A Deterministic Neural Network Approach to Playing Gin Rummy

Viet Dung Nguyen, Dung Doan, Todd W. Neller
Gettysburg College
{nguyvi01, doandu01, tneller}@gettysburg.edu

Abstract

This paper describes a deterministic approach to building a fixed-strategy gin rummy player. In the paper, we develop and evaluate both heuristic and neural network models for informing draw, discard, and knock decisions in the game. In this empirical study, we test performance of the models through competitive game play, show which best inform strategy, and demonstrate statistical significance of the improvement over a simple strategy. Through this empirical study, we indicate features that we expect to be helpful in future improvements to Gin Rummy play.

Introduction

Gin Rummy[1] is an imperfect information card game in which one collects cards in sets or runs called *melds*, minimizing *deadwood points* of unmelded cards so as to *knock*, i.e. end the hand, and have the least deadwood points. We use North American 25-point gin and undercut bonuses, and declare the winner as the first player to score 100 points or more over successive hands.

In this paper, we develop an AI Gin Rummy player that learns play strategy in simulated play with a simple Gin Rummy player class `SimpleGinRummyPlayer` supplies as reference implementation for a Gin Rummy research competition (Neller 2020). The Simple player implements a simple strategy:

- It ignores opponent actions and cards that are no longer in play
- It draws a face-up card only if it becomes part of a meld. Otherwise, it draws face-down
- It discards a highest-deadwood unmelded card, breaking ties randomly and without regard to breaking up potential meld patterns (e.g. pairs)
- It knocks as soon as possible

We restrict the scope of our work to approaches that are efficiently computable in real-time without high-performance hardware for the purpose of fast, good, accessible Gin Rummy play.

[1] https://www.pagat.com/rummy/ginrummy.html

We begin by describing prior work related to the computation of Gin Rummy strategy. We then introduce our deterministic approach for constructing or Gin Rummy Player. Next, we define heuristic and neural network models that engineer features that are helpful for learning play decisions: card goodness/badness, opponent hand estimation, and the goodness/badness of a decision to knock. We share our intuitions for these models, our experimental means of collecting training data, training non-heuristic features, and the results of such experiments. Comparing the performance of players that use different combinations of our engineered models, we observe their relative strengths, and conclude which models offer the greatest statistically significant improvements in Gin Rummy play.

Related Work

DeepStack

DeepStack (Moravčík et al. 2017) was one of two successful approaches to optimal Heads-Up, No-Limit (HUNL) Texas Hold'em Poker which, like Gin Rummy, is an imperfect information card game. DeepStack utilize concepts of counterfactual regret minimization (CFR) (Neller and Lanctot 2013), Monte Carlo Tree Search, and deep neural network learning in order to continually recompute a belief state regarding the opponent's two hole cards, which requires significant computational resources. The number of non-abstract information sets in Gin Rummy is prohibitively large given the 10-card hand in comparison to the 2-card hand of Texas Hold'em Poker.

TD-Rummy

The work by Kotnik and Kalita (Kotnik and Kalita 2003) is one of very few works that treat Gin Rummy specifically. In particular, their use of artificial neural networks (ANNs) influenced our decision to apply ANNs to our approach. In their work, they applied two different architectures of ANNs to Temporal Difference and Evolution Learning models. In their approach learning is continual and online, whereas we desire to pretrain models offline with batch learning so as to have fast real-time play decisions.

Deterministic Approach

Our approach to Gin Rummy play engineers features that can effect one or more types of Gin Rummy play decisions, training models that are based on fixed pattern decisions. In this section, we will describe the reason why we choose this approach which forms a fixed strategy Gin Rummy player, and we also compare it to other approaches.

As with many sequential games both simple and complex, the win/loss reward is delayed to the end of the game, and the value of any given state is temporally separated from stochastic feedback and thus not easily learned. As an imperfect information card game (Blair, Mutchler, and Liu 1993) like Poker, counterfactual regret minimization (Neller and Lanctot 2013) approaches as used by DeepStack could presumably compute an optimal or approximately optimal mixed strategy for the game. However, DeepStack's continual resolving based on the current state is incompatible with the real-time constraints of the research competition we develop our agent towards (Neller 2020). An entire game allows each player only 30 seconds of compute time for all game decisions across all hands. Therefore, we much seek a computationally efficient approach for our application. While one could employ CFR to compute mixed strategies offline for an abstraction of the state space, we initially take a simpler approach.

One approach to making fast, good play decisions in Gin Rummy is to apply reinforcement learning and train a deterministic player that maps states to actions without need to solve a subgame. As in the paper on TD-Rummy (Kotnik and Kalita 2003), we also build a reinforcement learning model that utilized an artificial neural network. We generated considerable play data using the contest `SimpleGinRummyPlayer` simulating self-play. Most decisions in play concern the draw (face-up/down), the discard, and whether or not to knock when it is an option. We treat each of these three types of decisions in isolation and seek to learn separate models for aiding each of these three decisions. These models then inform our reinforcement learning player's decisions. This player would then be able to generate better play data from self-play and close the loop. However, in this paper, we focus on what machine learning yields from `SimpleGinRummyPlayer` self-play. More specifically, we preprocess `SimpleGinRummyPlayer` self-play data to create four models that inform our three draw, discard, and knock decisions: hit card regression, opponent hand estimation, knock classification, and hand encoding.

Additionally, we decide to separate the training and playing process in order to implement desired behaviors, which is quite different from method used in the TD-Rummy paper where the neural network was forward- and back-propagated during game-play (Kotnik 2003, p. 24). In this study, we perform a single batch learning iteration on collected simple play data in order to train multiple models labeled with play outcome data. These models are then used to inform a second generation player that we then evaluate.

One limitation of this preliminary work is that reinforcement learning based on play against a fixed suboptimal benchmark learns to exploit that benchmark through play that is likely suboptimal itself. In the future, we envision that self-play as with DeepStack and its predecessors, would possibly yield optimal play.

Game Play Decision Models

Card Value Estimation

We define the set of *unknown cards* to be those cards which are not known to be in either player's hand or in the discard pile. We call an unknown card or the upcard, the top card of the discard pile, a *hitting card* if, in combination with at least one card in hand and possibly another unknown card, it is possible to form a 3-card meld. We define the *hit count* of a hitting card to be the sum over each card in hand of the number of 3-card potential melds shared by that hand card and the hitting card. The purpose of the *hit count* metric is to indicate the future melding potential of a card relative to one's current hand.

If we only draw cards that form melds but do not collect potential hitting cards, we do not quickly develop opportunities to meld. When we draw a card with a high hit count, we have a greater probability of forming melds faster, although doing so with a face-up draw offers information to the opponent of the current state of our hand.

Table 1 shows the results of a short experiment to determine whether there is an advantage to drawing hitting cards in addition to melding cards versus the `SimpleGinRummyPlayer`. In the "No Hitting" case, we note the number of cards drawn face-up and melds formed when only drawing cards that form melds. In the "Collect hitting" case, we note the same statistics when additionally drawing a hitting card with a probability of 0.5 when a hitting card is available.

Strategy	Draw face-up	Meld formed
No hitting	105,103 cards	171,170 melds
Collect hitting	123,504 cards	176,374 melds

Table 1: Average cards drawn and melds formed at the end of match over 50,000 hands of two play strategies

Card value with heuristic model: Let $\mathrm{Deadwood}(c_i)$ be the deadwood value of card c_i. Let n_i be the number of meld types (runs or suits) that exist in the potential meld set with card c_i (0 if there is no meld, 1 if there are just melds or just runs, or 2 if there are both types of meld). Hitting reward H, hand-tuned to 3, is a constant that weights the significance of having a higher potential of meld types for a card. The value function $R_i(c_i)$ of a face-up card, c_i, is defined as follows:

$$R_i(c_i) = 13 + (n_i \times H) - \mathrm{Deadwood}(c_i) \qquad (1)$$

If a face-up draw card would form or extend a meld, we automatically draw it without such heuristic evaluation. However, if it would now immediately form or extend a meld, we will choose to draw it if $R_i(c_i) \geq T$, where T is a hand-tuned threshold value. In Table 2, we can see some of the tuning results that indicate the number of melds formed with different threshold values. When this threshold is not met for a card, we instead draw face-down. Note that $T = \infty$ means that a player never draws the face-up card.

	Draw face-up	Melds formed
T = 1	62,664	10,470
T = 6	57,649	12,774
T = 21	19,355	17,409
T = ∞	10,988	14,396

Table 2: Performance of Hitting Player over Medium Player KH after 1000 games with different configuration of threshold value

Neural network approach: Our neural network model consists of a feed forward deep neural network tested on different numbers of neurons per layer and numbers of layers. Every unit of the network is a rectified linear unit. Our loss function is mean square error. Network inputs and output are as follows:

- Input - card deadwood points
- Input - turn of the game
- Input - card hit count
- Target Output: expected deadwood points from the card at the end of the hand

We simulated over 5,000 games with a `SimpleGinRummyPlayer` modified to additionally draw a face-up hitting card with a probability of 0.5, creating over 500,000 input-output instances for each upcard draw decision, with target outputs being the upcard deadwood value or 0 depending on whether the upcard contributed to a player's hand deadwood at the end of the hand.

Keras was used with this training data to train a feed-forward neural network with 3 layers with sizes 64, 32, and 1. Training used a batch size of 1000 for 100 epochs. The trained network appeared to converge to a MSE loss of 8.2. We used the network to make play decisions by noting that the network predicts the badness of drawing a card in the form of an estimate of how much that card will cost us in deadwood at the end of the hand. We therefore used the output of the network with a hand-tuned threshold value $T = .4$, so that we would draw the face-up card if the expected deadwood value was strictly less than T, and discarding the card estimated to cost the most deadwood.

We observed then that the play performance result of this neural network player against the original `SimpleGinRummyPlayer` was inferior to that of our aforementioned heuristic threshold model (Table 3). In the rest of our study, we thus prefer use of our heuristic model.

Opponent Hand Estimation

Hand estimation, i.e. estimating the probability of each card being in the opponent's hand, can supply important information for discard decisions. If one has a good opponent hand estimation, we can rate the relative safety of different discard options, i.e. how unlikely it would be for the discard to help the opponent form a meld. When an opponent discards a card or does not draw the upcard, we estimate that the opponent is less likely to hold cards that would form a meld with that card. Similarly, when an opponent draws the upcard, we estimate that the opponent is more likely to hold cards that would form a meld with that card. However, such observations do not specify an algorithmic approach to opponent hand estimation. In this section, we examine two algorithmic approaches to hand estimation: an application of Bayes Rule to simulated play data, and a trained LSTM network model.

	Heuristic model	Neural Network
Winning percentages over Simple Player	45 - 47 %	20 - 24 %
Average hitting card drawn per hand	0.04	2.63
Average deadwood per hand	15.54	18.43

Table 3: Heuristic and neural network model comparison over 10000 hands play

Bayesian estimation of an opponent's hand: One of our novel approaches began as an attempt to take a heuristic approach to opponent hand estimation and derive a similar approach that is Bayesian and play data driven. While the algorithm is not specified in (Rollason 2007), an opponent drawing a face-up card increases the estimated probability of having same rank or adjacent suit cards, and similarly, an opponent discarding a card or refusing a face-up card decreases the probability.

Our approach begins with Bayes' Rule and its simple proportional form:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2}$$

$$P(A|B) \propto P(B|A)P(A) \tag{3}$$

Here A represents the atomic sentence that the opponent holds a specific card, and $P(A)$ is the probability that the opponent holds that card. Event B is an observation of an opponent's draw and discard behavior on a single turn. Prior to event B we believe the probability of the opponent holding a specific card is $P(A)$. After event B, our posterior belief of $P(A|B)$ is proportional to the frequency of observing the same draw/discard event in simulated play data while holding that card, i.e. $P(B|A)$ times our prior $P(A)$.

We could naively collect play data to find the likelihood of event B conditioned on A, but we may apply a helpful abstraction that enforces a rational symmetry concerning (in)equality of the suits of the card in consideration, the card drawn, and the card discarded. For example, we have no reason to believe that the likelihood of the opponent drawing Q♣ face-up and discarding K♠ conditioned on the opponent holding K♣ should be any different than that of drawing Q♡, discarding K♢, and holding K♡, respectively. What matters is the suited/unsuited relationships of the draw/discard cards to the potentially held card.

For this reason, we abstract the frequency data we collect from play on draw/discard events for each card with respect to:

- Whether or not a card was drawn face-up
- Rank of the face-up card
- Rank of the card discarded
- Whether the card was suited with the face-up card and/or discarded card

Play data was collected from $10,000$ simulated games between two `SimpleGinRummyPlayers` that (1) only drew face-up when the card completed or extended a meld, (2) discarded randomly from discards that would maximally reduce deadwood, and (3) knocked at earliest opportunity. Far from optimal, this simple play nonetheless was adequate to capture general statistics that adjusted probabilities similar to the AI Factory Ltd. approach. In cases where fewer than 50 abstracted observations were observed, we opted not to update our hand estimation. Cards known to (not) be in an opponent's hand are probability 0 or 1 and are not updated.

Given that each player starts with 10 cards of a 52 card deck, our initial probability of a player holding a card is 0 for those cards in our own hand, and $\frac{10}{42}$ for the 42 cards that could be in the opponent's 10-card hand. After each opponent turn, we checked our frequency data to see if we had a minimum of 50 observations from which to form our expectation. If not, we conservatively do not revise our hand estimation. If so, we multiply each unknown card estimate by our frequency-based likelihood.

We then renormalize probabilistic estimates of unknown cards so that the estimates sum to the number of unknown cards in the opponent's hand.

Our approach was inspired by an ad-hoc technique of AI Factory Ltd sketched but not specified in a blog post (Rollason 2007). However, whereas AI Factory appears to adjust probabilities by chosen multiplicative constants, we apply Bayesian estimation with abstraction and independence assumptions.

LSTM Network Modeling: Although Long Short-Term Memory (LSTM) networks have traditionally been applied to time-series data from natural language, speech recognition, anomaly detection, etc. (Hochreiter and Schmidhuber 1997), we would observe that opponent hand estimations evolve in time series with observations of opponent play between each time step. We therefore attempted to build an LSTM network model with the following input-output specification:

- Input - One-hot encoded vector of which card the opponent discarded this turn
- Input - One-hot encoded vector of the upcard the opponent did not draw this turn (or zero vector if it was picked up)
- Input - One-hot encoded vector of the upcard that the opponent did draw this turn (or zero vector if it was not picked up)
- Input - One-hot encoded vector of known cards that are in our hand and in the discard pile
- Target Output - Probability vector of our estimation of the opponent's hand after the turn

Model: We simulated over 1,000 games between two `SimpleGinRummyPlayers` supplied for the research contest (Neller 2020), and use Keras to train the ANN model and LSTM model.

For our feed-forward ANN model, we first flatten the four input vectors and feed them through four rectified linear unit (ReLU) layers of sizes 512, 256, 128, and 64, followed by a final sigmoid layer of size 52, with one output unit corresponding to our belief that a card is in the opponent's hand. Training used a batch size of 1500 for 50 epochs. The training did not appear to converge according to the observed oscillation of the loss curve.

For our LSTM model, each of the four input vectors serve as inputs to each of four LSTM cells with size 128 output each. These four LSTM output vectors are concatenated as a size 512 input vector to a multilayer feedforward neural network with successive ReLU layers of size 768, 256, and 64. This is followed by a final size 52 sigmoid layer. For each successive turn, the aforementioned inputs are fed through the network, and stored ground truth of what the opponent held is used as feedback for training. Training used a batch size of 1500 for 130 epochs. The trained network appeared to converge to a categorical cross entropy loss of 12%.

Metrics: According to the normal accuracy metric, we decided to evaluate the accuracy of Gin Rummy hand estimation in 2 ways: categorical accuracy and binary accuracy. For categorical accuracy, the accuracy is measured by summing the card accuracy c_i (Equation 4) for each of the 52 cards, and then dividing that sum by the number of cards (Equation 5).

$$c_i = 1 - ||c_{actual} - c_{predict}|| \tag{4}$$

$$\text{Cate\_acc} = \frac{\sum_{i=0}^{n-1} c_i}{n} \tag{5}$$

The binary accuracy, on the other hand, rounds each predicted value to be zero or one using threshold function T (Equation 6), and divides the total number of true positives and true negatives by the number of cards (Equation 7).

$$T(x) = \begin{cases} 0, & \text{if } x < 0.5 \\ 1, & \text{if } x \geq 0.5 \end{cases} \tag{6}$$

$$\text{Bin\_Acc} = \frac{\sum_{i=0}^{n-1} 1 - ||T(c_{predict}) - c_{actual}||}{n} \tag{7}$$

When evaluating our LSTM network, for each turn, we input the vector data sequence ranging from the first turn through the most recent turn into our pretrained model. We then observe that our best metric accuracy for our trained LSTM model is an 81% categorical accuracy in the last turn of a hand (Table 4) and worse for earlier turns. However, performance according to this metric is worse than that of our application of Bayes' Rule for endgame and non-endgame turns, so we decided to use our Bayes' Rule estimation in the construction of our players.

Application of hand estimation to discard decision:

We apply our opponent hand estimation model to discarding decisions. In principle, discarding decisions consist of

	Cate_acc	Bin_acc
ANN Model	69.38%	60.85%
LSTM Model	81.07%	80.24%
Bayes System	82.30%	61.54%

Table 4: Metrical comparison among systems

two steps: choosing a set of candidate cards and choosing a discard card from among those candidates. In the `SimpleGinRummyPlayer`, the set of candidate cards consists of unmelded cards that minimize the deadwood points after the turn; then, from among those candidates, the player chooses the discard card randomly. In players with our hand estimation model, the players collect all unmelded cards in their set of candidate cards. Thus, the candidate set may contain lesser-deadwood cards that are safer to discard given what we believe about the opponent hand state. From among the candidates, the players will then choose the discard card with two criteria: the desirability of the card to the opponent and the card's deadwood points.

The opponent's desirability of a card is how much the opponent wants the card. Because a rational player will surely pick up a card if it completes a meld, we determine the desirability of a possible discard by assuming it to be in the opponent's hand and calculating the probability of the opponent's holding the rest of the cards in a certain meld determined by summing over each possible meld with that card the product of the estimated probabilities[2] of having each card of the meld, or

$$\text{Desirability} = \sum_{\text{all melds}} \left(\prod_{\text{cards of meld}} \text{Est\_Prob(card)} \right) \quad (8)$$

The desirability, as expressed in the formula, mostly falls within the range of [0,1], with some exceptions yielding greater-than-one values because of inaccuracies in our hand estimation. According to this formula, the lower the desirability is, the safer for the player to discard the given card.

The second criterion of the discarding decision is the given deadwood points of a card in the range of [1,10]. In rational play, the higher the deadwood points, the more a player would be inclined to discard a card.

To combine the two components into a single value for comparison, we decided to bring the two criteria to the same scale in a linear formula, which is suitable for our purpose of comparison. All other consideration being equal, the player's benefit of discarding a card runs counter to the opponent's desirability of that card, we decided to use the term (1 − desirability) to account for the importance of our opponent hand estimate, and the term also falls within the range of [0,1]. All other consideration being equal, the player's benefit is proportional to the discarded card's deadwood points and because the deadwood points fall into the range of [1,10], we decided to use the term (deadwood/10), for the term falls into the range of [0.1,1], similar to that of the desirability criterion. Given desirability D, deadwood points dw, desirability weight $W_{\text{DS}} = 1.0$, and deadwood weight $W_{\text{DW}} = 3.0$, we define a weighted heuristic formula for the discard value formula:

$$\text{Value}_{\text{discard}} = (1 - D) \times W_{\text{DS}} + \frac{dw}{10} \times W_{\text{DW}} \quad (9)$$

We use this formula to choose a discard most beneficial to the player by discarding the card with highest discard value (Equation 9) from among the candidates. The weight coefficients were manually tuned according to player performance.

[2]The estimated probabilities we multiply are conditioned on prior observed play history and our abstracted model for opponent play.

When Should We Knock?

The decision whether to knock or not is not trivial. Although one may knock after a turn where one has 10 or fewer deadwood points, it is not necessarily beneficial to do so. If one had perfect information, one would knock so as to maximize the expected utility, taking into account game scores, the outcome of laying off, etc., but without such information, one is left with the question of whether or not there is greater utility in continuing to reduce ones deadwood or even go gin, melding all cards.

In this section, we list input features we believe are relevant to a good knocking decision and describe our ANN approach to modeling the knocking decision. We frame the decision as a supervised learning problem with the following inputs and target output:

- Input - Number of turns
- Input - Number of deadwood points
- Input - Number of melds
- Input - Number of hitting cards
- Input - Number of cards drawn face-up by the opponent
- Target Output - 1 or 0 if knocking results in a hand win or loss, respectively

We simulated over 10,000 games between two modified `SimpleGinRummyPlayers` with varied deadwood thresholds ≤ 10 for knocking. This serves to provide a greater sampling of the knocking decision state space. We iterate through each valid pair of deadwood thresholds when simulating and collecting our play data. We then use Keras to train the ANN model with three ReLU feed-forward layers of sizes 128, 64, and 32, and a final sigmoid layer of size 1. We arrive at the knocking decision by thresholding the predicted value of the sigmoid layer, and knock if the threshold is exceeded. We experimented with different threshold values (Table 5), and found the optimal threshold to be .9.

Experimental Evaluation of Players

In the previous sections, we have engineered new models for aiding the three drawing, discarding, and knocking decisions. In this section, we construct players that make use of these models in order to modify the decision-making of our baseline `SimpleGinRummyPlayer` for comparison. We may describe our naming of these agents in a truth table style

	Loss	Acc	Increased winning percentages
Threshold=0.7	0.16	0.83	8%
Threshold=0.8	0.14	0.84	10%
Threshold=0.9	0.14	0.84	12%
Threshold=1.0	0.14	0.84	11%

Table 5: Winning percentage increases over `SimpleGinRummyPlayer` with knocking model. The dataset sizes are .5M and 1.0M for the first and subsequent rows, respectively

with model incorporation marked T/F if those models are incorporated into new decision-making for the `SimpleGinRummyPlayer` (Table 7).

Player decision-making with these models is described below:

- **Simple Player** is the `SimpleGinRummyPlayer` of (Neller 2020)
- **Knocking Player** substitutes the Simple Player knocking decision for that of our Knocking threshold model
- **Estimating Player** substitutes the Simple Player discard decision for one which makes the safest discard according to our Bayesian opponent hand estimation
- **Hitting Player** modifies both draw and discard decisions with the Card Value model so as to draw and retain cards predicted to maximize overall hand card value
- **Medium Player HE** brings together the Hitting and Estimating player modifications, by restricting discard candidates to non-hitting cards and choosing among candidate discards according to Equation 9 that moderates between card value and safety considerations. In the case that all cards are hitting-cards, then all cards are discard candidates
- **Medium Player KH** is constructed with both Hitting Player and Knocking Player modifications
- **Medium Player EK** is constructed with both Estimating Player and Knocking Player modifications
- **Advanced Player** is constructed with both Medium Player HE and Knocking Player modifications

Comparative Results

We played each pair of players against each other for 10,000 games and present winning percentages in Table 6. All of the confidence intervals are 90% Wilson confidence intervals (Wallis 2013) over win percentages.

We analyse our results in three player categories: players integrated with one decision-making model (the Hitting, Estimating, and Knocking Players), the players integrated with two decision-making models (the Medium Player KH, EK, and HE), and the player integrated with all three decision-making models (the Advanced Player). The players' competencies are mostly based upon their performances against the Simple Player. Although other observations of the created players playing against each other provide important insights, we limit ourselves from reading too much into performance of our own players against each other.

Among the players integrated with one decision-making model, the Knocking Player performs well against the Simple Player, which is evident from its winning percentages in the interval $[57\%, 59\%]$, while the Estimating Player and the Hitting Player show a clear under-performance when playing against the Simple Player with both winning percentages in the interval $[45\%, 47\%]$.

Among the players integrated with two decision-making models, the Medium Players integrated with the knocking model outperform the Simple Player: the Medium Players KH and EK have greater-than-50% winning percentages against the Simple Player, in the intervals $[58\%, 60\%]$ and $[56\%, 57\%]$, respectively. The Medium Player HE, which is not integrated with the knocking model, performs poorly against the Simple Player with a winning percentage in the interval $[45\%, 47\%]$. Furthermore, we also notice that the Medium Player KH performs slightly better than the Knocking Player, and the Medium Player EK performs worse than the Knocking Player against the Simple Player. This suggests that the card value model integrates better with the knocking model than does the hand estimation, and that the hand estimation model might expose the player to making worse decisions than those of the Simple Player's random mechanism. Also, HE performs just as well as the simpler Estimating and Hitting Players, suggesting that the card value model does not integrate well with the hand estimation model.

The Advanced Player, which integrates all three decision-making models, continues to gain the advantage of the knocking model in its strategy and performs well against the Simple Player, with a winning percentage in the interval $[56\%, 58\%]$. Its performance against the Simple Player is better than that of the Medium Player EK and worse than that of Medium Player KH. In experiments against the Knocking Player and the Medium Player KH, the Advanced Player also shows a poor performance with winning percentages in the intervals $[42\%, 44\%]$ and $[41\%, 43\%]$, respectively. These observations support our earlier conclusion: the integration of the hand estimation model opens up a weakness in the player's strategy. The Medium Player KH also performs better against the Advanced Player than the Knocking Player does, showing again that the integration of the knocking model and the card value model slightly boosts the player's performance.

From our experiments with these players, we can conclude that the integration of the knocking and the card value model in the Medium Player KH is the the best strategy we developed against the Simple Player.

Future Work

Training cycle: In this work, we have performed a single training iteration on Simple Player play data, but in the future we could use play data from our Advanced player's self-play and close the loop of a reinforcement learning training cycle.

	Simple Player	Hitting Player	Estimating Player	Knocking Player	Medium Player KH	Medium Player HE	Medium Player EK	Advanced Player
Simple Player		53 - 55	53 - 55	41 - 43	40 - 42	53 - 55	43 - 44	42 - 44
Hitting Player	45 - 47		48 - 49	38 - 40	37 - 39	48 - 50	35 - 36	34 - 36
Estimating Player	45 - 47	51 - 52		30 - 31	30 - 31	48 - 50	32 - 33	31 - 32
Knocking Player	57 - 59	60 - 62	69 - 70		49 - 50	68 - 70	57 - 58	56 - 58
Medium Player KH	58 - 60	61 - 63	69 - 70	50 - 51		68 - 69	58 - 60	57 - 59
Medium Player HE	45 - 47	50 - 52	49 - 51	30 - 32	31 - 32		32 - 33	30 - 32
Medium Player EK	56 - 57	64 - 65	67 - 68	42 - 43	40 - 42	67 - 68		48 - 49
Advanced Player	56 - 58	64 - 66	68 - 69	42 - 44	41 - 43	68 - 70	51 - 52	

Table 6: 90% Wilson confidence intervals of win percentages for each row player against each column player for 10,000 games.

Card Value	Hand Estimation	Knocking	Player Name
F	F	F	Simple Player
F	F	T	Knocking Player
F	T	F	Estimating Player
F	T	T	Medium Player EK
T	F	F	Hitting Player
T	F	T	Medium Player KH
T	T	F	Medium Player HE
T	T	T	Advanced Player

Table 7: Simple Player model incorporation

Improved application of opponent hand estimation: While we achieved some measure of success in estimating the relative likelihood of an opponent holding each unknown card, our poor play performance in applying that model suggests that there are better ways to make use of such information. Improved application of such opponent hand estimation to play decisions should be a priority going forward.

Hand abstraction model: In AI Poker research, work has been done to abstract similar Poker hands through binning (Shi and Littman 2001). This has the benefit of reducing the number of information sets, thus computationally simplifying analysis. An important question looking forward is whether or not we can similarly abstract Gin Rummy hands so as to simplify analysis for game play.

One approach might be to learn an auto-encoder that essentially engineers a compact representation of a Gin Rummy hand, reducing dimensionality and perhaps revealing important features for decision-making. Traditionally, autoencoders were used for dimensionality reduction (Wang, Yao, and Zhao 2016) or feature extraction (Meng et al. 2017).

Conclusion

In this work, we introduce both heuristic and learned models for card goodness/badness, opponent hand estimation, and knocking goodness/badness. For the first two of these, we define multiple models for evaluating alternatives. For the learned models that appeared best according to our loss functions, we used these to construct seven different players to compare in performance against each other and a baseline `SimpleGinRummyPlayer`. Each of these seven are formed by using or not using one of our three decision-making enhancements with respect to the Simple Player.

In Table 6, we observed that the best performance of our seven player population was Medium Player KH, which is constructed from decision components of the Knocking Player and Hitting Player, and demonstrates a statistically significant improvement over the Simple Player. Most of this improvement can be attributed to our improved knocking decision.

The Knocking Player uses our best performing deep neural network that learns knocking decisions from play data generated with a modified Simple Player with varying deadwood thresholds for knocking. The Hitting Player modifies both draw and discard decisions with the heuristic Card Value (Equation 1) so as to draw and retain cards expected to maximize overall hand card value.

In conclusion, we have constructed models to aid in drawing, discarding, and knocking decisions in Gin Rummy. Relative to the Simple Player, we see that our knocking decision model yields the greatest improvement in win-rate performance.

References

Blair, J. R. S.; Mutchler, D.; and Liu, C. 1993. Games with Imperfect Information. In *Symposium on Games: Planning and Learning*, 59–67. URL https://www.aaai.org/Papers/Symposia/Fall/1993/FS-93-02/FS93-02-009.pdf. AAAI Technical Report FS-93-02.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation* 9(8): 1735–1780.

Kotnik, C. L. 2003. *Training Techniques for Sequential Decision Problems*. Master's thesis, University of Colorado at Colorado Springs, Colorado Springs, Colorado, USA. URL http://www.cs.uccs.edu/~jkalita/work/StudentResearch/KotnikMSThesis2003.pdf.

Kotnik, C. L.; and Kalita, J. 2003. The Significance of Temporal-Difference Learning in Self-Play Training TD-Rummy versus EVO-rummy. In Fawcett, T.; and Mishra, N., eds., *20th Int'l Conf. on Machine Learning (ICML 2003)*, 369–375. Washington, D.C., USA: AAAI Press. ISBN 978-1-57735-189-4. URL https://www.aaai.org/Papers/ICML/2003/ICML03-050.pdf.

Meng, Q.; Catchpoole, D.; Skillicom, D.; and Kennedy, P. J. 2017. Relational autoencoder for feature extraction. In *2017 International Joint Conference on Neural Networks (IJCNN)*, 364–371. doi:10.1109/IJCNN.2017.7965877.

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 No-Limit Poker. *Science* 356. doi:10.1126/science.aam6960.

Neller, T. 2020. Gin Rummy EAAI Undergraduate Research Challenge. URL http://cs.gettysburg.edu/~tneller/games/ginrummy/eaai/. Last accessed on 2020-08-20.

Neller, T.; and Lanctot, M. 2013. An Introduction to Counterfactual Regret Minimization. URL http://modelai.gettysburg.edu/2013/cfr/cfr.pdf. Last accessed on 2020-08-15.

Rollason, J. 2007. Predicting Game States in Imperfect Information Games. URL https://www.aifactory.co.uk/newsletter/2007_02_imperfect_info.htm. Last accessed on 2020-08-16.

Shi, J.; and Littman, M. L. 2001. Abstraction Methods for Game Theoretic Poker. In *Computers and Games. CG 2000. Lecture Notes in Computer Science*, volume 2063. Springer, Berlin, Heidelberg. URL https://www.cs.rutgers.edu/~mlittman/papers/cg00-poker.pdf.

Wallis, S. 2013. Binomial Confidence Intervals and Contingency Tests: Mathematical Fundamentals and the Evaluation of Alternative Methods. *Journal of Quantitative Linguistics* 20(3): 178–208. doi:10.1080/09296174.2013.799918.

Wang, Y.; Yao, H.; and Zhao, S. 2016. Auto-encoder based dimensionality reduction. *Neurocomputing* 184: 232–242. ISSN 0925-2312. doi:https://doi.org/10.1016/j.neucom.2015.08.104. URL http://www.sciencedirect.com/science/article/pii/S0925231215017671. RoLoD: Robust Local Descriptors for Computer Vision 2014.