spAIdes Progress Report

Drew Gregory (drewgreg), Ryan Tolsma (rtolsma), Griffin Kardos (gkardos),
Kendrick Shen (kshen6)

November 2018

1 Introduction

Our project aims to create an agent that plays the cards game "Spades" well. We have approached the problem with a Q-Learning solution.

2 States

The state of the game is composed of the following: the cards the player has in their hand, the claimed cards each player has, the bids each player made for the round, the bags each player has, and the center pile. An example state is as follows:

PlayerHands: [7♣, 10♠, 2♡, A♡, 6♡, 10♢, 3♣, 2♣, 5♣, J♣, Q♠, 4♣] (cards of player)
PlayerClaimedCards: [set(), Q♡, K♡, 3♡, J♡, set(), set()] (claimed cards in order moving clockwise around table starting at player)
PlayerBids: [0, 3, 3, 2] (bids in order moving clockwise around table starting at player)
PlayerBags: [0, 0, 0, 0] (bags in order moving clockwise around table starting at player)
Pile: [3♡, J♡, Q♡, K♡] (cards placed in order of rotation)
We do not include the players' scores from previous rounds into our state, as these scores should not affect the player's strategy.

3 Actions

At any given rotation, the available actions consist of the legal cards a Player can play from his hand. Refer to the our Proposal for gameplay structure.

4 Rewards

With any Q-Learning Model, the structure of the reward feedback completely determines the policy and evaluation functions that the model tries to optimize. With this in mind, we desire to structure our reward feedback to ultimately correspond directly with our scoring system for Spades. More formally, for any scoring function Score(s), we desire to structure a reward function $r(s, a)$ such that the policy $\hat{\pi}_{\text{opt}}$ generated by the Q-Evaluation Function $\hat{Q}_{\text{opt}}(s, a, r(s, a), s')$ will maximize Score(s).

Very simply, we can reason that structuring our reward function to be zero for each action, and then be equal to the final score as the feedback reward in the terminal state would satisfy the mathematical constraints mentioned. However, this has the problematic effect of being difficult to train models with.

In developing our model, we have approached the training process in steps reflecting the ability to learn subproblems of our primary objective. Below we discuss a few subproblems and approaches to solving them by structuring new reward functions.

4.1 Trick Winning

In attempting to train reliable models, we first study the problem of training a Q-Learning model to optimize for winning tricks. In this structure, we defined the scoring to be given by

$$\text{Score}(s) = \#\text{of tricks}$$

After each rotation, where each player finishes playing their cards, we give zero rewards to the losers, and a reward of 1 for the player that won the trick. Implementing this reward function, and applying it to our Deep Q-Learning Model, lead to some reasonably good results. In general, our model achieved roughly 20% better than our Baseline comparisons, and on average over 10000 games achieved 3.5 tricks per round. While it is nothing remarkable, it is notable to see that the expected number of tricks in a round $E[t] = 134 = 3.25$ is lower than our model's performance, indicating some level of achieved proficiency. Do to the stochastic nature of multiplayer incomplete information games, it is difficult to judge whether this denotes any significant amount of generalized learning. Please refer to Figure 3 of the appendix.

4.2 Combined Trick and Bidding

Please refer to the Project Proposal paper for scoring guidelines. In order to construct a reward function that is conducive to consistent training and improvement, we opted to construct a reward function defined as follows:

$$\text{r(s,a)} = \begin{cases} 1 & \text{Player Won and Is Not Overtricking} \\ -0.5 & \text{Player Won and Is Overtricking} \\ \frac{\min(\text{Player Tricks Won} - \text{Player Bid}, 0)}{13 - \#\text{Tricks Left} + 1} & \text{Player Lost} \end{cases}$$

However, this reward function did not accurately relate to the Scoring function well enough for our model to perform adequately. Rather, the model was incentivized too much to play higher cards earlier on in the rounds which produced faulty short term strategies. Below, we show a graph of scores relating the model performance over many games using this setup. Please refer to Figure 4 of the Appendix.

4.3 Pure Terminal State Reward

Due to our current difficulties in achieving an accurate–but trainable– reward function, we attempted to resort to the mathematically pure reward function described earlier, that only describes the terminal state scores. However, this proved to be extremely difficult to train as expected, resulting in too few states reliably being affected by the weight updates through gradient descent. We graph the scoring over time (in comparison to several hardcode baseline strategies) as well as the loss function. The loss function is notably zero almost everywhere, as too few states are being explored relative to the state space.

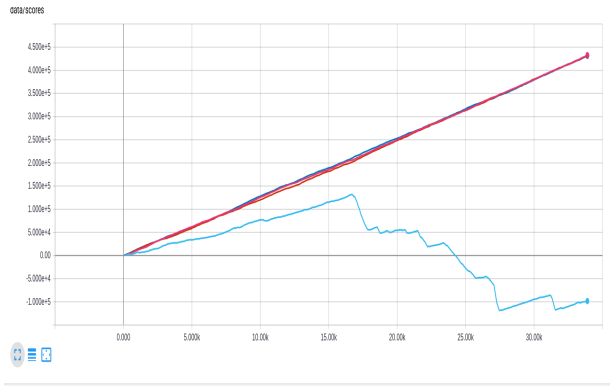

Figure 1: A Comparison of Cumulative Scores over repeated gameplay

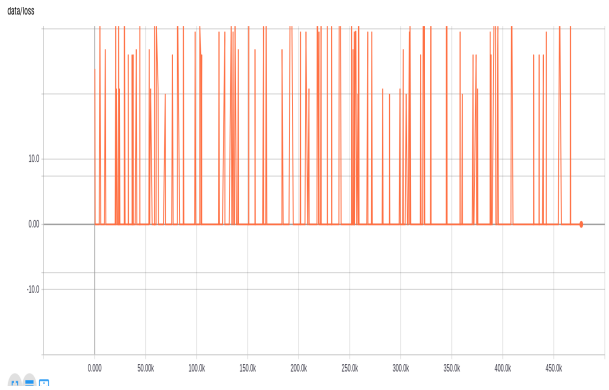

Figure 2: Graph of Loss Function

By analyzing the graphs of the Loss and Comparison of scores (Figures 3,4 of the Appendix), we realized that the reason why the model performed reasonably for so long was due to relatively consistent bidding of 3, which is near the expected value of tricks won. However, no distinguishable learning appeared to occur with regards to actual gameplay.

5 The Evaluator Function

The crux of our q-learning algorithm is our evaluator function $Q_{opt}(s, a)$. We decided to have our function learn our evaluator function via a neural network.

5.1 Features

Our features for our algorithm are as follows: The feature is one giant vector.

1. The first 52 indicators are for the presence of cards in a player's hand, 1 for each card.
2. The next 52 indicators are for whether a card is claimed or not in a previous rotation.
3. The next four components represent scalars for the number of bids for each player (in clockwise rotation order, starting at the player).
4. We then have another list of 52 variables for our pile. Each index represents a card. If a card is not in the pile, the value at the corresponding index is 0. If a card is the first card in the pile, the value at the corresponding index is 1. If a card is the second card in the pile, the value at the corresponding index is 2. If a card is the third card in the pile, the value at the corresponding index is 3.

5. The next four components represent scalars for the number of tricks each player (in clockwise rotation order, starting at the player) has accumulated so far this round.

6. The next four components are the number of bags for each player, organized in the same fashion as the number of tricks for each player.

7. Lastly, we have another list of length 52 for our chosen card. Each element is an indicator for whether you choose to play a card or not. If we're in the bidding stage, this entire list is comprised of 0's.

The total length of this vector is 220.

5.2 The Neural Network

The Neural Network has been the most finicky and conceptually challenging element of our project. We originally used a linear neural network. This performed poorly, so we added some relu layers and convolutional layers.

6 A Detailed Example/Run-Through

During the bidding round, our model proceeds via the following:

1. With probability 0.05 (explore probability), the model bids a random value between 0 and 12, inclusive.

2. Otherwise, the model determines a best bid action via its getQ() function, based on the neural network's model.predict().

Get q values for each possible bids and choose bid for highest q-value: bid Q-values: -51.87 -52.41 -52.73 -53.05 -53.40 -53.80 -54.23 -54.67 -55.11 -55.55 -55.99 -56.43 -56.87 -57.32 BID 0 , Q-value: -51.87419128417969
During a playing round, the model proceeds as follows:

1. With a probability 0.01 (explore probability), the model choose a random valid card to play.

2. Otherwise, the model evaluates the predicted Q-value for all valid cards, playing the card that maximizes the predicted final score.

3. At the end of the round, the model incorporates feedback into the evaluator function via the neural network's model.update().

Sample q-values and actions:
score:tensor([-26.4551])action:Q♢
score:tensor([-26.4962])action:3♢
score:tensor([-26.4780])action:J♡
score:tensor([-26.4506])action:10♣
score:tensor([-26.4749])action:9♣
score:tensor([-26.4604])action:8♢
score:tensor([-26.4719])action:K♢
score:tensor([-26.4536])action:6♢
chosen action and score: 10♣ tensor([-26.4506])

7 Appendix

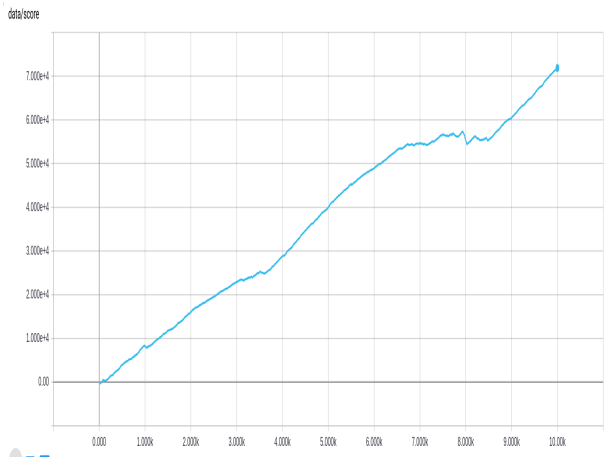

Figure 3: Cumulative Scores using Trick Optimizer

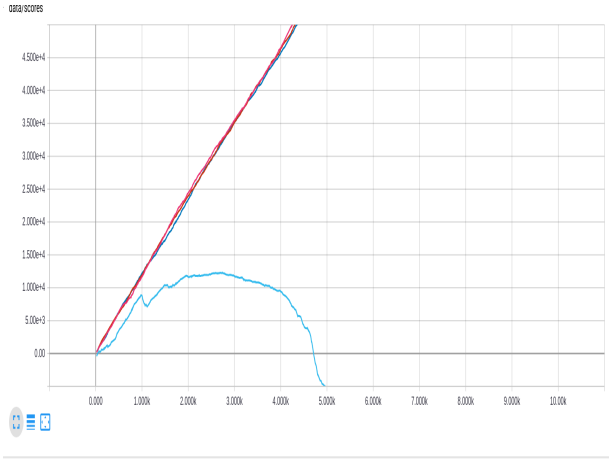

Figure 4: A Comparison of Cumulative Scores over repeated gameplay