Introduction

- Hearthstone is a popular turn based Trading Card Game produced by Blizzard Entertainment.
- Hearthstone has a lot of factors at play in any given game, and we wished to find the most optimal features to capture game state.
- Pro players will guess at what cards their opponent has.
- Our goal is to write a card predictor that will predict our opponent's deck based on what has been played so far, as well as a feature extractor that we can use to evaluate game states with our own minimax game player.

Data Collection

- Monte Carlo simulation used to generate episodes, where each episode was a game.
- Rewards given at end of game, either win or loss.
- Generated games by simulating random agent playing against itself.
- Data about popular decks scraped from hsreplays.com.

Methods

Setup:
Used HearthSim/Fireplace framework to simulate the Hearthstone environment.

TD Learning
- Used TD-learning with Monte Carlo simulation to learn weights for our linear minimax evaluation function.
- Used $\epsilon$-greedy policy with $\epsilon = .75$ to explore most of state space.
- Update formula:
$$ w \leftarrow w - \eta \left[ \frac{1}{k} \sum_{i=1}^{k} \left( r_i + \gamma V(s') - V(s) \right) \right] $$

Card Prediction
- Used card predictor to find the k-nearest decks given the cards played by the opponent throughout the course of the game.
- As the game progresses, the predictor updates the prediction, learning which decks are the most probable.
- When simulating opponent's turn for minimax, the card predictor was used to figure out all their possible sequences of actions from a given state, then utilized beam-search to limit branching factor.

Feature Extraction
- Started with sets of high level features to capture overall state.
- Ignored details such as interactions between cards.
- Found the optimal set of features by utilizing backward search starting with set of features that were initialized from existing domain knowledge.
- Trained and tested by running 200 simulated games (each game is around 10 turns) against an aggressive agent.

Evaluation

Deck Predictions
- The best set of features we found was a set of 7 features, each of which is highly valued during human play.
- Calculated deck prediction percentages by averaging over 80% of most frequently played decks, repeated 100 times for all decks and averaged the classification accuracy.
- With the most relevant features, our minimax player achieved a peak win rate of 81.5% against an aggressive agent.
- To test our mulligan, we kept all other factors equal, and one agent chose to mulligan random cards, while the other used the mulligan weights learned.
- The agent we played against is a close approximation of real world play, as aggressive decks are the most common that people play.

Conclusion and Future Work

- Our AI performed better than we expected. We suspect that this may be due to feature extraction pruning all the unnecessary features. That set up td-learning to learn more accurate weights for the relevant features, resulting in enhanced performance.
- Our card predictor accurately predicted the cards the opponent would play, leading to more accurate evaluation of game states and making overall better decisions.
- For future work, we’d like to focus on:
  - Try a neural network for TD-Learning.
  - Try a neural network for Softmax Card Prediction.
  - Interact with live game to test against real players.

Mulligan
- Used mini-batch gradient descent to determine which cards to replace and resample from the deck.
- Trained by simulating games against itself repeatedly.
- If won, the weights of the starting cards are increased by the learning rate.
- If loss, the weights of the starting cards are decreased by the learning rate.
- For faster convergence, simulated games against same deck, so that both players’ starting cards used as data for gradient descent.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>$\eta$</th>
<th>$\gamma$</th>
<th>$r_{\text{if loss}}$</th>
<th>$r_{\text{if won}}$</th>
<th>$r_{\text{else}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>0.001</td>
<td>0.9</td>
<td>-100</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>