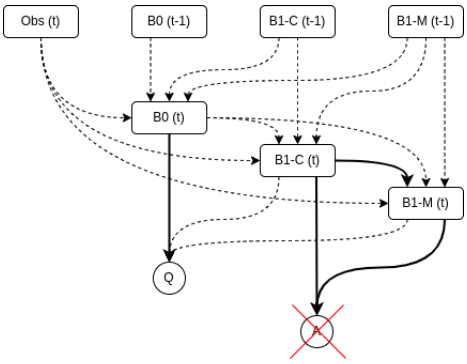


FIGURE 3.7: Turning off the A-loss function. When I do this, the algorithm successfully learns to play.

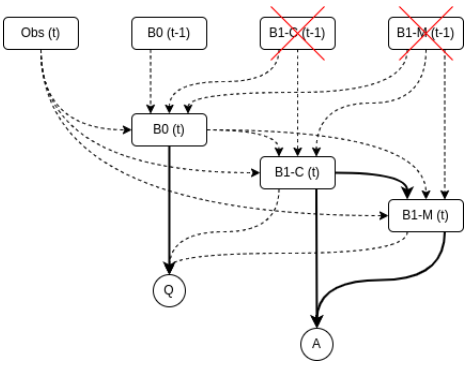


FIGURE 3.8: Setting the character and mental embedding inputs to zero. When I do this, the algorithm still fails to learn.

computation of the Q function (see Figure 3.9). This would cause the severed part of the network to be basically identical to RainbowZero, and indeed, in this case the network was able to learn just fine. When I tried only removing the character and mental embeddings from being used in the computation of the Q function, the network was somewhat improved and seemed able to learn, but did so at least twice as slowly and was quite unstable (see Figure 3.10).

From this I surmised that the problem was probably happening at the places where the two halves of the network connected to each other. All the neurons in the network are Rectified Linear Units (ReLU) which is a term introduced by Nair and Hinton (2010) to describe the activation function: $f(x) = max(0, x)$. Because the activation function is unbounded, hypothetically the activation of neurons can vary widely. Normally this isn't an issue, but because the two halves of the network are being trained by different loss functions, I hypothesized that the activations of the neurons on each side of the network might have very different magnitudes.

To deal with this I looked into layer normalization, which is a method developed by Ba, Kiros, and Hinton (2016) to normalize the activations of neurons. Across a layer of neurons, the mean μ and standard deviation σ are calculated for a single forward pass, and then the activations in that layer are immediately scaled by μ and σ.[5] This keeps the magnitude of the activations within the same approximate range. When I applied layer normalization to the character and mental embeddings being used in computing the Q function I observed marginal improvements (see Figure 3.11). When I applied layer normalization to all connection between the two halves

[5]This is distinct from other kinds of normalization, like batch normalization, which calculates μ and σ according to the activations produced from many samples and not just the current sample.

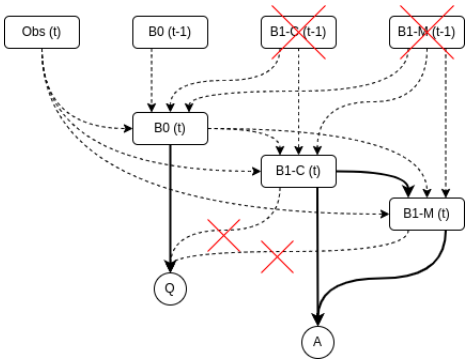


FIGURE 3.9: Setting the character and mental inputs to zero and simultaneously removing the character and mental embeddings from Q function calculation. When I do this, the algorithm successfully learns to play.

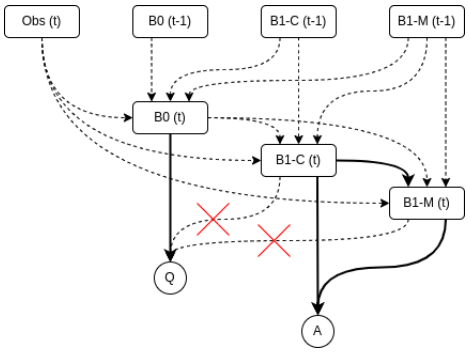


FIGURE 3.10: Removing the character and mental embeddings from the Q function calculation. When I do this, the algorithm does learn to play, but at least twice as slowly (and it becomes quite unstable).

of the network, as shown in Figure 3.12, the network was finally able to learn with all three losses applied.

3.2.4 Rule-based Agents

For my experiments I am defining a set of rule-based agents which play the game according to simple heuristics. These agents are designed according to my own experience playing the game, as well as the conventions described by *H-group conventions* (2022). Each agent has the same basic decision pipeline:

1. If a card is known explicitly to be playable, play the card.
2. If a card is known explicitly to be safely discardable, discard the card. Safely discardable means that an identical card has already been successfully played, and so there is no harm in discarding it. (Note that this only applies if there are less than eight hint tokens available, otherwise the discard action is not legal according to the rules of Hanabi, explained in Section 1.3)
3. If it is possible to give a hint, give a hint.
4. Discard the oldest card in the hand

All of the agents follow this same pipeline, but differ from each other in four different ways:

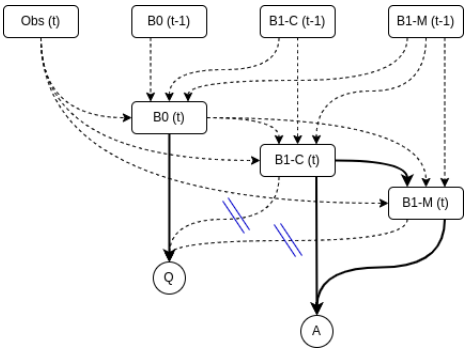


FIGURE 3.11: Applying layer normalization to the character and mental embeddings before use in Q function calculation. When I do this, the algorithm does learn to play, but at least twice as slowly. (similar to the case in Figure 3.10)

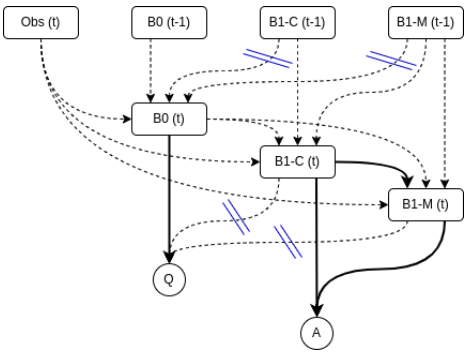


FIGURE 3.12: Applying layer normalization to the character and mental embeddings before use in Q function calculation, as well as to the character and mental embeddings of the previous timestep. This architecture was able to learn well with all three loss functions applied.

- **Risky**: Risky agents will play or discard in steps 1 and 2 if they are more than 50% confident that a card is playable or safely discardable. This is determined by checking which fraction of all possible cards (consistent with past hints) are currently playable or safely discardable.
- **Maxinfo**: Maxinfo agents will, when giving a hint, aim to maximize the amount of information their partner receives. This is measured by counting the total number of cards that are included in the hint and were also not known before. Agents which are not maxinfo agents will give an entirely random hint.
- **Intentional-Sending**: These agents will give hints that follow the protocol that if the most recently drawn card is included in the hint, it is playable. If the most recently drawn card is not playable, these agents will not include them in the hint (to avoid confusion). [6]
- **Intentional-Receiving**: These agents will assume the other player is using the protocol, and will infer that their most recently drawn card is playable if it is included in the hint.

[6]This is a simplified protocol based on conventions I've observed playing the game with friends. Similar protocols are also employed by an online community of Hanabi players. (*H-group conventions* 2022)

From these four character traits I define a set of nine unique agents (see Table 3.1).

	Risky	Maxinfo	IntSend	IntReceive
MaxSafe		X		
MaxRisk	X	X		
RandSafe				
RandRisk	X			
IntMaxSafe		X	X	X
IntMaxRisk	X	X	X	X
IntRandSafe			X	X
IntRandRisk	X		X	X
IntSuperSafe		X	X	

TABLE 3.1: Nine unique rule-based agents. 'X' means that the agent has a certain property.

3.2.5 Ad-hoc Environment

Ad-hoc Hanabi, as defined by Bard et al. (2020), is the regime where a player must play with a wide range of partners, and no more than 10 games are played with the same partner. This forces players to learn a general ability to coordinate with other players.

In order to test the ability of RainbowToM and RainbowZero to coordinate with arbitrary agents, I will be training and testing them in an environment where every 10 games they will be paired with a randomly selected rule-based agent from those defined in Section 3.2.4. They will then play 10 games with that partner, during which RainbowToM may develop a persistent character embedding which may be used between games. After those 10 games have been played, the character embedding is reset, and they are paired with a new random agent.

3.3 Results

3.3.1 Rule-based Agents

In Table 3.2 I recorded the average score of each configuration of rule-based agents in two-player games. This will serve as a baseline to understand how well a reinforcement learning agent is able to cooperate with each of the rule-based agents. The results show that while the agents which engage in intentional sending and receiving are able to play relatively well with each other, when an agent which is intentional receiving is paired with an agent which is not intentional sending, they completely fail to coordinate. This is to be expected, because an intentional receiving agent is expecting a certain communication protocol, and will incorrectly interpret the hints of the other player as communicating implicit information which was not, in fact, communicated. For this reason, IntSuperSafe does relatively well against all agents, because while it is intentional sending, it is not intentional receiving.

3.3.2 Self-play

The results of RainbowZero and RainbowToM training at self-play are shown in Figure 3.13. Both agents are able to learn a reasonably competent joint policy to play

	MaxSafe	MaxRisk	RandSafe	RandRisk	IntMaxSafe	IntMaxRisk	IntRandSafe	IntRandRisk	IntSuperSafe
MaxSafe	**10.07**	**10.2**	**6.6**	**7.16**	**1.69**	**0.84**	**1.82**	**1.07**	**9.89**
	(2.02)	(3.19)	(2.16)	(2.75)	(4.03)	(2.97)	(3.42)	(2.74)	(1.87)
MaxRisk	**9.93**	**8.64**	**7.65**	**6.99**	**1.54**	**0.65**	**1.16**	**0.72**	**9.51**
	(3.71)	(5.1)	(2.85)	(4.04)	(4.07)	(2.77)	(3.03)	(2.4)	(2.98)
RandSafe	**6.79**	**7.6**	**4.77**	**5.42**	**4.09**	**3.15**	**3.26**	**2.9**	**7.0**
	(2.15)	(2.73)	(1.74)	(2.17)	(4.44)	(4.5)	(3.27)	(3.32)	(2.11)
RandRisk	**7.28**	**7.04**	**5.53**	**5.43**	**2.75**	**2.02**	**2.26**	**1.86**	**6.88**
	(2.81)	(4.11)	(2.12)	(3.13)	(4.47)	(4.08)	(3.25)	(3.27)	(2.55)
IntMaxSafe	**1.95**	**1.5**	**4.05**	**2.68**	**14.18**	**13.67**	**13.09**	**12.99**	**11.27**
	(4.22)	(3.96)	(4.46)	(4.36)	(2.2)	(4.16)	(2.14)	(3.8)	(2.12)
IntMaxRisk	**0.85**	**0.92**	**3.18**	**2.23**	**13.82**	**11.68**	**12.93**	**11.35**	**11.51**
	(2.94)	(3.18)	(4.41)	(4.22)	(3.94)	(5.08)	(3.17)	(4.62)	(3.32)
IntRandSafe	**1.97**	**1.42**	**3.56**	**2.34**	**13.12**	**12.73**	**12.22**	**11.89**	**8.18**
	(3.44)	(3.2)	(3.26)	(3.22)	(2.17)	(3.26)	(2.08)	(3.14)	(2.54)
IntRandRisk	**1.32**	**0.89**	**3.54**	**2.2**	**12.78**	**11.37**	**11.89**	**10.86**	**8.6**
	(2.99)	(2.73)	(3.63)	(3.33)	(3.87)	(4.44)	(3.17)	(4.01)	(3.05)
IntSuperSafe	**9.82**	**9.48**	**6.66**	**6.99**	**11.34**	**11.68**	**8.5**	**8.8**	**9.51**
	(1.88)	(2.81)	(2.23)	(2.62)	(2.16)	(3.35)	(2.46)	(3.18)	(1.77)

TABLE 3.2: The capability of various rule based agents to play with one another. Average (standard deviation) number of points earned by pairs of agents is recorded after they play 1000 games. Rows represent the agent who starts the game.

the game against copies of themselves. It seems that RainbowZero is better able to do this than RainbowToM, which suggests that the RainbowToM architecture is not well suited to self-play. RainbowToM also learns to predict its partner (in this case, a copy of itself). The prediction accuracy rises at first, but then decreases as training continues. This is likely due, not to the degredation of RainbowToM's prediction ability, but rather that the strategy being employed by the partner (RainbowToM) is becoming more and more sophisticated, and thus harder to predict.

3.3.3 Ad-hoc Experiments

The results of training both RainbowZero and RainbowToM in the ad-hoc regime are shown in Figure 3.14. While RainbowZero and RainbowToM perform similarly at first, eventually RainbowZero appears to be able to achieve slightly higher average scores. I stopped the training at 60 million iterations. While the average score for both RainbowZero and RainbowToM grows steadily, the prediction accuracy of RainbowToM is not as simple. First it grows to a maximum at around 20 million iterations, but then it enters into a cycle of rising and falling accuracy.

From Table 3.2 I can estimate a lower bound on optimal ad-hoc play. If an agent had the policy to adopt the strategy of the best performing agent for each of the possible partners, then such an agent would get an average score of 11.16. Indeed, both RainbowZero and RainbowToM are able to outperform this baseline, as shown in Figure 3.14.

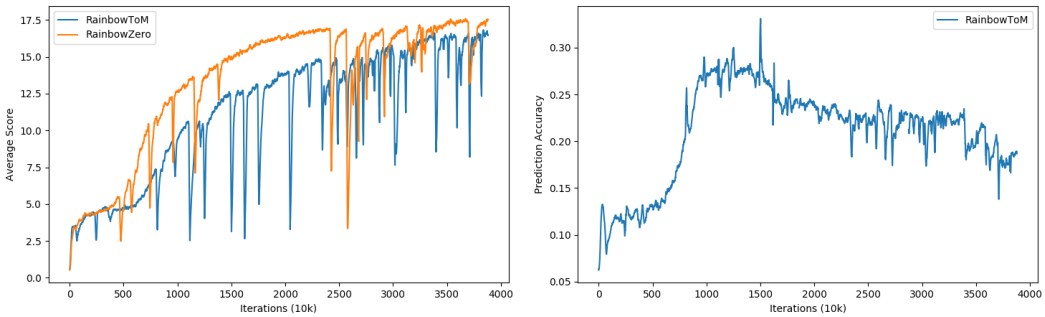


FIGURE 3.13: RainbowToM and RainbowZero learning to play against copies of themselves. On the left is shown the average score and on the right is shown the average prediction accuracy. RainbowToM is also tasked with predicting what its partner will do in the next turn, while RainbowZero is not.

While I had expected the RainbowToM architecture to provide significant advantages, both architectures appear to perform about equally. RainbowToM does manage to predict the rule-based agents far better than chance (chance would be 0.05 for 20 possible actions) which indicates that the prediction head is functioning as intended. This suggests that the failure of RainbowToM to outperform RainbowZero is a true negative result for that particular architecture in this environment. Future work could test variations on the architecture in different environment, perhaps with a wider variety in rule-based agents.

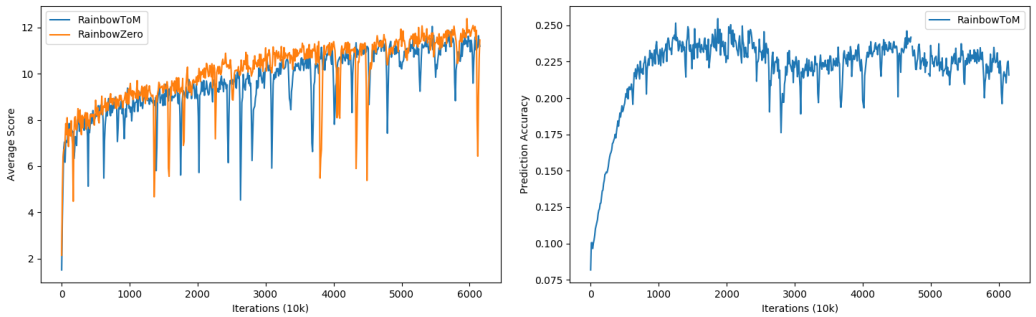


FIGURE 3.14: RainbowToM and RainbowZero learning to play against randomly chosen rule-based agents. On the left is shown the average score and on the right is shown the average prediction accuracy. RainbowToM is also tasked with predicting what its partner will do in the next turn, while RainbowZero is not.

There is a question how "intentional" the models are that RainbowToM makes of its partners. While it must build a model which predicts the behavior of its partner, it may not satisfy my strict interpretation of the "intentional stance", which requires the model represent the beliefs and decision-making process of the partner agent. In an effort to get some glimpse into the internal model being constructed by RainbowToM, I also did some analysis on the character embeddings produced by RainbowToM during play, to see if the network really was encoding something useful about the rule-based agents it was partnered with. I produced a dataset of

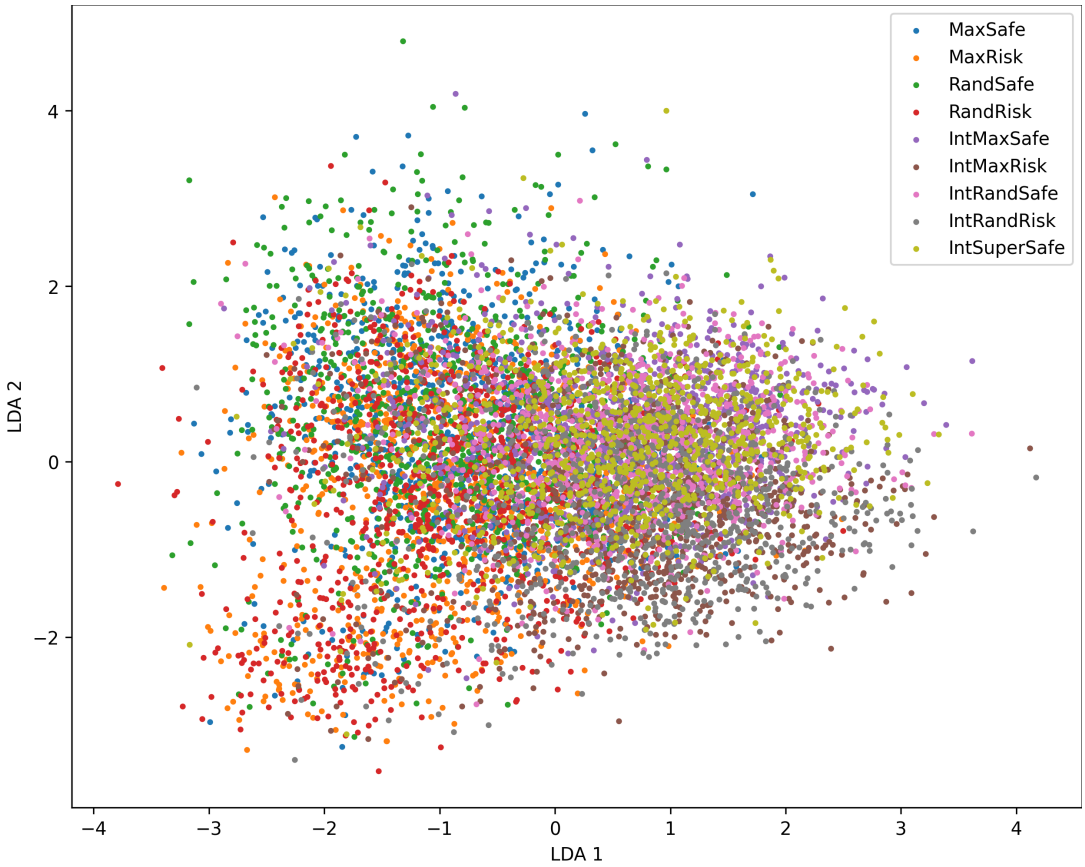


FIGURE 3.15: 9000 character embeddings (1000 for each rule-based agent). Each embedding was formed by RainbowToM by playing 8 consecutive games against a particular partner. The character embeddings are eight dimensional vectors, and the two most informative dimensions were extracted by LDA.

9000 character embeddings by having RainbowToM play 1000 sets of eight[7] games against each of the nine rule-based agents, and recording the character embedding formed at the end of the eighth game. I then performed Linear Discriminant Analysis (LDA) to reduce the dimensions from eight to two, and plotted the embeddings in Figure 3.15.

Because there is significant overlap, I also visualized the summary statistics for each distribution in Figure 3.16. As you can see, there is visible structure, and four distinct groups which are weakly distinguishable along the two dimensions.

I also checked to see if other dimensions extracted by LDA might show some structure as well. Figure 3.17 shows the characted embeddings plotted along the 3rd and 4th dimensions extracted by LDA. As you can see, the distributions are completely indistinguishable.

By analyzing the character embeddings, shown in Figures 3.15 and 3.16, I was able to determine that RainbowToM's representation contains two significant dimensions, representing true features of the agent types:

1. Intentional Hinting: Distinguishes **Int** agents from non-**Int** agents

[7] In training, RainbowToM plays sets of ten games. Because the final embedding is not trained to be useful (except as a generalization of being trained to be useful for previous games), I chose to record the embeddings at the end of the eighth game.

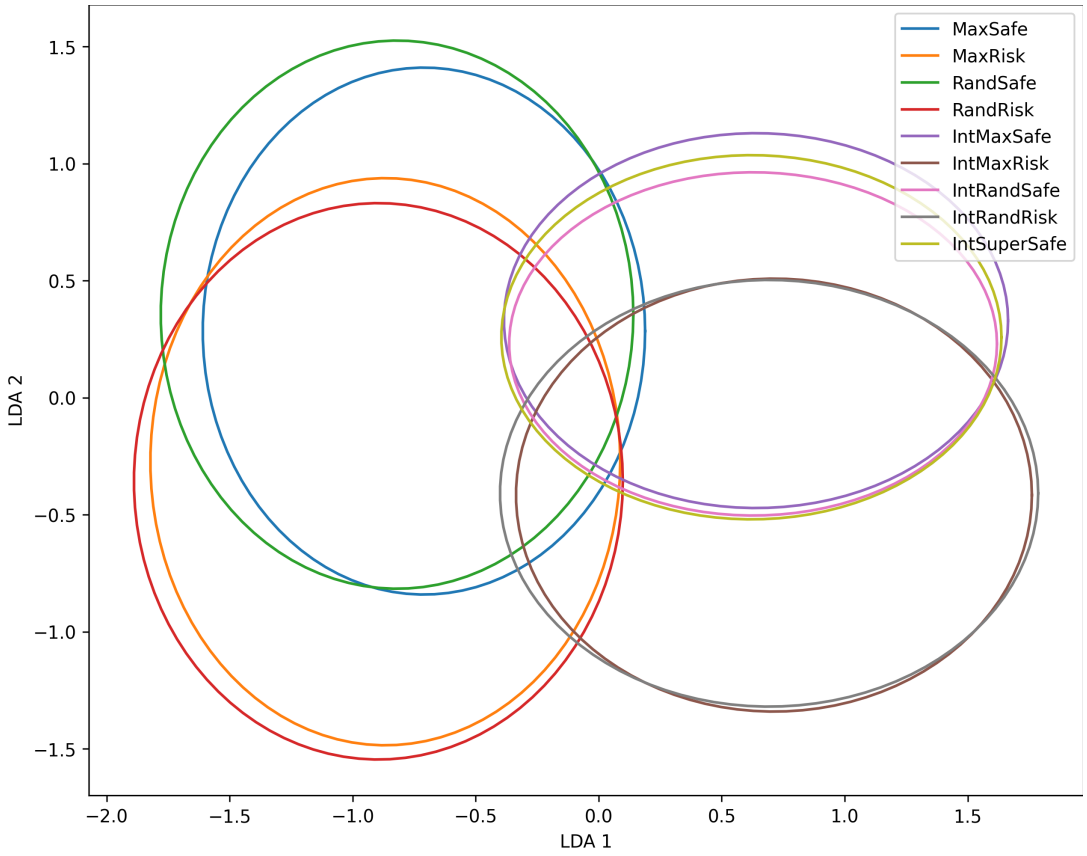
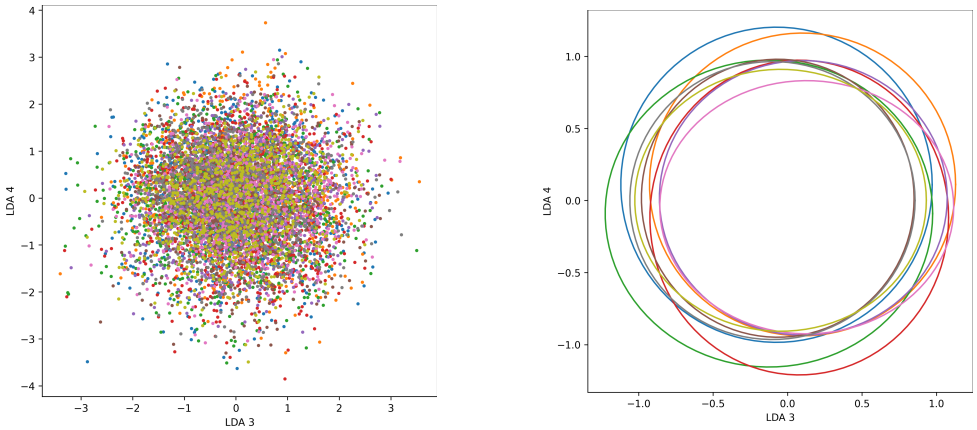


FIGURE 3.16: Representation of Figure 3.15 to visualize the mean and standard deviations of each of the 9 distributions. The radii of each ellipse represent a single standard deviation in each of the two dimensions, and the center of each ellipse the mean.



(A) Visualization of the character embeddings using the 3rd and 4th dimensions extracted by LDA.

(B) Visualization of the summary statistics of the character embedding distributions using the 3rd and 4th dimensions extracted by LDA.

FIGURE 3.17

2. Safe play or risky play: Distinguishes **Safe** agents from **Risk** agents

These features were discovered by RainbowToM by interacting repeatedly with all nine of the agent types, and were apparently the most useful for predicting the agents' behavior.

Chapter 4

Discussion and Conclusion

In this chapter I will give a high level overview of the results, what general conclusions they point to, and what further research directions would be fruitful. I will also discuss the limitations of the work done in this paper.

4.1 General Findings and Future Work

ToM does not always improve the ability of agents to coordinate. In the agent-based framework, while certain pairs of ToM agents seem to outperform the zeroth-order baseline, others significantly underperform it. In the RL framework, RainbowZero outperforms RainbowToM in self-play, and in ad-hoc play they achieve about the same, suggesting no advantage to ToM reasoning here.

In the agent-based framework, it seems that ToM is most useful when the default zeroth-order policy does not produce a very high win rate, as seen in Table 2.5. For larger models produced by MCTS, the zeroth-order policies are much better, and the benefit of ToM seems to mostly disappear. This would seem to connect well to the results found by De Weerd, Verbrugge, and Verheij (2022) which show that higher-order theory of mind is most advantageous in more complex environments.

These findings in the symbolic case have not been verified in the RL framework. It would be valuable future research to test RainbowToM and RainbowZero in ad-hoc environments which include a much wider range of agents, and thus offer a more significant challenge to building a competent heuristic strategy. The results from the agent-based study suggest that in scenarios where RainbowZero really struggles to play well it might be that RainbowToM can offer some advantage. It could be that the ad-hoc environment used here is not challenging enough, and therefore the results of the RL study are more analogous to the agent-based study performed with the largest model, where ToM provides very little advantage.

Furthermore, the RL framework only tests scenarios with zeroth-order partners. In the agent-based framework, ToM agents generally perform best when paired with other ToM agents. This could be similarly tested in the RL framework by running experiments in ad-hoc regimes with agents that use some form of ToM.

4.1.1 Interpretable AI

The results of probing the representations formed in the character embedding space were quite limited, and I did not come up with or implement any method for analyzing the mental embeddings formed by RainbowToM. Future work could undertake this to study more qualitatively what kind of internal models are produced by neural networks of this kind. Interpreting the latent representations of neural networks is still an open problem, and so any such work would likely yield novel insights.

The quality of future work exploring the application of ToM in neural networks will always be limited by the ability to verify how ToM is exactly implemented internally. It would be very important to build a better understanding of what kind of internal computation is happening within neural networks trained in multi-agent environments.

4.1.2 Limitations

A major limitation of the agent-based experiments is the particular way in which I implemented ToM, which is itself highly simplified. Firstly, ToM only flows in one direction, that is, agents use ToM in order to select actions, but they don't use ToM to update their world model. Secondly, there isn't the ability for agents with higher-order ToM to use lower-order ToM when it is more useful, by maintaining a belief over the reasoning order of their partner. An example of this is De Weerd, Verbrugge, and Verheij (2015) which mixes different orders of ToM by means of a dynamically updated term representing an agent's confidence in the order of their partner.

The method used in the cooperative RL framework, while it does in theory allow an agent to use ToM to both select actions and update their world model, due to the inability as of yet to interpret the function being implemented by the neural network, it is difficult to verify that this is actually taking place.

I only tested a single training regime and architecture, and because deep learning is highly sensitive to the initial conditions it could be that other training regimes and architectures lead to different results. Furthermore, due to limitations on compute, I only performed a single run for each agent, which means that my results have the risk of not being very repeatable.

4.1.3 Concluding Remarks

There are two major findings from the results. First, higher-order ToM is more useful when the problem is too challenging to be solved with zeroth-order ToM alone, otherwise zeroth-order ToM tends to outperform. Second, ToM is most useful when the environment contains other agents also engaging in ToM.

A lot of further research is needed to back up these findings, especially in the cooperative RL case where the results are especially weak. In particular I expect neural network interpretability directions to be especially fruitful, given that I only probed the neural network representations a small amount and already found meaningful structure.

Bibliography

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton (2016). "Layer Normalization". In: *Stat* 1050, p. 21.

Bard, Nolan, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. (2020). "The Hanabi challenge: A new frontier for AI research". In: *Artificial Intelligence* 280, p. 103216.

Baron-Cohen, Simon (2001). "Theory of mind in normal development and autism". In: *Prisme* 34.1, pp. 74–183.

Bellemare, Marc G, Will Dabney, and Rémi Munos (2017). "A distributional perspective on reinforcement learning". In: *International Conference on Machine Learning*. PMLR, pp. 449–458.

Blokpoel, Mark, Marlieke van Kesteren, Arjen Stolk, Pim Haselager, Ivan Toni, and Iris Van Rooij (2012). "Recipient design in human communication: simple heuristics or perspective taking?" In: *Frontiers in Human Neuroscience* 6, p. 253.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). "OpenAI Gym". In: *arXiv e-prints*, arXiv–1606.

Canaan, Rodrigo, Xianbo Gao, Youjin Chung, Julian Togelius, Andy Nealen, and Stefan Menzel (2020). "Evaluating RL agents in Hanabi with unseen partners". In: *AAAI'20 Reinforcement Learning in Games Workshop*.

Castro, Pablo Samuel, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G. Bellemare (2018). "Dopamine: A Research Framework for Deep Reinforcement Learning". In: URL: http://arxiv.org/abs/1812.06110.

Chomsky, Noam (1959). "Review of skinner's Verbal Behaviour". In: *Language* 35, pp. 26–58.

Cox, Christopher, Jessica De Silva, Philip Deorsey, Franklin HJ Kenter, Troy Retter, and Josh Tobin (2015). "How to make the perfect fireworks display: Two strategies for Hanabi". In: *Mathematics Magazine* 88.5, pp. 323–336.

De Laplace, Marquis (1794). *A philosophical essay on probabilities*. Courier Corporation.

De Weerd, Harmen, Rineke Verbrugge, and Bart Verheij (2013). "How much does it help to know what she knows you know? An agent-based simulation study". In: *Artificial Intelligence* 199, pp. 67–92.

De Weerd, Harmen, Rineke Verbrugge, and Bart Verheij (2015). "Higher-order theory of mind in the tacit communication game". In: *Biologically Inspired Cognitive Architectures* 11, pp. 10–21.

De Weerd, Harmen, Rineke Verbrugge, and Bart Verheij (2017). "Negotiating with other minds: the role of recursive theory of mind in negotiation with incomplete information". In: *Autonomous Agents and Multi-Agent Systems* 31.2, pp. 250–287.

De Weerd, Harmen, Rineke Verbrugge, and Bart Verheij (2022). "Higher-order theory of mind is especially useful in unpredictable negotiations". In: *Autonomous Agents and Multi-Agent Systems* 36.2, pp. 1–33.

Dennett, Daniel C (1981). *Brainstorms: Philosophical essays on mind and psychology*. MIT Press.

Filan, Daniel, Stephen Casper, Shlomi Hod, Cody Wild, Andrew Critch, and Stuart Russell (2021). "Clusterability in Neural Networks". In: *arXiv e-prints*, arXiv–2103.

Flint, Alex (2020). *The ground of optimization*. URL: https://www.alignmentforum.org/posts/znfkdCoHMANwqc2WE/the-ground-of-optimization-1.

Fuchs, Andrew, Michael Walton, Theresa Chadwick, and Doug Lange (2021). "Theory of Mind for Deep Reinforcement Learning in Hanabi". In: *arXiv e-prints*, arXiv–2101.

H-group conventions (2022). URL: https://hanabi.github.io/.

Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver (2018). "Rainbow: Combining improvements in deep reinforcement learning". In: *Thirty-second AAAI Conference on Artificial Intelligence*, pp. 3215–3222.

Kinderman, Peter, Robin Dunbar, and Richard P Bentall (1998). "Theory-of-mind deficits and causal attributions". In: *British Journal of Psychology* 89.2, pp. 191–204.

Kocsis, Levente and Csaba Szepesvári (2006). "Bandit based Monte-Carlo planning". In: *European Conference on Machine Learning*. Springer, pp. 282–293.

Kocsis, Levente, Csaba Szepesvári, and Jan Willemson (2006). "Improved Monte-Carlo search". In: *Univ. Tartu, Estonia, Tech. Rep* 1.

Kullback, Solomon and Richard A Leibler (1951). "On information and sufficiency". In: *The annals of mathematical statistics* 22.1, pp. 79–86.

Liddle, Bethany and Daniel Nettle (2006). "Higher-order theory of mind and social competence in school-age children". In: *Journal of Cultural and Evolutionary Psychology* 4.3-4, pp. 231–244.

Meyer, John-Jules Ch and Wiebe Van Der Hoek (2004). *Epistemic logic for AI and computer science*. 41. Cambridge University Press.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, pp. 529–533.

Nair, Vinod and Geoffrey E Hinton (2010). "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th International Conference on Machine Learning*, pp. 807–814.

Nguyen, Anh, Jason Yosinski, and Jeff Clune (2016). "Multifaceted feature visualization: uncovering the different types of features learned by each neuron in deep neural networks". In: *arXiv e-prints*, arXiv–1602.

Nguyen, Dung, Phuoc Nguyen, Hung Le, Kien Do, Svetha Venkatesh, and Truyen Tran (2022). "Learning theory of mind via dynamic traits attribution". In: *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 954–962.

Olah, Chris, Alexander Mordvintsev, and Ludwig Schubert (2017). "Feature visualization". In: *Distill* 2.11, e7.

Rabinowitz, Neil, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick (2018). "Machine theory of mind". In: *International Conference on Machine Learning*. PMLR, pp. 4218–4227.

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2016). "Prioritized Experience Replay". In: *International Conference on Learning Representations*.

Schrödinger, Erwin (2012). *What is Life?: With Mind and Matter and Autobiographical Sketches*. Cambridge University Press.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. (2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *arXiv e-prints*, arXiv–1712.

Sutton, Richard S (1988). "Learning to predict by the methods of temporal differences". In: *Machine Learning* 3.1, pp. 9–44.

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep reinforcement learning with double Q-learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1, pp. 2094–2100.

Walton-Rivers, Joseph, Piers R Williams, and Richard Bartle (2019). "The 2018 Hanabi competition". In: *2019 IEEE Conference on Games (CoG)*. IEEE, pp. 1–8.

Watkins, Christopher JCH and Peter Dayan (1992). "Q-learning". In: *Machine Learning* 8.3, pp. 279–292.

Weiss, Izaak (2016). *Double Pendulum Chaos Demonstration*. URL: https://www.youtube.com/watch?v=pEjZd-AvPco.

Wellman, Henry M, David Cross, and Julanne Watson (2001). "Meta-analysis of theory-of-mind development: The truth about false belief". In: *Child Development* 72.3, pp. 655–684.

Wimmer, Heinz and Josef Perner (1983). "Beliefs about beliefs: Representation and constraining function of wrong beliefs in young children's understanding of deception". In: *Cognition* 13.1, pp. 103–128.

Yudkowsky, Eliezer (2008). *Optimization*. URL: https://www.lesswrong.com/posts/D7EcMhL26zFNbJ3ED/optimization.

Zhang, Quanshi, Ying Nian Wu, and Song-Chun Zhu (2018). "Interpretable convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8827–8836.