Learning Game Playing Strategy for Durak

Sammy Nguyen (smnguyen): CS221, CS229; Narek Tovmasyan (ntarmen1): CS229

Introduction

Durak is a card game played with a 36-card deck (6 through Ace from a regular 52-card deck), where rounds are taken with one player "attacking" and the other "defending". The objective of the game is to get rid of all the cards on one's hand. To reduce complexity, we consider the case of two-player Durak. The first player to have no cards left is declared the winner. The game starts by shuffling the full deck of cards and dealing out 6 cards each to the 2 players. Then the top card on the remaining deck, unlike the rest, is revealed and put face up under this common deck. The suit of this visible card thus becomes the trump suit. However, this differentiated card is still a part of the deck, from which the players will draw cards during the actual gameplay (including this revealed card, in the end).

The player who has the lowest trump card goes first, by initiating the first attack of the game. The defender attempts to beat any subsequent attacking card by playing a higher card of the same suit as the top card on the table or playing a trump card of any rank.[1] The attacker may maintain the attack by playing only cards of the same rank as those already on the table. If the defender is unable to play a superior card or chooses to not play a card, they must pick up all the cards on the table -- including all the cards the current attacker pitched in -- and add them to their hand. In either case, the defender thus fails to discontinue the defense. Accordingly, the attack is considered to have been successful and thus the "right" to attack is *not* passed from the current attacker to defender. So the attacker begins a brand-new attack at the next round. Otherwise, if the attacker cannot continue attacking (or is unwilling to), this immediately ends the current round and all played cards on the table are discarded. The defender is said to have "triumphed" defending at the previous round and opens the next turn as the new round's attacker.

At the end of each round, if any player has less than six cards in their hand, cards are drawn from the deck until the player has six cards again or until all cards in the deck are gone. The attacker of the completed round always draws the required number of cards first; the last round's defender draws any needed cards second. The order in which this is carried out is strategically important, because the last card in the deck is a trump. By the rules of the game, a player is not allowed to see their opponent's hand, and no players may examine the discard pile or current deck at any point.

When the deck is emptied, players keep going without drawing cards. This phase constitutes the endgame where cards are either in the player's hand, the opponent's hand, the discard pile, or on the table. Through card-counting, a player can determine with 100% certainty the cards in his opponent's hand. At this point, the first player to get rid of their cards becomes the winner of the game.

[1] A trump card beats any non-trump card regardless of its denomination/rank (e.g., a trump 6 or 7 beats any of the three non-trump aces).

In this project, we use reinforcement learning techniques to develop an AI that will play competitively against a player that plays with the simple yet quite powerful policy of playing its lowest ranked non-trump card, or its lowest ranked trump card if no non-trump cards are available. The vast state space of Durak will make it impossible to run value iteration, so we featurize each state and run temporal difference (TD) learning to learn weights that will help approximate the value for each state. During the endgame, we also use minimax to help determine the action to take at each turn, assuming that the player's opponent will always take their optimal action. Similar to the agents in Tesauro's TD-Gammon, we train our agents by having them play games against themselves.

Methods

Data and Simulation:

Using Python, we created a console program to run a game of Durak, and several baseline agent classes: a `HumanAgent` which prompts the user to input decisions into the console, a `RandomAgent` which chooses actions at random, and a `SimpleAgent` that executes the policy of choosing the lowest valued card possible to play. To provide a baseline for the `MinimaxAgent`, we also created a `SimpleHandicappedAgent`, which executes the same strategy as the `SimpleAgent` until the endgame, where it chooses actions at random. The simulator logs data about the game state for each player, i.e. what each player knows about the state of the game at his turn. Specifically, the simulator records at each player's turn, the cards in the player's hand, the current number of cards remaining in the deck, the cards that have been discarded, the cards on the table, the number of cards the opponent has, the cards it knows its opponent has, and cards that the player has not yet seen.

To extract features for learning, we examine a game state recorded by the simulator, and calculate for the player's hand: the average rank of each suit, the number of cards per rank, the number of cards per suit, the number of trump cards, and how many more cards the player has compared to the opponent. Additionally, from the cards that we know the opponent has, we determine a minimum bound on how many cards the opponent can respond with. These features were used for the reflex agent, and one version of the minimax agent. One problem with this set of features is that it does not compute the relative strength of a player's hand compared to his opponent's hand. We cannot reliably compare the strength of the player's hand until the endgame without using probabilistic guesses through the *hypergeometric distribution* (selection without replacement), which can be very noisy -- an extreme example of this would be the first round of the game, where the players know nothing about which cards their opponents have. In general, the game environment is not fully observable, so decision making becomes more difficult, since an action will only be good with some probability.

At the endgame, however, a player can know with 100% certainty which cards the opponent has. Thus, for the minimax agent, we can compare the strengths of the players' hands and calculate the following features: the difference in average ranks per suit, the difference in number of cards per rank, the difference in cards per suit, the difference in number of trump cards, how many more cards the player has compared to the opponent, and indicator functions on whether a player has the maximum card for each suit. Finally, instead of a minimum bound, we can compute the exact number of cards that are valid for the opponent to use on his turn.

AI Development:

To create game-playing agents for Durak, we used reinforcement learning and the temporal difference (TD) learning framework to learn the values for each state. TD learning uses sets of weights, which are learned, that help approximate the value function.The general TD learning weight update is as follows: given a state s, an action a, a reward r, and a successor state s',

$$w^{(t+1)} = w^{(t)} - \eta[V(s; w^{(t)}) \; - \; (r \; + \; \gamma V(s'; w^{(t)}))]\nabla_{w^{(t)}} V(s; w^{(t)})$$

Because the state space for Durak is so large, it is impossible to use value iteration to compute the value for each state. Therefore, TD learning allows us to use an approximation of the value function, parameterized by the weights. For Durak, we chose to use a logistic function to approximate the value of each state -- this can be interpreted as the probability of winning the game given a specific state.

Because of the rules of Durak, a certain hand that might be strong for attacking might be weak for defending. An extreme example of this would be a hand with many low ranking cards, e.g. all four 6s, an 8, and a 9. This hand would be good for attacking, since there are many duplicates, but weak for defending, since the cards are all low in rank. To solve this problem, we use two sets of weights which are used for determining the best attacking action and the best defending action. For a state diagram as follows ('a' denotes attack, 'd' denotes defense), we have the following TD updates:

$$\ldots s_a{}^{(t)} \rightarrow a_a{}^{(t)} \rightarrow r_d{}^{(t-1)} \rightarrow s_d{}^{(t)} \rightarrow a_d{}^{(t)} \rightarrow r_a{}^{(t)} \rightarrow s_a{}^{(t+1)} \ldots$$

$$w_a{}^{(t+1)} = w_a{}^{(t)} \; - \eta[V(s_a{}^{(t)}; w_a{}^{(t)}) \; - \; (r_a{}^{(t)} + \; \gamma V(s_a{}^{(t+1)}; w_a{}^{(t)}))]\nabla_{w_a{}^{(t)}} V(s_a{}^{(t)}; w_a{}^{(t)})$$

$$w_d{}^{(t+1)} = w_d{}^{(t)} \; - \eta[V(s_d{}^{(t)}; w_d{}^{(t)}) \; - \; (r_d{}^{(t)} + \; \gamma V(s_d{}^{(t+1)}; w_d{}^{(t)}))]\nabla_{w_d{}^{(t)}} V(s_d{}^{(t)}; w_d{}^{(t)})$$

Using the TD learning framework, we trained three different agents: a reflex agent and a minimax agent using features based solely on the player's hand, and another minimax agent based on features involving the strength of the player's hand in relation to the opponent's hand. The reflex agent determines an action to take relatively quickly, since it only examines the immediate state of the game. The minimax agents work by following the strategy of the `SimpleAgent` until the endgame, at which point it determines the best move to take to maximize its chance of winning, given that the opponent will try to maximize its own chance of winning. This decision occurs by actually playing through the game for each possible action and choosing an action that results in win. In practice, doing a complete search takes too long, even at the endgame, so exploration only occurs to some specified depth -- we chose 4 plies (turns of play) -- at which point the value of the resulting state is approximated.

Results

As a baseline comparison, `RandomAgent` can only beat `SimpleAgent` approximately 1% of the time. `SimpleHandicappedAgent` does better, but only beats `SimpleAgent` approximately 15% of the time. As might be expected, each agent wins 50% of the time when playing itself. To evaluate each agent, we train each agent for at least 500 games, and evaluate the performance of the learned weights every 50 games by having the agent play 100 games against the random and simple agents.

Reflex Agent

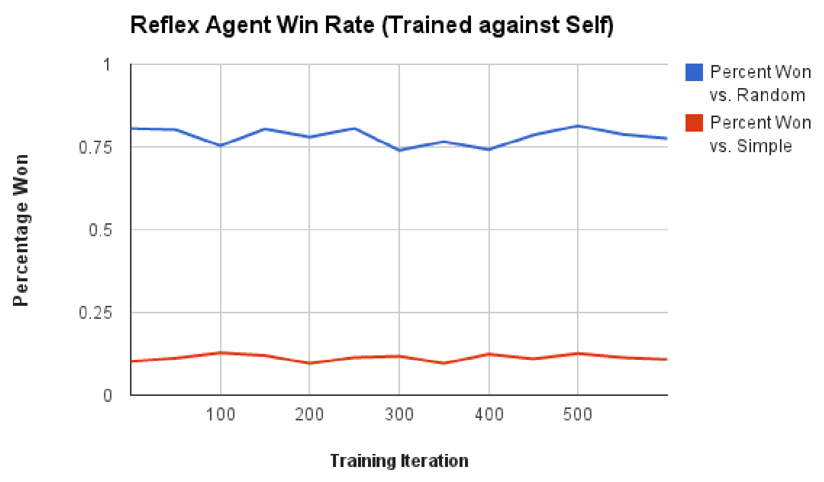

Figure 1. ReflexAgent was trained against itself over 600 games. After every 50 games, the learned weights would be evaluated by having ReflexAgent play RandomAgent and SimpleAgent for 100 games. The learned weights for ReflexAgent result in wins around 80% of the time against RandomAgent, and wins 10% of the time against SimpleAgent. No significant improvement in success was detected after training.

Minimax Agent v1

Features for this minimax agent were based mainly on cards in the player's hand.

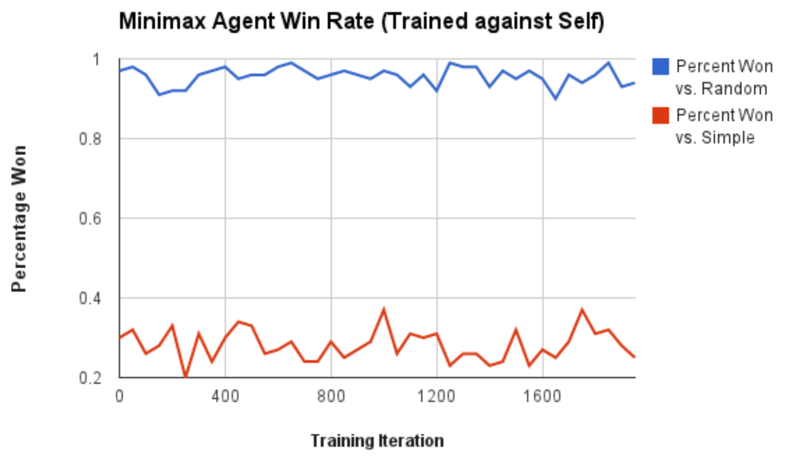

Figure 2. MinimaxAgent v1 was trained against itself over 2000 games. After every 50 games, the learned weights would be evaluated by having ReflexAgent play RandomAgent and SimpleAgent for 100 games. The best learned weights for ReflexAgent result in wins around 96% of the time against RandomAgent, and wins 38% of the time against SimpleAgent. No significant improvement in success was detected after training.

Minimax Agent v2

Features for this agent were based on comparing the player's hand to the opponent's hand.

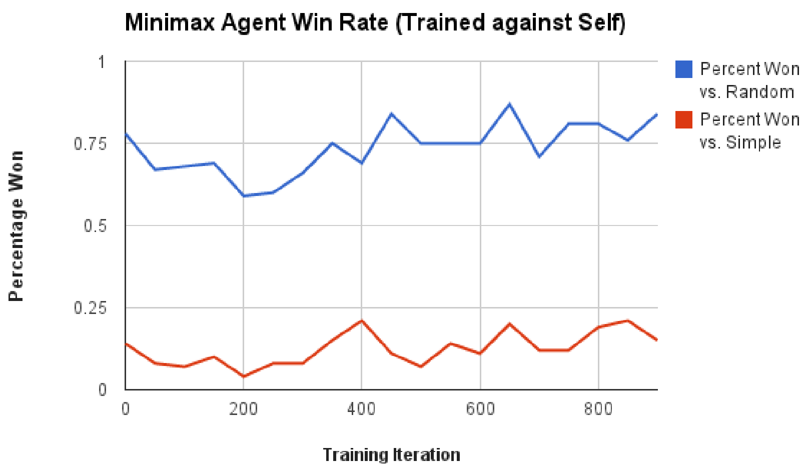

Figure 3. MinimaxAgent v2 was trained against itself over 900 games. After every 50 games, the learned weights would be evaluated by having ReflexAgent play RandomAgent and SimpleAgent for 100 games. The best learned weights for ReflexAgent result in wins around 85% of the time against RandomAgent, and wins 21% of the time against SimpleAgent.

Conclusions

Although all trained agents do significantly better than their random counterparts (`Random` vs. `Reflex`, `SimpleHandicapped` vs. `Minimax`), they still do not reach or exceed the performance of `SimpleAgent`. Additionally, the performance of each trained agent does not seem to visibly improve over time. This is most likely because the set of features we selected was either not large enough, or because the features don't accurately represent the information needed to make a good decision on which action to take. For instance, comparing Minimax v1 with Minimax v2, it appears that features based mainly on cards in the player's hand do better than features comparing strengths of the player's hand and the opponent's hand. However, Minimax v2 does seem to be improving, albeit slowly, so training for more iterations may be a useful improvement as well. The main impediment to increasing the number of training iterations was speed, so future work would either necessitate increasing speed, or using a reinforcement learning library already optimized for speed.

We have been excited and challenged by this project, and would like to capitalize on the great opportunity this project has afforded us and extend the work done in this paper in the future.

References

Tesauro, G. (1995). Temporal difference learning and TD-Gammon.*Communications of the ACM*, *38*(3), 58-68.

Ng, A. (2013). CS229 Lecture notes: Reinforcement Learning and Control. Stanford University.