Final Report for CS394R - RoboHearts: Simple States: The Importance of Features in a Partially Observable Markov Decision Process

Lucas Kabela
lak2258
lucaskabela@utexas.edu

Akash Kwatra
awk457
akashkw@utexas.edu

!!!!!! Things to review throughout paper (First Draft): 1. Abstract provides essential motivation and primary findings of the paper 2. Ideas are logically structured (including sequence of sections, and where sections are) 3. No major conceptual errors in paper (topics and methodology align) 4. Writing should be in present tense

!!!!!! Things to review throughout paper (Final Draft): 1. References to papers, figures, and statistics (numbers) are correct 2. Formatting is consistent, significant figures match, 3. Proper Attribution to sources, ideas, and code 4. All information from Assignment Page is present 5. No spelling errors, writing is of a proper level

Abstract

We examine the results of using various combinations of state features in value approximation for the game of hearts. Features for the cards in a player's hand, the cards in play, the cards previously played, and the cards won by each player were evaluated in both simple linear approximation and nonlinear approximation with a neural network. Our results indicate that both simple linear and nonlinear function approximation can learn to perform 5 – 7% better than random with just the feature for cards in the player's hand, with nonlinear methods outperforming linear ones by 2% with this limited feature representation. Using history, cards in hand, and cards in play, we produce an agent using REINFORCE with baseline that performs ???% better than random, and ???% better than the full set of state features while training in xyz less minutes.

1 Introduction

Reinforcement learning has enjoyed a plethora of recent success in many game settings, thanks in part to the emergence of deep reinforcement learning in recent years. One domain which reinforcement learning has excelled in is card games, with recent results such as those of Brown & Sandholm (2019) outperforming professional poker players. However, networks that have fueled these victories have traditionally processed the raw state information, such as Mnih et al. (2013) work on Atari, which processed a 84 x 84 gray-scale cropped region of the game state, which totals 7056 values per frame, which are then stacked 4 at a time, resulting in 28224 parameters. As problems in the learning community continue to scale as with the computational cost and complexity of deep networks, it is worth investigating if there is a simple paradigm in which the important regions of raw state space can be identified, and thus model sizes and training time can be reduced with minimal loss in performance.

For our initial inquiry, we seek to obtain empirical evidence in a domain which relies heavily on state representation and has a rich history of feature analysis. Card games are one such domain, in which features can provide not only direct information from processing, but also lend themselves to inference by clever agents. One such game is Hearts, which is naturally phrased as a POMDP in which a player seeks to minimize the points they win. Due to the partially observable nature of this game, literature for feature abstractions and state representation is dense. Furthermore this domain has seen impressive gains due to raw feature input in neural networks by Wagenaar (2017). Finally, and perhaps most importantly, the entire raw feature space for the domain is rather small, so lends itself to study under different combinations of features, and naturally decomposes into a small number of them. Thus, we find it natural to select this domain to pose the question of how state representation through subsets of raw features impacts the performance of an agent. To answer this question, we attempt to empirically study the effect of different feature sets in both linear and nonlinear function approximation.

2 The Game of Hearts

2.1 Description of Hearts

We believe Hearts to be a unique setting for feature representations and state simplification as a result of the many complexities of its rules, which we describe here. The game of hearts is a trick-winning game, a “trick” being defined as a collection of four cards won after each player places down a card. In this game, four players are dealt 13 cards to play one round, composed of 13 tricks. The winner of each trick is the player that has placed down the highest card of the leading suit. In the event a player does not have a card of the leading suit, any card may be played (with the exception of the first trick - in which hearts cannot be played). For each heart in a trick, the winner of the trick gets one point. The queen of spades awards 13 points. After 13 tricks, points are counted, cards are dealt, and another round begins. These rounds repeat until one player accumulates 100 points, at which point the player with the fewest points is declared the winner.

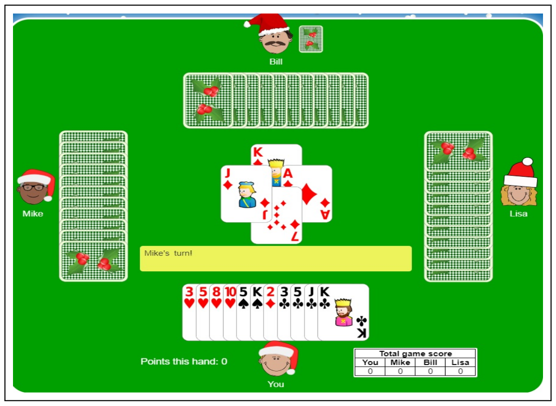

Figure 1: A trick of Hearts, game available at `https://cardgames.io/hearts/`

If a player manages to win every single heart and the queen of spades in one round, they have “Shot the Moon”. Shooting the moon awards 0 points to the winner, and 26 points to all other players. In one round, the player with the 2 of spades leads the first trick, with the winner of the previous trick leading all subsequent tricks. Hearts cannot be used to lead a trick until someone has “broken hearts”, meaning someone has placed down a heart in a previous trick. A final rule is that before each round, players can select three cards from their hand to trade with another player.

2.2 Hearts as a MDP

The first task of our research requires embedding the rules of Hearts into an environment for our reinforcement learning agent. We use the rules of the game to shape the formulation of our MDP. We begin by defining the environment as discrete time, as the game of hearts can easily be broken up into logical units of time. In increasing order of granularity, we have the game, which we consider an episode, the round, and the trick, which corresponds to a single time step in which the agent selects an action and receives reward.

The raw state representation includes the current score of all the players, point bearing cards that have been won by players, cards that have been played, cards in play in the current trick, and cards in agent’s hand. This is the full extent of state information available to players; however, from this state information, it is not possible to fully observe the state of the game, as the cards in other players’ hands are hidden - therefore it is natural to formulate Hearts as a POMDP. Similarly, the action space is the choice of card to play from the agent’s hand, and 3 cards to trade at the beginning of each hand. This actions space is dynamic - the size of the action space is constantly changing between each trick, and is not even fixed for card passing, as the cards available to pass will be different each hand. The negative of the score received from playing a card serves as the reward function, which is a natural formulation as the agent will seek to minimizing score. Lastly, the game of Hearts is clearly multi-agent, with up to 4 players in classic settings each making information based on observations of the environment.

2.2.1 Restricted Formulation

To produce a tractable setting for the examination of our problem, we made several restrictions to this MDP. Our first major relaxation was to lighten the partial observability. While other works such as Ishii et al. (2005) have dealt with this assumption, we found that working within a full POMDP framework was overly strict for investigating raw state features, so we choose to ignore this aspect of the game, overcoming some of these issues by opting to add history of cards played as a raw feature of state for our agent. Another important abstraction is our treatment of action space. We have already acknowledged the dynamic and complex dimensionality of our action space. To address this issue, we treat the action space as a constant dimension 52, and mask the output of our approximators to only consider valid moves for any given hand. To overcome the drastically different task of passing cards, we simply randomized the 3 cards we would pass, and since the agents we compared against were random too, this became effectively abstracted as part of the environment. Lastly, we relaxed the multi-agent setting, considering enemy agents as part of the environment for our investigation into feature space. We believe this assumption to be especially strong, as the agents are random. With these relaxations, we gain tractability turn to previous works for motivation.

3 Relevant Works

The game of Hearts has been heavily studied in terms of feature engineering and abstractions (Sturtevant & White, 2007) (Sturtevant, 2003) (Wagenaar, 2017)(Ishii et al., 2005). Our work draws inspiration from the features, abstractions, and motivations of previous research in this area heavily, and uses the important concept of state abstractions to motivate our investigation into the importance of features in states, and their abstractions (Ho et al., 2019). However, we found most of this prior work to be outdated, as the bulk of it was published pre-deep learning. Therefore, to fully leverage modern techniques and learning in this domain, we looked into more contemporary efforts to solve partially observable games such as the work of Brown & Sandholm (2019) Silver et al. (2016) and Charlesworth (2018)

One strategy we found particularly prominent in these works was the idea of using neural networks for function approximations, and training them through policy gradient techniques, with some of the most popular techniques being PPO Schulman et al. (2017) and Neural Fictituous Self PlayHeinrich & Silver (2016), both of which have achieved remarkable success. After further investigating these techniques, we empirically believed these suite of policy gradient methods to be particularly robust, and hoped to implement these in our work to provide a quantifiable upper bound on the performance of our reduced raw state representations.

4 State Abstraction

From our investigation into previous research, it became immediately apparent that the state space for Hearts would be intractable to learn using a tabular method. Hearts is not alone in this issue - many card games explode exponentially in the number of states that can be encountered during play. Fortunately, a great deal of prior work such as Wagenaar (2017), Sturtevant & White (2007) and Sturtevant (2003) has been done into investigating feature engineering in the game of Hearts. However, little work has been done into investigating which features are empirically the most important to the success of agents, and as such our first task was to produce a set of raw state features that captured all information an agent could need in the game of hearts.

The most important feature in our state abstraction to represent was the hand. We wanted to present important information to the agent about the game state, with the most important being the hand. We produced a binary feature vector (1 or 0) for the cards in the agents hand on each step, which was a constant length 52. This constant length encoding enabled us to use a flexible learning framework for the duration of the problem - one neural network for every trick in a round. The next most important portion of state space to model is the cards that are played during a trick. Without a representation of these cards, it is not possible for the agent to know if it will lose a hand before playing its card, although, due to the partial observability of the problem, this cannot be learned perfectly even with this feature. Therefore, we next modeled the cards in play as another 52 length binary feature vector.

As imperfect agents, we believe this is most of the representation that humans will build during a game of Hearts. However, expert card players utilize more information during the game in the form of the history of cards for a round. Using a strategy referred to as ”counting cards”, experts can minimize their uncertainty about opponents cards from cards that have been played. This information can also inform the risk of their strategy, especially due to the presence of the queen of spades or not in this history. Thus, we encoded the history of played cards with another 52 length binary vector. Similarly, player scores may also inform the risk of players, as a player with a significantly higher score than others may be significantly more risk averse. This feature was encoded as a length 4 integer vector.

Finally, we introduce features for the extreme case of shooting the moon. In particular, if an opponent wins all the points during a hand, then everyone, including the agent receives 26 points for that round, while the opponent receives 0. Therefore, we wanted to add information for a suitably smart agent to prevent shooting the moon. This came in the form of 4 14 length binary vectors, one for each point bearing card that could be won by a player. We hope this information, in combination with player scores, would be sufficient to prevent shooting the moon.

These features define the raw features of any state, and we believe provide all information an agent could hope to extract from the environment. However, other approaches such as Sturtevant & White (2007) attempt to further distill this information in favor of specific heuristic based features. While we find such abstraction important, as addressed by Ho et al. (2019), these abstractions are outside of the scope of this project which investigates raw features, or features that can be directly extracted from the game observations without processing. Thus, we opted to pass more raw information into a large neural network, in imitation of previous work by Brown & Sandholm (2019) and Charlesworth (2018) to examine the importance of each feature. Therefore, our next task was to devise a series of empirics to investigate these questions.

5 Monte Carlo and Classical methods

5.1 Linear vs Non-Linear Function Approximation

Our initial inquiry was to determine the effectiveness of linear and nonlinear value function approximation with respect to each other. Intuitively, the nonlinear representation can model more interesting functions, such as the xor, which linear functions cannot, so we expected the nonlinear model to greatly out perform the linear model. Both techniques were trained against random agents for 10000 episodes, then evaluated on a 25 sets of 1000 games against random agents, with the average number of wins recorded. The weights were learned using Monte Carlo rollouts, due to efficiency of rollout samples we could collect. For this portion of the experiments, we used the smallest and simplest set of features possible, which is only the cards in the agent’s hand

For the linear model, we simply used a weight vector which was dotted with the binary feature vector calculated for each state. The weight vector was initialized uniformly to 0, and weights were learned with a parameter α = 1e-4, which was found from a sample of 5 values of α to be the highest. The policy π was an ϵ-greedy method, with ϵ set to .05, as this value is fairly standard for ϵ greedy in practice. Lastly, we performed a hyper parameter search on γ, and found that the technique performed best for γ between .8 and .99, with no significant variance within this range. We believe this higher value of γ worked better for this setting as nonzero rewards were not rewarded by the environment until the end of a round, in which case the entire score for the round was given to the player.

Surprisingly, we found with a setting of $\alpha = 1e-4, \epsilon = .05, \gamma = .98$, linear function approximation produced results approximately 5% better than random, with extremely high statistical significance using the t-test method (*see table 5.1*). We found these results especially surprising as the feature representation was incredibly limited. The agent only had access to the cards in the hand, meaning the values learned simply corresponded to what cards are ”good”, and which cards are ”bad”. From the weight vector we found, available in appendix C, in particular, the 2 of clubs is learned to be the only positive value, as the agent will often not win tricks if it leads with this card. Similarly, face cards of all suits, but especially hearts are learned to be especially high value, which we expected as face cards are more likely to win suits. However, one anomaly with this data is that the values do not change linearly, it appears in almost all suits there is a significant jump from the 10 to the Jack.

	MC w/ Linear	Random
Mean	30.792	25.932
SD	1.343	1.255
SEM	.269	.251
N	25	25

Figure 2: MC % Wins with Linear Approximation over 1000 Games

For the nonlinear model, we used a 2 layer, 256 node network with ReLU for the non linearity. These parameters were chosen from prior work by Wagenaar (2017). We trained this network with the Adam Optimizer, and used the same setting of parameters as the nonlinear case, as we expected these to be particularly robust. We performed a hyper-parameter search on a logarithmic scale on λ, the learning rate, and found 1e-4 to be the most effective. So, we trained the neural network on 10000 episodes with $\alpha = 1e-4, \epsilon = .05, \gamma = .98$, and recorded an improvement over random by 7.5%. Once more this exceeded our expectations for the simple feature representation chosen (*see table 5.1*).

Group	MC w/ NN	Random
Mean	33.3	25.7
SD	1.339	1.319
SEM	.0423	.0417
N	10	10

Figure 3: MC % Wins with NonLinear Approximaton over 1000 Games !! NEED TO UPDATE

Comparatively however, we expected the nonlinear function approximation to perform significantly better than linear approximation due to its ability to encode interactions between features. However, we discovered this gain was not as significant as we expected for the in-hand representation. This is likely because the network has issues learning a valid parameterization that is robust to hand size - less active features will regularly score lower, so unfortunately it may be the case the policy chooses cards that happen to lower the score by reducing the number of activations. Another explanation may be that the nonlinear interactions between cards in hand is not an important contributor to performance, although we do not think this is likely, as having a hand of cards that are the same suit will be more risky for the player, unless they have all of the cards of the same suit (although such a hand has astronomically small chances of occurring). From these experiments, we established, at least in the case of Hearts, that nonlinear function approximation is indeed more powerful than linear function approximation - however, training took almost an hour for nonlinear function approximation, while it only took on the order of 10 minutes for linear approximation.

5.2 Importance of Raw Features

With the results of the previous subsection established, we proceeded to use combinations of raw state features with the nonlinear function approximation to investigate their importance in agent performance. We used the same process as the previous section - 10000 training iterations, with the same hyper parameters. However, we changed the input to the neural network by trying different combinations of features, which required scaling the number of nodes in the hidden layers. We chose this scaling to be 2*input features for the first layer and 4*input features for the second. The results and combinations are established in

GRAPH OR TABLE GOES HERE FOR THE FEATURE DATA

We first examine all pairs of combinations. Here, it was found that the combination of in-play and in-hand perform best, reaching a 33% win rate, an 8% improvement over random with a limited set of features. Perhaps more surprisingly was just how good this combination of features did over all others. Comparatively, the next best combination was in-hand and scores, which we believe to be the next best due to the small size of the score features. We believe the weights learned for these features were very small, so did not largely impact the learning of in-hand. However, we notice that the history feature did not perform well with the in-hand, surprisingly causing a decrease. We believe this to be an artifact of the difficulty for the feature to be learned - it is a complex feature to fully take advantage of, and this simple two layer network likely was not complex enough to learn the interactions efficiently. Finally, the combination of hand and cards won also did poorly. However, this is to be expected - these features are provided only to prevent shooting the moon, an important strategy in intelligent agents, but likely an astronomically unlikely event for the random agents we learned against. Therefore, the important observation of the data from pairs is that not all representations of raw state information are the same, and removing some information may improve the learning of models.

Next, we looked at higher order combinations of features. For our purposes, we did not try triplets including score, as the agents were random so score would not provide any meaning full information for our agent's strategy. Thus, we tried three triplets of in play, cards won, in hand, and in play, history, in hand, and cards won, in play, in hand. Unsurprisingly the best triplet was found to be in play, history and in hand, as we have mentioned the fault of cards won with this level of agent. However, it is interesting to observe that this triplet did not perform better than just in hand and in play. Lastly, when we tried all features together, again unsurprisingly the model performed worse. Once more this can be seen because the complexity of modeling the relation between these features.

6 Policy gradient

Now, using the results we found in section 5, we wanted to provide an upper bound on the accuracy of the more limited and entire set of features using state of the art methods for reinforcement learning. We decided on REINFORCE as a stable algorithm, compatible with the framework of our project. Thus, we implemented a policy gradient method, using a value network as a baseline for the REINFORCE algorithm. The output of the policy gradient network was a 52 length vector converted to a soft-max distribution, which models the probability of selecting each card to play. We then took the dot product of this output and the binary representation of valid cards (0 or 1 for each card) to produce a distribution over possible actions for a given hand. We rained two models with this network under the REINFORCE framework - one model used the combination of features we found to be optimal in section 5. The other network trained with all potential features of the state. Of interest was the final accuracy of both networks, and the time taken to train. We observed the model with the limited feature set achieved ... % wins over random in xxx minutes of training time, while the full feature set network achieved % wins over random in xxx minutes of training time. Surprisingly, the REINFORCE method was much faster to train than the NN - this is likely a result of how we modeled the policy (epsilon greedy) which required up to 13 look aheads for the network vs the simplicity of policy gradient generating a action instead of values.

TABLE FOR POLICY GRADIENT (RETURNS.TXT)

7 Future Work

Our work provides promising direction for future endeavours for agents in card games and general domains which lend themselves to state space abstraction and feature approximation. This research was mostly empirically based, however it would be intriguing to see if some mathematical framework for the importance of different features can be developed, instead of the heuristic based approach of this work. Additionally, we would be interested in investigating a general purpose analysis across domains to see if there is some common features of state space which are especially important (for instance, a meta study on Atari games). In terms of techniques themselves, several limitations in our experiments can be overcome in future work. In particular, future work should include more computational time for training the networks in our work, as many gains in recent game agents have been made because hundreds of hours of GPU time (Brown & Sandholm, 2019). We believe larger and deeper networks may be the key to more impressive results, as well as training against agents that are more powerful than random, perhaps in a multiagent framework. Further, our hyper-parameter analysis was conducted by manual search over set intervals for parameters, but Bayesian parameter optimization could be promising to improve performance. We believe these theoretical and empirical aspects of our work can be expanded for interesting results

8 CONCLUSION

In conclusion, our results provide empirical quantification for the importance of state abstraction and function approximation in the game of hearts. In particular, we evaluate the utility of a variety of feature approximations, and find the best performing combination in the nonlinear function approximation to be .... Using state of the art methods, in particular PPO, we train two networks, one with the full state of features, and another with a smaller subset to produce a network trained in xyz time that is zzz % accurate compared to the full state space approximation. We hope this work encourages future researchers to prune feature states in nonlinear approximation, and informs the development of a mathematical framework for the importance of features in value approximation.

ACKNOWLEDGMENTS

We would like to acknowledge Dr. Peter Stone and Dr. Scott Niekum for their instruction and guidance this semester. We would also like to acknowledge the OpenAI-Gym-Hearts project located at `https://github.com/zmcx16/OpenAI-Gym-Hearts` which provided a great deal of the starter code for our project

References

Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019. ISSN 0036-8075. doi: 10.1126/science.aay2400. URL `https://science.sciencemag.org/content/365/6456/885`.

Henry Charlesworth. Application of self-play reinforcement learning to a four-player game of imperfect information. *CoRR*, abs/1808.10442, 2018. URL `http://arxiv.org/abs/1808.10442`.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121, 2016. URL `http://arxiv.org/abs/1603.01121`.

Mark K Ho, David Abel, Tom Griffiths, and Michael L Littman. The value of abstraction, Jun 2019. URL `psyarxiv.com/6fm9a`.

Shin Ishii, Hajime Fujita, Masaoki Mitsutake, Tatsuya Yamazaki, Jun Matsuda, and Yoichiro Matsuno. A reinforcement learning scheme for a partially-observable multi-agent game. *Machine Learning*, 59(1):31–54, May 2005. ISSN 1573-0565. doi: 10.1007/s10994-005-0461-8. URL `https://doi.org/10.1007/s10994-005-0461-8`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016. URL `http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html`.

Nathan R. Sturtevant and Adam M. White. Feature construction for reinforcement learning in hearts. In *Computers and Games*, pp. 122–134, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-75538-8.

Nathan Reed Sturtevant. *Multiplayer Games: Algorithms and Approaches*. PhD thesis, University of California, Los Angeles, 2003. AAI3089030.

M. Wagenaar. Learning to play the game of hearts using reinforcement learning and a multi-layer perceptron. 2017.

A Appendix

You may include other additional sections here.

B Citations, figures, tables, references

These instructions apply to everyone, regardless of the formatter being used.

B.1 Footnotes

Indicate footnotes with a number[1] in the text. Place the footnotes at the bottom of the page on which they appear. Precede the footnote with a horizontal rule of 2 inches (12 picas).[2]

B.2 Figures

All artwork must be neat, clean, and legible. Lines should be dark enough for purposes of reproduction; art work should not be hand-drawn. The figure number and caption always appear after the figure. Place one line space before the figure caption, and one line space after the figure. The figure caption is lower case (except for first word and proper nouns); figures are numbered consecutively.

Make sure the figure caption does not get separated from the figure. Leave sufficient space to avoid splitting the figure and figure caption.

You may use color figures. However, it is best for the figure captions and the paper body to make sense if the paper is printed either in black/white or in color.

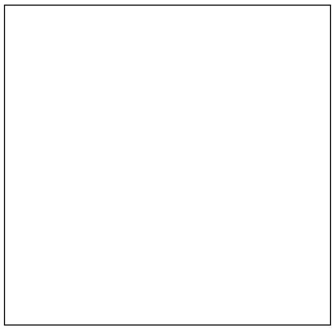

Figure 4: Sample figure caption.

B.3 Tables

All tables must be centered, neat, clean and legible. Do not use hand-drawn tables. The table number and title always appear before the table. See Table 1.

Place one line space before the table title, one line space after the table title, and one line space after the table. The table title must be lower case (except for first word and proper nouns); tables are numbered consecutively.

C Default Notation

In an attempt to encourage standardized notation, we have included the notation file from the textbook, *Deep Learning* **?** available at `https://github.com/goodfeli/dlbook_`

[1]Sample of the first footnote

[2]Sample of the second footnote

Table 1: Sample table title

PART	DESCRIPTION
Dendrite	Input terminal
Axon	Output terminal
Soma	Cell body (contains cell nucleus)

`notation/`. Use of this style is not required and can be disabled by commenting out `math_commands.tex`.

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0,\dots,0,1,0,\dots,0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_\mathcal{G}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathbf{\mathsf{A}}$
$\mathbf{\mathsf{A}}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Calculus

$\frac{dy}{dx}$	Derivative of y with respect to x
$\frac{\partial y}{\partial x}$	Partial derivative of y with respect to x
$\nabla_{\boldsymbol{x}} y$	Gradient of y with respect to $\boldsymbol{x}$
$\nabla_{\boldsymbol{X}} y$	Matrix derivatives of y with respect to $\boldsymbol{X}$
$\nabla_{\mathbf{\mathsf{X}}} y$	Tensor containing derivatives of y with respect to $\mathbf{\mathsf{X}}$
$\frac{\partial f}{\partial \boldsymbol{x}}$	Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{m \times n}$ of $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$
$\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ or $\boldsymbol{H}(f)(\boldsymbol{x})$	The Hessian matrix of f at input point $\boldsymbol{x}$
$\int f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral over the entire domain of $\boldsymbol{x}$
$\int_{\mathbb{S}} f(\boldsymbol{x}) d\boldsymbol{x}$	Definite integral with respect to $\boldsymbol{x}$ over the set $\mathbb{S}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x} \sim P}[f(x)]$ or $\mathbb{E} f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P \Vert Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

Table 2: MC Statistics

Group	MC	Random
Mean	297.00	256.30
SD	11.33	10.56
SEM	3.58	3.34
N	10	10

$f : \mathbb{A} \rightarrow \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\dfrac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\lvert\lvert \boldsymbol{x} \rvert\rvert_p$	L^p norm of $\boldsymbol{x}$
$\lvert\lvert \boldsymbol{x} \rvert\rvert$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise

D DATA

D.1 MC-INITIAL RUN

P value and statistical significance: The two-tailed P value is less than 0.0001 By conventional criteria, this difference is considered to be extremely statistically significant.

Confidence interval: The mean of Group One minus Group Two equals 40.70 95% confidence interval of this difference: From 30.41 to 50.99

Intermediate values used in calculations: t = 8.3077 df = 18 standard error of difference = 4.899

Raw Data: weights = [[-0.46031637, -1.02296217, -1.64597146, 0.50871499, 0.05907032, -0.04527117, -2.67590788, -1.78400492, -0.08667306, 0.48108891, 0.66066313, -1.57675411, -0.56494518, -0.07736412, -0.3257198, -0.65003209, -0.63740714, 0.44494984, -0.1545964, 0.67457139, 2.31472314, 0.8694452, -2.29173301, 0.52783125, -0.86950875, -1.77655688, -3.29970913, -0.242993, -1.57548922, -1.34238258, 0.36816378, -3.23065985, -0.07919411, -2.1089143, -3.12815169, -0.74580836, 0.98398675, -0.75271283, -0.81051661, -0.60567687, -3.42010519, -0.63186969, -2.02352157, -0.27534069, -0.28736574, -1.15836776, -3.28679005, -0.33767846, -0.41568405, 0.2782292, -1.23761129, -1.80559854]

MC won: [292, 307, 283, 286, 309, 311, 309, 296, 282, 295] times

Rand won: [248, 245, 242, 266, 255, 268, 273, 252, 251, 263] times

NN approx results:

Table 3: MC with NN Statistics

Group	w/NN	Random
Mean	298.00	257.00
SD	13.39	13.19
SEM	4.23	4.17
N	10	10

P value and statistical significance: The two-tailed P value is less than 0.0001 By conventional criteria, this difference is considered to be extremely statistically significant.

Confidence interval: The mean of Group One minus Group Two equals 41.00 95% confidence interval of this difference: From 28.51 to 53.49

Intermediate values used in calculations: t = 6.8975 df = 18 standard error of difference = 5.944

Monte Carlo won 298.0 times on average :: [290, 276, 313, 302, 298, 302, 276, 304, 306, 313]

Random won 257.0 times on average :: [252, 260, 255, 279, 255, 246, 242, 266, 275, 240]