Opponent Hand Estimation in Gin Rummy Using Deep Neural Networks and Heuristic Strategies

Bhaskar Mishra, Ashish Aggarwal
University of Florida, USA
{bmishra1, ashishjuit}@ufl.edu

Abstract

A vital part of any good strategy for most imperfect-information games is making predictions about the information that is unavailable. For example, in card games like Poker and Gin Rummy, predicting the kinds of cards the opponent is holding is necessary for playing well. Specifically, it is useful for agents to be able to map the partial game states that are made available to them to the probabilities of each of the possible complete game states, given that they are playing against other rational player(s). Finding this relationship, however, is difficult, as it requires knowledge of how a rational player would play, which is the problem this relationship is being used to solve. In this paper, we attempt to find this relationship in the context of the card game Gin Rummy, though instead of predicting the complete game state, we focus on what is most useful to a player: the opponent's hand. We do this by using heuristic utility functions to create an agent that approximates how a rational player would play, and then using the resulting game data to train a Deep Neural Network mapping known information to predictions about the opponent's hand. This model is used to improve the existing agent and, in turn, produce more data to create better models.

Introduction

Imperfect-information games model decision-making situations involving two or more agents, where agents do not have access to complete information. Considerable work has gone into finding solutions for such games. Techniques like counterfactual regret minimization (CFR) have been used to find Nash equilibrium strategies for players (Zinkevich et al. 2008), and various kinds of abstractions have been used to find approximate solutions to increasingly complex games (Sandholm 2015). In 2016, *DeepStack* achieved super-human performance in the game of heads-up no-limit Texas hold'em, using deep neural networks to approximate the value of game states and efficiently find approximate Nash equilibria without using the entire game tree (Moravčík et al. 2017). At around the same time, *Libratus* achieved super-human performance in the same game, using an approach called "nested solving" where the agent repeatedly solves an increasingly detailed strategy of the portion of the game that is relevant in real time (Brown and Sandholm 2018).

As of now, the common benchmark that has been used to test these techniques is the game of Poker, specifically heads-up Texas hold'em (HTH). Though the specific variants of HTH have changed – older techniques used heads-up *limit* Texas hold'em, but after that was solved in 2015 (Bowling et al. 2015), newer techniques like *DeepStack* and *Libratus* used the much more complex game of heads-up *no-limit* Texas hold'em (Moravčík et al. 2017; Brown and Sandholm 2018) – the general structure of the game has not changed. Relative to a game like Gin Rummy, HTH has a very small number of unique starting hands, and even when including the 5 community cards, the number of unique card combinations is still less than the number of unique starting hands in Gin Rummy.

- HTH Starting Hands = $\binom{52}{2} = 1326$
- HTH Card Combinations = $\binom{52}{2} \times \binom{50}{5} \approx 2.8 \times 10^9$
- Gin Rummy Starting Hands = $\binom{52}{10} \approx 1.5 \times 10^{10}$

Additionally, in HTH, there are many card combinations that require very similar strategies and as a result can be bucketed together without significantly affecting the resulting strategy. This bucketing can reduce the number of information sets in the abstracted game by several orders of magnitude (Zinkevich et al. 2008). In Gin Rummy, the strategy seems very dependent on the specific set of cards a player has, and there does not seem to be room for such significant state abstraction.

The game tree of Gin Rummy also has much greater depth. Whereas each player only makes 4 decisions in each hand of HTH (at each of the 4 rounds, a player chooses either to fold, call, or bet one of the allowed betting amounts), in a single round of Gin Rummy, players typically make 2-3 decisions per turn – where to draw from, which card to discard, and whether or not to knock, if able – and there can theoretically be an infinite number of turns per round.

These differences make it so that the techniques used to solve variants of HTH can not be used in an obvious way to approach a game like Gin Rummy. There would either need to be modifications to existing techniques to fit this context, or new approaches altogether. In this paper, we do not attempt to "solve" the game of Gin Rummy, but aim to create an agent that can perform well despite how large the game tree of Gin Rummy is. In any good Gin Rummy

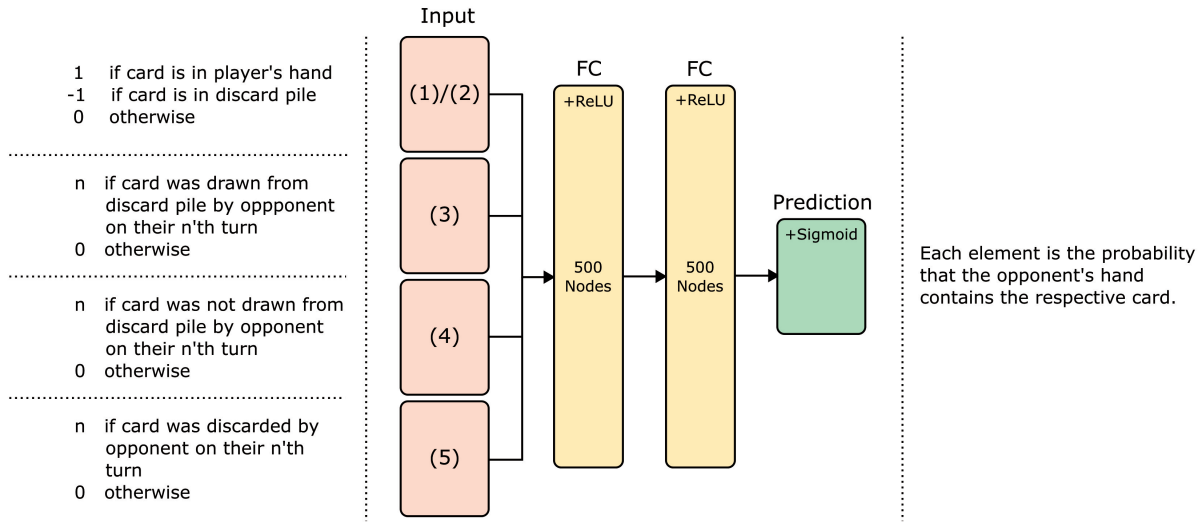

Figure 1: Model Design: A standard feed-forward neural network with two fully connected hidden layers using ReLU activation functions, and a fully connected output layer using a sigmoid activation function. The input consists of four 13×4 matrices, where each element in each matrix represents information regarding one of the 52 cards.

strategy, and in strategies for most imperfect-information games, it is important for agents to be able to make predictions about the information that is unavailable to them. In Gin Rummy, an agent needs to be able to understand which cards their opponent is likely to have, given that they are playing against a rational opponent. We aim to use a Deep Neural Network to model this relationship, and combine this model with heuristic-based strategies to create a strong Gin Rummy agent.

Model Design

Our goal here is to design a model that can take the complete information that is available to a player, and use it to determine the likelihood of each of the possible game states being the actual game state. For the sake of efficiency, the model we will be designing will be a simplified version of this idea. We will instead take only the information that we expect to be useful as input and focus on making predictions about just the opponent's hand, rather than the complete game state. The opponent's hand seems to be the unknown that a player would most benefit from knowing, as the player can then avoid discarding cards that are likely to benefit the opponent, and know the cards they are more likely to get when drawing from the deck.

For input, we will be gathering data from 5 sources:

1. Player's Hand
2. Discard Pile
3. Cards Opponent Drew from Discard Pile
4. Cards Opponent Chose not to Draw from Discard Pile
5. Cards Opponent Discarded

Suppose that the ranks and suits of cards are each subject to some strict ordering. The information in (1) and (2) can be stored in a 13×4 matrix where the element at indices i, j represents information regarding the card with the i'th rank and the j'th suit. Each element will be set to 1 if the respective card is in the player's hand, -1 if it is in the discard pile, and 0 otherwise. The information in (3) can similarly be stored in a 13×4 matrix with each element being set to 0 if the respective card hasn't been drawn from the discard pile by the opponent, and otherwise set to the turn in which the opponent drew the card (1 for their first turn, 2 for their second turn, and so on). The same approach can be used to create two more 13×4 matrices to store the information in (4) and (5) respectively.

For output, we will be producing another 13×4 matrix where each element contains the probability of the opponent's hand containing the respective card. Though we have chosen to output only individual probabilities to meet our time constraints, future iterations of this agent could also output relative probabilities between cards in order to better model the probability distribution of the opponent's hand. For now, we will act as if the probabilities are independent.

For our network, we will be using a standard feed-forward neural network. We will take the 208 nodes of input (four 13×4 matrices), pass them through two fully connected layers with 500 nodes each and ReLU activation functions, and then end with a fully connected layer with 52 nodes and a sigmoid activation function to produce the output. A diagram for the network architecture can be seen in Figure 1.

Heuristic Based Strategy

With a design for the neural network, the next problem is generating data to train the network on. The issue is that generating the data requires knowledge of how a rational player would play, which is the problem the data is being used to solve. To approach this problem, we start by creating approximate rational agents. We create a heuristic-based

strategy that can be tuned using various hyper-parameters to change the weight of various elements of the strategy. Specifically, we use the following eight hyper-parameters, each of which will be explained in following sections.

MELD_BONUS	COMBINATION_BONUS
DEADWOOD_BONUS	KNOCK_BONUS
GIN_BONUS	OPP_UTIL_IMPORTANCE
LOW_CARD_BONUS	EMERGENCY_BOOSTER

These hyper-parameters are tuned using a combination of random search and grid search to produce 2-3 well-performing strategies. The resulting agents are then put against each other to simulate several games, and at each turn of each game, the data from the current player and their opponent's hand and action history is used to create an entry in the dataset. Multiple unique agents are used to avoid overfitting and ensure the network is able to handle opponents utilizing strategies it hasn't seen before.

Heuristic Utility Function

The basis of our heuristic-based strategy is a heuristic utility function that estimates the value of a hand, with hands with higher values being preferable to hands with lower values. The utility function is the sum of 3 metrics: average card utility, the number of melded cards, and a knockability bonus.

The latter two are relatively straight forward. The number of melded cards is equal to the number of cards that are part of a meld in the hand, based on the melding that the agent chooses. How the agent chooses to meld will be discussed later. The knockability bonus gives points for a hand that can be knocked, giving KNOCK_BONUS if the hand can be knocked but can't be used to go gin, GIN_BONUS if the hand can be used to go gin, and 0 otherwise.

Average card utility is a more complex metric. It assigns a "card utility" to each unmelded card in the hand based on the card's "usefulness" in forming future melds and its point value, and then measures the average utility of those cards. The usefulness of a card is measured by looking at all 3-card melds that both contain the card and are possible – contain only cards that are neither in the discard pile, nor already melded in the player's hand. MELD_BONUS is added to card utility for each such meld, and COMBINATION_BONUS is added if the hand contains two of the cards in the meld. Points are also either added to or subtracted from card utility based on how low or high the point value of the card is. The pseudo-code for card utility is roughly as follows:

```
CardUtility(card, hand):
utility = 0
for each possible 3-card meld containing card:
    utility += MELD_BONUS
    if hand contains two of the cards in the meld:
        utility += COMBINATION_BONUS
utility += (5 - card point value) * DEADWOOD_BONUS
return utility
```

The actual card utility function used in our agents differs in a few ways. First, all points that are added as the result of a "possible meld" are scaled by the probability that none of the cards in the meld are part of a meld in the opponent's hand. This is because if it is the case that a card in the meld is part of a meld in the opponent's hand, it is unlikely that the card will be discarded, and thus, the meld is unlikely to be attained. Calculating this probability, however, requires a prediction about the opponent's hand. We get around this by initially using a prediction which assigns an equal probability to each card that isn't either in the player's hand or the discard pile, and 0 to all other cards. Once a model has been trained, its predictions can be used here instead. Secondly, we add LOW_CARD_BONUS and LOW_CARD_BONUS$/2$ to the utilities of cards with ranks A and 2, respectively, as they are cards that are particularly useful in creating a knock-able hand, and are typically undervalued because they have fewer possible melds. Lastly, when the player's hand is one card away from being knock-able, our agents enter an "emergency" state, where DEADWOOD_BONUS is scaled by EMERGENCY_BOOSTER. This is because when the player is very close to knocking, it becomes more useful to discard cards that contribute higher amounts to the total deadwood.

Melding

When making decisions, our agents frequently need to know how they would currently meld any cards in their hand. Usually, this decision is easy, as there is typically only one possible melding[1] that minimizes the hand's deadwood value. Sometimes, however, there are multiple possible meldings resulting in the minimal deadwood value. In these cases, we've set our agents to pick the melding that results in the hand with the highest average card utility.

Drawing Strategy

During the drawing phase of Gin Rummy, the player has to decide to draw from either the discard pile or the stock pile. If drawing from the discard pile results in the agent's hand's deadwood value decreasing, then the agent always chooses to draw from the discard pile. When this isn't the case, the agent calculates an estimate of the utilities of either action. For drawing from the discard pile, the agent simulates the drawing of the respective card, and a reasonable discard action. The utility of drawing from the discard pile then becomes the utility of holding the resulting hand. The utility of drawing from the stock pile is calculated similarly, except rather than looking at a single card, the agents take the weighted average of the values produced for each card that isn't already either in the discard pile or the player's hand, with the values being weighted based on the probability of the respective card being in the deck (the complement of the probability that the opponent's hand contains the card). The reasonable discard action is typically set to the unmelded card that results in the highest hand utility when discarded. The only exceptions to this are that the agent will always pick the highest deadwood card in their hand if it results in a knock-able hand, and if all cards are melded, the agent will pick any card that ensures all cards are still melded after the

[1] A melding refers to a way that the cards in a hand can be melded.

discard. After producing these utilities for both draw actions, the agent will draw from the pile with the higher associated utility.

Discard Strategy

When actually choosing a discard, our agents will use a more sophisticated version of the strategy used to pick a reasonable discard above. Rather than just looking at the discard that results in the highest hand utility, the agents also seek to minimize the opponent's expected gain in hand utility as a result of that discard. In order to do this, for each unmelded card in their hand, our agents use their current prediction of the opponent's hand to sample 100 possible hands. For each hand, the agents will simulate a draw phase for the opponent, calculating a utility for drawing from the deck and a utility for drawing the hypothetical discard. The opponent's expected gain in hand utility for a potential discard is then set to the average gain in utility for the opponent when they choose to draw from the discard pile rather than the stock pile, with this gain being set to 0 if the utility of drawing from the stock pile is higher. Each unmelded card is then assigned a value equal to the difference between the hypothetical hand utility if that card is discarded, and OPP_UTIL_IMPORTANCE times the opponent's expected gain in hand utility from that card being discarded. The card with the highest value is discarded.

As before, the only exceptions to this are that the highest deadwood card is always discarded if it results in a knockable hand, and if all the cards in the hand are melded, any card that results in a hand that is still completely melded is discarded.

Knocking Strategy

For this project, we have chosen to focus primarily on developing the drawing and discard strategy. For the knocking strategy, our agents choose to knock as soon as the option is made available to them. This was done to allow the focus of the work to be on developing this technique for opponent hand estimation, rather than dealing with the complexities of a good heuristic-based strategy. Future versions of these agents could, however, implement a more complex knocking strategy without affecting the effectiveness of the approach used in this paper.

Training

Training the model took place in 3 phases. First, 2 agents using the above strategy were used in combination with the "SimpleGinRummyPlayer" agent provided by the competition hosts[2] to create an initial dataset. Several games were simulated between each pair of players, creating a total of approximately 32 million data entries, of which approximately 4 million involved the "SimpleGinRummyPlayer" as the opponent, and the remaining 28 million were split evenly between the two agents using our strategy. This data was used to train our first model, and the resulting model was used to create a new agent (this agent was re-tuned to account for changes in the optimal hyper-parameters due to the new model). This new agent was then put up against itself, and the 2 original agents, to create approximately 9 million more data entries. Again, this was used to create a new agent, which was re-tuned appropriately, and then used to create another 9 million data entries. The resulting dataset of roughly 50 million data entries was used to train our final neural network, which was used to create our final agent.

When tested on a test set with 1.5 million data entries following roughly the same distribution of agents used in the training data, our final model achieved an accuracy of 82.47%, correctly determining whether or not a card was in the opponent's hand 82.47% of the time, and a precision of 68.43%, correctly identifying cards in the opponent's hand as being in the opponent's hand 68.43% of the time. The hyper-parameters for our final agent were as follows:

MELD_BONUS	=	1.5
COMBINATION_BONUS	=	4.0
DEADWOOD_BONUS	=	1.0
KNOCK_BONUS	=	10
GIN_BONUS	=	20
OPP_UTIL_IMPORTANCE	=	0.77
LOW_CARD_BONUS	=	0.2
EMERGENCY_BOOSTER	=	2.5

Performance

Since Gin Rummy is a context in which very little work has been done, there isn't any benchmark to compare to, and as a result, measuring performance is difficult. In games like Poker, past works have used regret minimization to find the best response to a particular strategy, and have used it to measure an agent's "exploitability" (Johanson et al. 2011). This, however, is difficult to do here because of how large the game tree of Gin Rummy is, and how much computation our agent requires. Since there doesn't seem to be a good metric for measuring performance here, we will instead show an example of the agent in play, utilizing its opponent hand predictions to make better decisions.

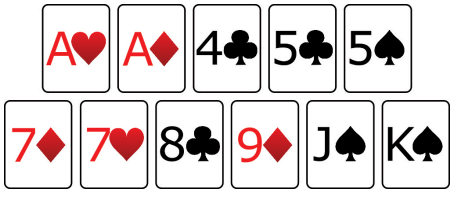

Figure 2: Example Hand

Consider a game state where the agent holds the hand shown in Figure 2 and is currently in the discard phase. In this circumstance, a naive agent might just discard the

[2] "SimpleGinRummyPlayer" is an agent provided by the hosts of the Gin Rummy EAAI Undergraduate Research Challenge. It follows a very basic strategy, drawing cards that contribute to melds, discarding cards that contribute the most deadwood, and knocking when possible.

card that results in the highest hand utility. Using our heuristic utility function, the optimal card to discard here would then be the 8♣ with a resulting hand utility of approximately 4.94. The K♠ would be a close second with a resulting hand utility of approximately 4.88. Our agent, however, would also take into account the opponent's previous actions and the resulting predictions from those actions. In the game that this hand is taken from, the opponent has just discarded the K♢. This, along with the rest of the input from the game state, causes our model to predict that there is only a 9% chance of the opponent having the K♣ and a 13% chance of them having the K♡. With the J♠ also being in the agent's hand, this means that discarding the K♠ is very unlikely to help the agent in creating new melds. This is confirmed by the agent as it calculates the opponent's expected gain in hand utility to be approximately 0.05. Conversely, as little to none of the opponent's actions suggest that the 8♣ won't be useful to the opponent, the agent calculates the opponent's expected gain in hand utility from the 8♣ to be approximately 1.61. As a result of this, our agent assigns a value of 4.85 to discarding the K♠ and 3.70 to discarding the 8♣. The agent discards the K♠, maintaining a high hand utility and simultaneously making it difficult for the opponent to increase theirs.

Obviously, the above example is not necessarily representative of how our agent generally plays. As more agents are published for the game of Gin Rummy, a more concrete estimate of the relative performance of our agent can be determined.

Future Work

There are a number of avenues that can be explored in order to improve this agent. First, as knocking strategy wasn't significantly studied in this work, a future agent could incorporate more complex knocking strategies into the heuristic strategy. Second, whereas our agent designs a model that only outputs a probability for each card and assumes the probabilities are independent, a future work could create a model that outputs second-order relative probabilities or even more probability information. Being able to sample efficiently from these distributions, however, would not be trivial and would likely be the primary difficulty in using this approach. Finally, while our agent only takes into account how much an opponent would immediately benefit from a card that is discarded, a future agent could simulate up to a further depth to get a more accurate measure of that benefit.

Future works could also create entirely new agents, combining the ideas from this paper with the more traditional approach of counterfactual regret minimization. The heuristic utility function from this paper could be used to provide an estimate of the value of any game state, allowing CFR to be used without traversing the entire game tree. It is also worthwhile to search for ways to modify existing approaches like those used in *DeepStack* and *Libratus* to work under reasonable time in the context of Gin Rummy, allowing good strategies to be found without game-specific heuristics.

Competition Modifications

The agent designed here is being used to compete in the Gin Rummy EAAI Undergraduate Research Challenge. Though we simplified some elements of the strategy when creating the original agent, with our focus being primarily on refining our approach for opponent-hand estimation, we added some elements to the strategy of the final agent to improve its standing in the competition. Specifically, we added more depth to the knocking strategy of our agent in the following four ways:

1. Rather than always knocking as soon as the option is available, if the current prediction of the opponent suggests that the opponent may be close to having a knockable hand or may already have a knockable hand, our agent only knocks if it has a hand with relatively low deadwood (less than or equal to 6).
2. To accommodate for the above change, when the opponent's hand's deadwood value does seem low, the agent no longer forces the discard of the highest card in its hand if it leads to a knock-able hand.
3. If the opponent knocks and the agent's hand has multiple possible meldings that minimize deadwood, the agent will choose the melding that maximizes the amount of deadwood it will be able to lay off.
4. If the agent knocks and has multiple possible meldings that minimize deadwood, the agent will choose the melding that minimizes the amount of deadwood the opponent will be able to lay off.

References

Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up limit hold'em poker is solved. *Science* 347: 145–149.

Brown, N.; and Sandholm, T. 2018. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* 359: 418–424.

Johanson, M.; Waugh, K.; Bowling, M.; and Zinkevich, M. 2011. Accelerating Best Response Calculation in Large Extensive Games. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume One*, IJCAI'11, 258–265. AAAI Press.

Moravčík, M.; Schmid, M.; Burch, N.; Lisý, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; and Bowling, M. 2017. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* 356: 508–513.

Sandholm, T. 2015. Abstraction for Solving Large Incomplete-Information Games. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, 4127–4131. AAAI Press.

Zinkevich, M.; Johanson, M.; Bowling, M.; and Piccione, C. 2008. Regret Minimization in Games with Incomplete Information. In Platt, J. C.; Koller, D.; Singer, Y.; and Roweis, S. T., eds., *Advances in Neural Information Processing Systems 20*, 1729–1736. Curran Associates, Inc.