Hidden Information, Teamwork, and Prediction in Trick-Taking Card Games

Hadi Elzayn
School of the Arts and Sciences
University of Pennsylvania
Pennsylvania, PA 19103
hadas@sas.upenn.edu

Mikhail Hayhoe
School of Engineering and Applied Sciences
University of Pennsylvania
Pennsylvania, PA 19103
mhayhoe@seas.upenn.edu

Harshat Kumar
School of Engineering and Applied Sciences
University of Pennsylvania
Pennsylvania, PA 19103
harshat@seas.upenn.edu

Mohammad Fereydounian
School of Engineering and Applied Sciences
University of Pennsylvania
Pennsylvania, PA 19103
mferey@seas.upenn.edu

Abstract

We highlight a class of card games which share several interesting features: hidden information, teamwork, and prediction as a crucial component. This family of games in question, known as "Whist" games, all consists of games of trick-taking, turn-based play, with team relationships of varying intensities, differing betting and scoring rules, and slight variations in mechanics. Using self-play, we have trained a DeepRL-style algorithm to bet and play *Four Hundred*, a flagship game in the family (*Hearts*, *Spades*, and *Bridge* are all related to varying degrees). Our algorithm reaches human-competitive performance, dominating all baselines it was tested against.

We believe this family of games provides an interesting testing ground for reinforcement learning algorithms because of its features; however, we are most interested in approaches to transfer insights across variations of games. We propose a meta-algorithm as follows: learn one game in the family or through self-play using a large number of games, then learn a cross-game invariance mapping from a small number of (human or self-played) games, and finally return the optimal policy of the new game with respect to the learned value function of the original game and the invariance mapping. We call this approach *invariance discovery* by analogy to image classification tasks, or *insight abstraction* with an eye towards its goal. We hope that this approach will result in more efficient training and perhaps more human-like play, and provide inspiration or machinery for other settings in which generalizing learning is a major goal.

Keywords: Reinforcement Learning, Self-Play, Games, Transfer Learning

Acknowledgements

We appreciate valuable feedback from Shivani Agarwal, Heejin Chloe Jeong, and Steven Chen.

1 Introduction

Inspired by the recent successes in the design of artificial intelligence to play games such as Backgammon, Chess, Poker, and Go with superhuman performance, we study a family of card games ("Whist" descendants) that provide a rich strategic environment for testing learning algorithms. These games share several interesting features, including hidden information, teamwork, and prediction as a crucial component. The Whist family consists of games of trick-taking, turn-based play, with team relationships of varying intensities, differing betting and scoring rules, and slight variations in mechanics. In work so far, we have focused on *Four Hundred*, a flagship of the game family, and design a reinforcement learning algorithm to play the game at a level competitive with humans. We describe our problem formulation, algorithms, and results in Section **??**. However, we are interested more in this family of games as a testing ground, and in particular we wish to explore the transfer of insights and strategies from one game into similar games. We discuss this question and our initial algorithm in Section **??**.

2 Solving *Four Hundred*

Rules of *Four Hundred*

;/lA deck of cards is distributed evenly (face down) to four players. Each player sits across from their teammate. No communication of hands is allowed. Before beginning play, players must bet an expected number of 'tricks' they plan to take over the 13-card hand. In each round after bets are placed, players take turns choosing a card to play from their hand in order, beginning with the player to take the previous trick. The suit of the first card played determines the 'lead suit', and all other players must play cards from that suit if they have any. The winner of the trick is the player with the highest card of the 'lead suit', or the highest card of the 'trump suit' if any were played. At the end of 13 rounds, each player's score is increased by their bet if they meet or exceed it, but decreased by their bet if they fail to meet it. The game is over once one team has a player with 41 or more points, and the other player has positive points.

Learning Problem *Find an optimal betting policy $\beta^{i,*}$ and playing policy $\pi_t^{i,*}$ to maximize expected reward for player i given each other and the play of the others:*

$$\beta^{i,*} = \arg\max_{\beta \in \mathcal{B}} E\left[\sum_{t=0}^{13} R\left(\beta(\mathcal{H}_0^i), \pi_t^{i,*}(\mathcal{H}_t^i)\right)\right], \qquad \pi_t^{i,*} = \arg\max_{c \in \mathcal{H}_t^i} E\left[R(\beta^{i,*}(\mathcal{H}_0^i), c) \mid S_t^i\right].$$

where $\mathcal{H}_0^i$ is player i's starting hand, $\mathcal{H}_t^i$ is their hand after trick $t \in \{1, \ldots, 13\}$, and S_t^i is the information known to player i about the state of the game up until trick t.

BETTING: While both betting and playing may be viewed from a reinforcement learning perspective, we chose to view betting as a supervised learning problem. That is, given some particular strategy and initial hand, the player can expect to win some number of tricks (where randomness comes from the distribution of cards across opponents' hands as well as variation in player strategies, stochastic and otherwise). Given observed games under a given strategy and initial hand compositions, a model that predicts tricks taken by the end of the round may serve as a good bet model.

Thus, the data we generate during games in the form of initial hands can serve as input data, while the total number of tricks won that round functions as the label. We implement a neural network for regression using this data. The input is a 4×13 binary matrix with exactly 13 non-zero elements representing which cards are in the player's hand. Each column of the matrix represents a value (e.g. 7 or *Queen*), while each row represents a suit. The first suit is reserved for the trump suit, which is higher in value than all other suits. The output of the neural network is a real number, which we then map to the closest integer between 2 and 13 (since bets of 0 and 1 are not allowed). We define the loss as the squared difference between the score received and the best possible:

$$\ell_{\text{bet}}(y, \hat{y}) = (y - \text{sign}(\hat{y} - y) \cdot \hat{y})^2 = \begin{cases} (y - \hat{y})^2, & \text{if } y \geq \hat{y}, \\ (y + \hat{y})^2, & \text{otherwise,} \end{cases}$$

where $\hat{y}$ is the *bet*, and y is the *tricks won*. Motivated by the game's scoring rules, this loss function is asymmetric - it penalizes more for bets which are higher than the tricks obtained. Therefore our goal in this

supervised learning problem is to learn a relationship between the initial cards dealt to each player and the number of tricks that player won at the end of the round. In this regard, the actual play of the game greatly affects the number of tricks expected to win.

CARD PLAY: The second component of the game is the playing of cards, wherein we consider a reinforcement learning approach to solve the problem. We capture the state of the game by three 4×13 matrices and two 1×4 vectors in the following way. The first matrix represents the order history of the game, which begins as a zero matrix. Each card played is represented in the order history by an integer between 1 and 52. For example, in the 5th trick of the game, the card played by the third player will have a 23 in the corresponding location (5th trick $\times$ 4 cards per trick + 3rd card played = 23). The second matrix is for player history, which is also initially a zero matrix. As each card is played, its location will be filled by a number denoting the ID of the player. Continuing our example, if the card above was played by player 2, then a 2 will be put in the corresponding location. Representing order and player history as a matrix in this fashion was inspired by the state representation of AlphaGo. The final matrix is the player hand, which is similar to the input for the betting neural network, and indeed is identical at the beginning of the game. As the game continues, whenever a card is played, the 1 indicating the presence of a card in the hand becomes a zero. Returning to our example, the matrix after the fifth round will have 8 (13 initial cards - 5 cards played) non-zero elements. The first 1×4 vector contains the bets that each player made at the beginning of the round. The second vector contains the tricks that each player has won so far in the round.

Given the states, we define a reward at the end of each round by

$$\text{Reward} = \begin{cases} \text{bet}, & \text{if tricks} \geq \text{bet}, \\ -\text{bet}, & \text{otherwise.} \end{cases}$$

The state size is combinatorially large, and hence we do not consider tabular reinforcement learning solution methods but rather function approximation with neural nets. Ultimately, given the possible actions available to the player, we will evaluate the function at each of the potential states and choose the action which enters the state with the highest value. Our approach is informed by the classical Q-learning approach, where the value function is updated by

$$V^{t+1}(s) = \max_a \sum_{s'} p(s'|s,a)(r(s,a,s') + \gamma V^t(s')),$$

where s is the state, a is an action, s' is the state after taking action a from state s, and $\gamma \in [0,1)$ is the discount factor. We consider a mild adaptation, similar to [**?**], in which the reward is provided as a label to each observed state, and the neural net Q-update occurs in batches. Given that reward is observed at the end of the round, for trick $t \in \{1, \ldots, 13\}$, we assign a reward to that state s by

$$\text{Value}(s) = \gamma^{13-t} \cdot (\text{Reward}_{\text{Team member 1}} + \text{Reward}_{\text{Team member 2}}) + \mathbf{1}\{\text{Team won the trick}\}.$$

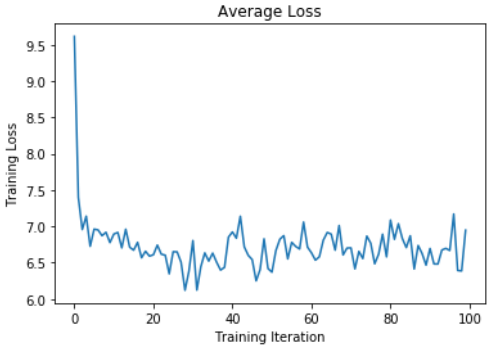
We include the **1** term for reward shaping [**?**] as it is a favorable outcome which should help increase convergence. Once we have assigned a label to each of the states in terms of the value defined above, we use a neural network for regression to map each state to the appropriate value.

BASELINES: All data was generated from a game simulator developed from scratch. Three baselines were created to compare against: the first baseline is **Random Play-Random Bet**. Random bet selects a random value from $\{2, 3, 4, 5\}$ during the betting stage, and in each round chooses a random card to play from the set of valid cards. The second baseline is **Greedy Play-Model Bet**. During play, greedy simply selects the highest card in its hand from the set of valid cards, without consideration of its partner. Betting for greedy uses a neural net trained on 100,000 games of 4 greedy players with the same architecture as *Over400*, as described earlier. For the final baseline, **Heuristic Play-Model Bet**, a heuristic strategy was defined based on human knowledge of the game. This heuristic takes into account knowledge of the team member's actions as well as opponents' actions. While a betting heuristic was also defined, the betting for the heuristic baseline is also given by a model which was trained, in the same way as greedy, on 100,000 games of 4 heuristic players.

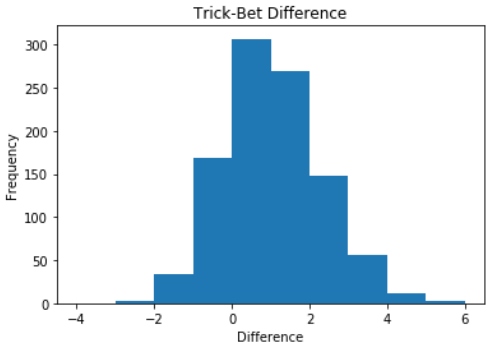
2.1 Results

To understand the performance of the betting model, we consider the metrics of loss and Trick-bet difference. The noise observed is due to the betting and playing strategies being constantly updated in tandem.

Figure **??** shows the Trick-bet difference metric. As we can see, the highest peaks of the histogram occur at Tricks - Bet $\in \{0, 1, 2\}$. This means that most of the bets are slightly less than the number of actual tricks taken, indicating that the players are betting conservatively.



(a) Loss function for the Betting Neural Network for regression

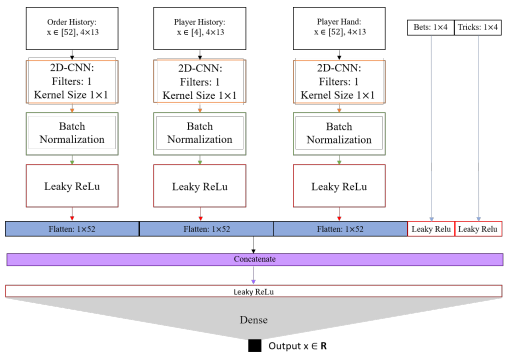


(b) Histogram showing difference between tricks won and bet made

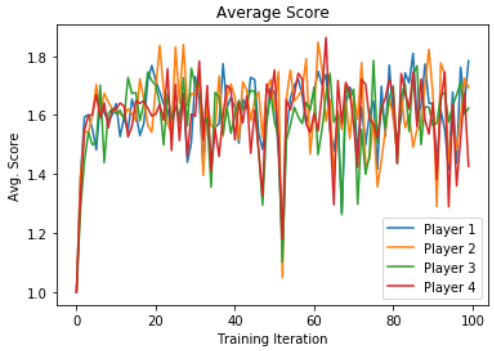
Figure 1: Betting Network Performance

To understand the performance of the card play network, we consider the average score shown in Figure **??**. As expected, the average score of all four players increases as the number of training iterations increase. It is important to note that the game is learning how to play relative to the other players, in that it understands the relationship between its team member and its opponents relative to its position. Finally, we consider Table 1 which shows the performance of our Program against the baselines. Our Heuristic algorithm was able to defeat Random and Greedy Baselines 100 - 0, as did *Over400*. Even more impressively, *Over400* **beat the heuristic baseline 89.5 % of the time**.

Human players also squared off against the network to get a sense as to how *Over400* was playing. The AI understood to win tricks by playing high cards of the lead suit and also to play a trump card when the lead suit had run out in its hand. Interestingly, it seemed that one of the AI players was consistently betting higher and performing better than the other team member. However, *Over400* would occasionally play high cards which were not capable of winning since they were not the lead or trump suit. We imagine this is due to the difficulty in learning the concept of the lead suit, which may have been accomplished with more samples of training data or with a deeper neural net structure.



(a) Neural Network Architecture for Card Play



(b) Average score showing the performance of the self play NN model

Figure 2: Card Play Network and Performance

	Random	Greedy	Heuristic	*Over400*
Heuristic Win %	100	100	-	10.5
Over400 Win %	100	100	89.5	-

Table 1: Percentage of wins against the baselines over 200 games.

3 Future Work

There are many games that can fit into our problem formulation with minor tweaking; there are over 30 varieties of Whist games with major or minor variations in rules. In future work, we will examine whether and how, by learning one game, we can learn them all.

For example, *Spades* is nearly the same as *400*: the objective is to take tricks, but Spades is the trump suit (rather than Hearts), and there is usually no betting or teams. *Tarneeb* is nearly the same as *400*, but the trump suit is chosen by players (and the betting is more complicated). Finally, *Hearts* has the same mechanics as *Spades*, but the goal is to *avoid* taking tricks.

A policy trained for *Four Hundred* will likely fail to produce good results on any of these other games. However, it is clear that the similarity of structure in these games *should* mean a learner which is good at one game should perform well when playing others. An analogy can be drawn to classification: playing *Spades* well and *Hearts* poorly feels similar to classifying objects with high accuracy but failing to do so if they are rotated and translated. Overcoming this problem in image classification required, in some sense, focusing on features that were 'invariant' to translation and rotation; in our case, there is some 'invariance' about the relative ordering of the cards. We thus call our proposed approach *invariance discovery*, or *insight abstraction*.

Let S be the set of observed states, V_G be the value of each state under game G, and $V_{G'}$ be the value of the state under game G'. Recall that we estimate V_G to be able to return the following policy:

$$\pi_G = \text{argmax}_a \mathbb{E}_{s' \sim P(s,a)} \left[V(s') \right].$$

In order to avoid retraining for G', we would like to learn the *invariance mapping* χ in order to produce a policy $\pi_{G'}$ for the game G':

$$\chi : V_G(s) \mapsto V_{G'}(s) \qquad \Rightarrow \qquad \pi_{G'} = \text{argmax}_a \mathbb{E}_{s' \sim P(s,a)} \chi(V(s'))$$

Such a mapping exists whenever the value functions do, but learning this function from noisy samples may be hard. Indeed, in full generality it may be impossible without exponentially many samples, since we may not see the same states across games. However, we may make this problem tractable by restricting χ to live in some class of functions Ω. For example, χ could be a linear function; this may approximate the truth when mapping *Spades* to *Hearts*, since having high cards is good in *Spades* but bad in *Hearts*. The meta-algorithm would be roughly as follows in Algorithm **??**.

Algorithm 1 Value Correspondence

Require: Games G, G', T_{many}, T_{few}
1: Learn a policy π_G and value function V_G via self-play of G over T_{many} rounds.
2: Learn a policy π'_G and value function V'_G via self-play of G' over T_{few} rounds.
3: Learn $\chi : V_G(s) \to V_{G'}(s)$ from rounds $1 : T_{\text{few}}$ of G and $1 : T_{\text{few}}$ of G'.
4: Return $\pi(s) = \text{argmax}\mathbb{E}\chi(V_G(s))$.

References

[1] V. Mnih et al. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[2] M. Riedmiller. Neural fitted q iteration – first experiences with a data efficient neural reinforcement learning method. *European conference on Machine Learning*, 05:317–328, 2005.