Advancing AI: Hanabi Challenge

Jungkyu (JP) Park
Center for Data Science
New York University

Abstract

I combine Self-Other Model (Raileanu et al., 2018) with the rainbow agent (Hessel et al., 2017) with some modifications in order to tackle the Hanabi challenge (Bard et al., 2019). I observe that one of my implementations outperforms Deepmind's reported score with a rainbow agent for 2-player Hanabi.

1 Introduction

Hanabi can be understood as cooperative solitaire, where the common goal is to complete five decks of cards of different colors from number 1-5 in increasing order. A total of 50 cards are used in a game: three 1's, two 2's, two 3's, two 4's, and one 5 each from 5 different colors. Players start by having 5 cards in their hand, and draw a new card from unused deck every time they either play or discard a card. The game ends if the 5 decks are completed, if all cards are drawn from the unused deck, or if the team makes three incorrect moves in total. Score is calculated as 1 per card in the assembled deck: 25 is the perfect score. A random agent will score under 5 due to incorrect moves.

One can see everyone else's hands except for one's own, which makes this game interesting and challenging. One must convey information about the other player's cards via hints; by using one hint token, one player can choose either a specific color or specific number and let another player know all of his cards which has that color or number. The number of hint tokens are limited to 8, and one can recover a hint token if one succeeds at discarding an unnecessary card. One can make an incorrect move by trying to play a card that cannot be played or discarding a card that's necessary to complete the game (discarding a 5, for example).

A number of handwritten algorithms for solving Hanabi exists (Eger et al., 2017; Walton-Rivers et al., 2017). Since the number of hints are limited, the approaches which maximizes the amount of information conveyed per move such as Cox et al. (2015) and Bouzy (2017) seem to perform the best. However, one of the state-of-the-art reinforcement learning algorithms (Hessel et al., 2017) fails to perform better than those hand-written algorithms, which is why the Hanabi challenge is created (Bard et al., 2019). There exists a reinforcement-learning method that succeeds at achieving decent score at this (Foerster et al., 2018), but it is only state-of-the-art in two-player Hanabi.

I implement some combinations of 5 different approaches that involve theory of mind and one of my implementations get max score of 21.29 in 2-player Hanabi, outperforming Deepmind's rainbow agent by 0.65.

2 Motivation: Human conventions of Hanabi

Looking at conventions developed by human players of Hanabi such as Zamiell (`https://github.com/Zamiell/hanabi-conventions`), it seems important for players to agree upon some conventions in order to figure out which card is being hinted at despite the ambiguity of multiple cards being hinted at once. With help of conventions such as good touch principle (hint only the cards that will eventually be played), save principle (save all 5's), chop, finesse and prompt, each player can convey more specific information via hint such as playing, discarding, or save a particular card.

I attempt to autonomously learn these conventions by (1) using the same model for all agents in the game (then everyone will always act in the same way), (2) modifying the action space so that they output which specific card to give hints to rather than what type of hint to give to (so that the agent has some explicit intention behind the hints), and (3) using Self-Other model to infer the intention behind hints of the agents as well as estimates for the current agent's cards based on other agents' actions in the previous round. This estimates will be extra input parameters to the model of each agent.

Also keeping track of which unique cards have been hinted so far could be useful in learning the good touch principle.

3 Methods

My starting point is the repository from the Hanabi challenge (`https://github.com/deepmind/hanabi-learning-environment`). I edited hanabi environment written in cpp and the model code written in python TensorFlow, which can be found in the following repository: `https://github.com/jpatrickpark/hanabi-learning-environment`. I tried some combinations of the following 5 modifications. For this paper, I only experiment with 2-player game of hanabi for simplicity.

3.1 input_SoM: Self-Other Model to guess my unknown cards

I apply Self-Other Model (Raileanu et al., 2018) to the rainbow agent in order to estimate one's own hand and feed it as input. Each agent reconstruct input bit for other agents using only the knowledge available to them, make prediction, calculate loss with observation, backpropagate to input, and choose the cards that are the most likely.

In order to implement this, I introduce extra 125 input bits for the estimates of my own cards. At the beginning of the game, the very first agent who has no observation of any actions uses all 0's for this input. I will refer to the current agent of the turn A, and the previous agent B. When A pretends to be B,

1. A puts A's perfect knowledge of B's card in place of B's estimates for B's card.
2. Unlike B, A doesn't have perfect knowledge of A's own cards. Instead, A puts his current estimates of A's cards in place of B's knowledge of A's cards. This estimates are constructed as follows:
   (a) Get the representation of A's cards from the history of hints, which is visible to all agents. A card is represented using 25 bits, 1 for each of 25 possibilities. All remaining possibilities are set to 1. For example, $0000011111000000000011111$ is a representation for either yellow or blue card of any rank.
   (b) Divide each card representation by its sum, turning the 0-1 representation into probabilities. For example, change $0000011111000000000011111$ into list of floating point values, $[0, 0, 0, 0, 0, 0.1, 0.1, 0.1, 0.1, 0.1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.1, 0.1, 0.1, 0.1, 0.1,]$.
   (c) This reconstructed input gets fed to the model, cross-entropy loss with A's observation of B's previous action is calculated, backpropagation is performed all the way to the input, and card representation gets updated. A then chooses one most likely possibility and creates a one-hot vector for it. A does this for all 5 cards.

As a result, A receives updated estimates of A's own hands generated by its model weights. A then uses this for its own input in order to decide its action inside the game. Number of backpropagation steps to take in estimation is a hyperparameter, I use value of 1 because it is almost always enough to break ties between the possibilities.

3.2 Action space modification

Previously, unmodified rainbow agents choose to give a hint to one of the other players about either 5 color hints or 5 rank hints. Here, I redefine the action space to be about which specific card to give a hint to using either its color or its rank. In both cases, the size of the action space remains 10+10*(number of agents in game - 1). First ten bits refer to play or discard action for the 5 card in

one's own hand. For unmodified agents, the rest refer to hint actions (the 5 color or 5 rank) per other agents. For modified agents, the rest refer to 2 bits per card per other player.

3.3 output_SoM: Guess intentions behind hints via Self-Other Model

This method can only be used when section 3.2 is applied. I use self-other model to guess what the intention behind hints were by pretending to be another agent who gave me a hint in the previous round and choosing the output with the biggest value in softmax layer among possible ones. Concretely, I introduce 5 extra input bits for indicating which one of one's own cards have been hinted, and set it to one-hot vector when an agent received a hint in the previous round, and otherwise zero. When player size is greater than 2, I can introduce $(\text{hand size}) * ((\text{number of agents}) - 1)$ bits.

3.4 Extra input bits for Good Touch Principle

Zamiell's group invented a principle where they only 'touch' cards that would eventually be played. They define that a card is 'touched' when a hint directly reveals rank or color of that card. It could be difficult to keep track of this information which card identities should not be hinted anymore with current way of input representation. Therefore, I include an additional input bits of size 25 for the 25 card identities and set it to 1 if this card had been either played or hinted in the current hands. This must be done with care here since an agent doesn't know the card identities of one's own hinted cards unless he received enough information for them, and I set these bits using the information available to each agent only. This does not hurt autonomy because we are in no way forcing the model to use this information.

3.5 Decrease replay buffer size

I experimented on reducing the amount of experience replay buffer since the play depends greatly on "conventions" and the agents won't be able to learn much from experiences when they had different model weights. I decreased max replay buffer size from 50000 to 1000 steps.

4 Results

Results with default max replay buffer We stop training of all models at iteration 10,000 (game 100,000,000) and compare the best evaluation performance (with no exploration). The result is shown in figure 1. The unmodified model (red) is outperformed by GTP, input_SoM, and GTP+input_SoM throughout almost entire training period, while input_SoM is performing the best.

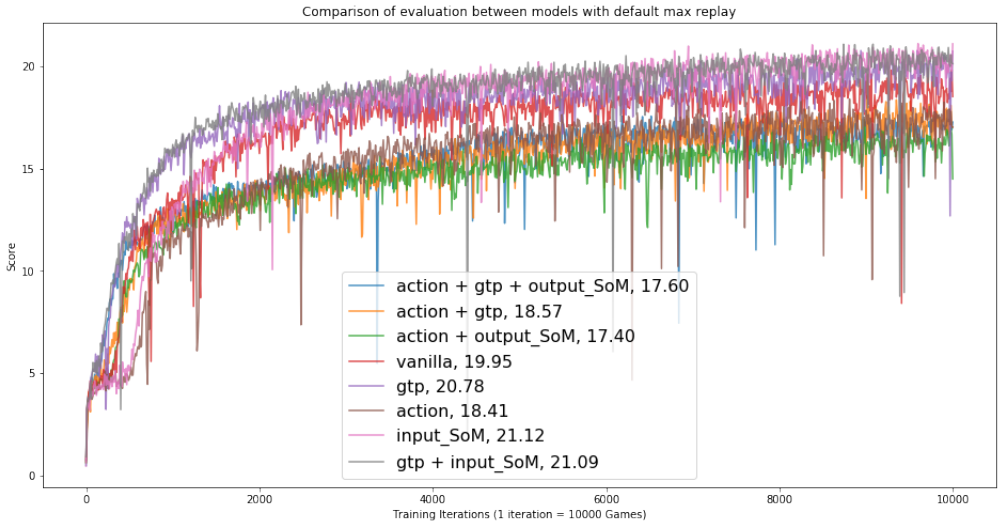

Figure 1: Default max replay buffer (50000 steps).

Results with small max replay buffer Figure 2 is for the models with smaller replay buffer. Here input_SoM still outperforms the vanilla model. My best model (21.29) not only outperforms model with bigger replay buffer (21.12) but also deepmind's reported performance of Rainbow agent on 2-player Hanabi (20.64).

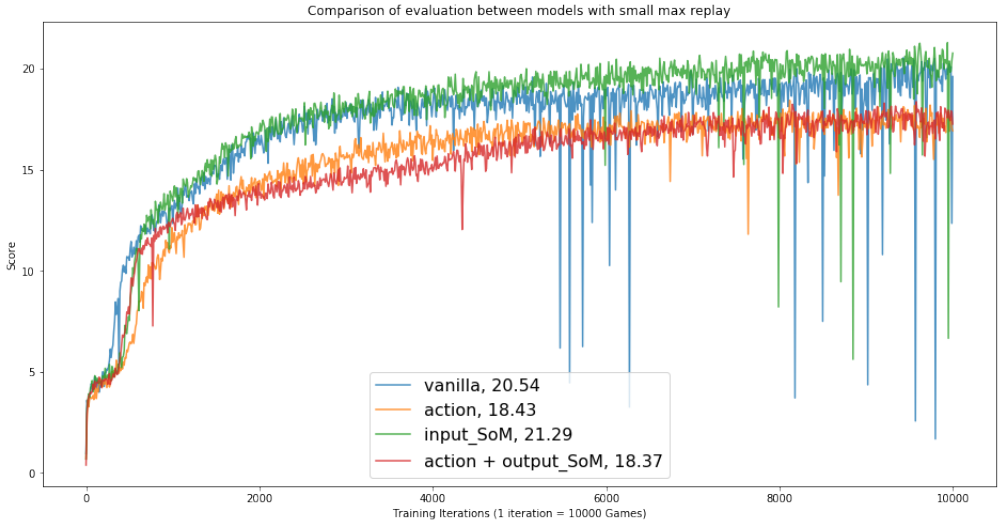

Figure 2: Small max replay buffer (1000 steps).

5 Discussion and Future Work

input_SoM, despite the current simple implementation, is performing better than unmodified Rainbow agent. We could experiment further with hyperparameter-tuning for how many times to perform gradient descent in estimating the other agent in order to improve the result further.

Replay buffer size is also another hyperparameter that is worth tuning, since reducing it slightly outperforms models with default replay buffer size.

Anything that involves modification of action space is performing poorly, so it could be the design of action space modification that is problematic rather than output_SoM itself.

Extra input bits for good touch principle (20.78) does outperform the unmodified baseline (19.95), but good touch principle with input_SoM does not outperform input_SoM only. However, using it helps increasing performances at the early stages of training in both cases ($<$2,000 iterations).

It should be easy to extend output_SoM to games with more than 3 player Hanabi. We can just add more input bits for each of other players for estimates of their intentions behind their hint toward my own cards, and run SoM separately for each agent. However, it is doubtful if it will perform well with more players since it did not perform well on 2-player Hanabi.

On the other hand, extending input_SoM to more than 3 player game should be done with care. To get multiple estimates of my own hand and feed them all as my own input by concatenation can be dangerous since the agent might not know how to deal with possibly conflicting information about its own hand. Combining the estimates using either 'or gate', average, or majority vote should be a safer approach which I would like to try next.

Also I can try representing each card using two one-hot vectors of length 5 rather than a single one-hot vector of length 25. This way, I can get estimates of color and rank separately, which makes sense if the other agent's action is only relevant to the rank of some of my cards.

Final direction is to apply some of the methods that worked to A3C (Mnih et al., 2016). Deepmind already showed that actor-critic approach works better than Rainbow. An advantage of A3C to note is that it does not require a replay buffer which I found out to be harmful.

References

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for AI research. *CoRR*, abs/1902.00506, 2019. URL http://arxiv.org/abs/1902.00506.

Bruno Bouzy. Playing hanabi near-optimally. In Mark H.M. Winands, H. Jaap van den Herik, and Walter A. Kosters, editors, *Advances in Computer Games*, pages 51–62, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71649-7.

Christopher Cox, Jessica De Silva, Philip Deorsey, Franklin H. J. Kenter, Troy Retter, and Josh Tobin. How to make the perfect fireworks display: Two strategies for hanabi. *Mathematics Magazine*, 88 (5):323–336, 2015. doi: 10.4169/math.mag.88.5.323. URL https://doi.org/10.4169/math.mag.88.5.323.

M. Eger, C. Martens, and M. A. Cordoba. An intentional ai for hanabi. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 68–75, Aug 2017. doi: 10.1109/CIG.2017.8080417.

Jakob N. Foerster, Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian action decoder for deep multi-agent reinforcement learning. *CoRR*, abs/1811.01458, 2018. URL http://arxiv.org/abs/1811.01458.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL http://arxiv.org/abs/1710.02298.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling others using oneself in multi-agent reinforcement learning. *CoRR*, abs/1802.09640, 2018. URL http://arxiv.org/abs/1802.09640.

J. Walton-Rivers, P. R. Williams, R. Bartle, D. Perez-Liebana, and S. M. Lucas. Evaluating and modelling hanabi-playing agents. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1382–1389, June 2017. doi: 10.1109/CEC.2017.7969465.