Reinforcement Learning Applied to Cribbage

Danielle Cloutier

December 13, 2022

1 Introduction

Cribbage is a card game, normally played between two players which is split into multiple phases of play[1]. In this project, I attempted to create a neural network model that uses Deep Q-Learning to learn how to play the discard phase of the game. This decision is because the game's different phases essentially require completely different decision processes, which would likely require two separate agents to be trained somewhat in tandem and would require far more computational power than I possess to train. This network was trained in a multitude of different ways to verify certain methods that have been used by others not only for cribbage, but also for other games.

1.1 Reinforcement Learning and Deep Q-Learning

Reinforcement learning is a paradigm of machine learning which deals with learning in sequential decision making problems where there is limited feedback. This is done by imitating a Markov Decision Process. This is a process where an agent chooses actions on a state space based on a policy to optimize the amount of reward that the agent receives[2]. This policy is often depicted using a table, often called a Q-Table, and is then continually updated so the agent can make better decisions, using a Bellman Equation [3]:

$Q(s,a) = (1-\alpha)Q(s,a) + \alpha * (r_t + \gamma * max_a(Q(s',a)))$

where α is the learning rate and γ is a discount rate, so the agent learns to more heavily emphasize current rewards over future rewards. We also consider s and s' as the current and later states, with a being the action on that state. We also encode a reward, r_t, in this equation, which is the reward that the agent will receive from choosing that action. This equation is then recursively iterated, until the agent achieves an optimal policy. This process where we store these values in a Q-Table which is initially randomized is called Q-Learning[2].

One issue with this method is that as the state and action spaces enlarge, it requires significantly more memory to store these significantly more massive tables. This is especially an issue in my specific case, where a player is dealt 6 cards from a 52 card deck, meaning there are $\binom{52}{6}$ different hands, or more specifically, 20,358,520 possible hand states and 15 actions, which will be defined in more detail later. When it comes to applying the Bellman Equation over this massive table, it would take an incredible amount of time, as every single hand would have to be progressively iterated in order to optimize the decision process.

One way that has been attempted to circumvent this is to implement these Q-Tables as a neural network with inputs representing the state space and ouputs representing the different Q-Values of the function in an attempt to create a proper function approximator instead of using a table. This has however historically been known to be unstable using a nonlinear function approximator, which is precisely what a neural network is[4]. This shortcoming was then addressed by Google DeepMind in 2015, where they attempted successfully to fix this with two ideas, namely what they call experience replay, as well as a mechanism that adjusts the outputs of the network towards the target value only periodically[5]. In experience replay, the data is randomized and replayed, instead of iteratively applying this network over the environment as is standard with Q-Learning. These two ideas are quintessential to the concept of a Deep Q-Network.

1.2 Cribbage

The form of cribbage that I'll be implementing and creating this network for is a two player version of cribbage, using a standard 52 card deck. The way the game is played as follows:

1. Dealer shuffles the cards
2. Non-dealer cuts the cards
3. Dealer deals 6 cards to each player one at a time
4. Both players discard two cards from their hand. These four cards go into what's called the "crib"
5. The non-dealer then cuts the deck, and the top card is turned face up. This card counts for when the pegging phase is done towards all hands, including the crib. If the card turned up is a Jack, the dealer scores 2 points. This card is called the "start card"
6. Players take turns playing during what's often referred as the "pegging" phase
7. Once the pegging phase is done, the hands are scored in order non-dealer, then dealer, then the crib is scored for the dealer

This is repeated, alternating the dealer between both players until one player reaches 121 points or more. If at any point during scoring a player reaches 121 points or more, play is stopped and that player is declared the winner. This holds true even if it's during the pegging phase, meaning a player can lose even before their hand is scored [1][6]. One thing to note is that every card uses the value on its face for all purposes where adding is necessary except for face cards; the Ace is worth 1, and all other face cards are worth the value 10.

Because this project only deals with the scoring phase, I'll not be describing the scoring rules for the pegging phase. In my modified version of the game, everything remains the same, however there will be no pegging phase. During scoring the rules are as follows:

1. Any combination of cards adding up to 15 scores 2 points. For example a Jack and Five count as fifteen, but so do a Four, Five and Six. As a more specific example, a hand of Four, Five(1), Five(2), Six would contain 4 points worth of 15s. These are the combinations Four, Five(1), Six as well as the second trio of Four, Five(2), Six.

2. Any pair of cards of the same face scores 2 points. This effect can stack as well, for example a hand Four(1), Four(2), Four(3) will contain 6 points worth of pairs, for each combination of pairs possible. This effect runs the same for a four of a kind as well, scoring 12 points. To note is that two different face cards, such as a Jack and King, despite holding the same value of 10 for counting do not count as a pair.

3. Any three cards or more of consecutive rank scores one point per card in the run, such as Four-Five-Six, which would count 3 points. This effect can happen multiple times for a hand such as Four-Five-Five-Six, which contains two runs of three cards, totalling 6 points in runs, but for a hand such as Five-Six-Seven-Eight, this only counts 4 points for the run of four cards.

4. If all four cards in a player's hand are of the same suit, 4 points are scored. If the start card also matches suit, it instead scores 5 points. Note that you only score this if all the player's hand is the same suit; if the start card matches suit with three of the player's hand, this does not count as a 4 point flush.

5. The last rule is called ”one for his nob”. If the hand contains the Jack card that matches the suit of the start card, that player scores an extra point.

One thing to note during all of this scoring is that the same card can be used as part of several of these different scoring rules. For example, if you have a hand Seven-Eight-Eight-Nine, you can score 4 points of 15s (Seven-Eight twice), 2 points for a pair (Eight-Eight), as well as 6 points for runs (Seven-Eight-Nine twice). This hand would therefore score a total of 12 points. These same scoring rules also apply to the crib, and those points go to the player who is currently acting as the dealer.

Because of these very clear cut rules for scoring, and some of the emergent strategy behind the crib, where you want to minimize the amount of points you give away via your discards to the opponent if they're dealing, and you want to maximize the number of points split between your hand and the crib when you're dealing, this makes it a perfect candidate for reinforcement learning. As described, reinforcement learning tries to maximize reward and these rules give us a very clear definition for rewarding our agent.

2 Related Works

Previous works on the game of cribbage are incredibly limited, likely due to to it being significantly less popular than games like poker, backgammon or blackjack and possibly due to it's incredibly large and random environment.

Namely, Russel O'Connor applied temporal difference learning on a multi-layer perceptron to the game of cribbage in 2000[7]. However, their approach was applied to the game as a whole and their application of temporal difference learning was with a single hidden layer, where the weights of each layer was updated according using the policy and reward similar to the formula described above for Q-Learning. It also does not specify exactly how the network was used, it merely explains the structure of the network.

There is also a genetic algorithm that was applied to cribbage by Graham Kendall and Stephen Shaw in 2003 [8]. This paper does also only deal with the scoring phase and not the pegging phase, however they do also completely cut out suits from the game, reducing the number of possible six card hands down to 18,395. This is mostly fine seeing as suits play an incredibly small part in the game, but I wanted to be as accurate as possible to the entire discard and scoring phase as a whole. They do mention some interesting ideas, however most of these are not applicable to my situation and are specific to a genetic adversarial approach, as they did in their paper.

3 Network Structure

With all of these requirements done, I can now begin my approach to codify the network.

The input for the network is relatively straightfoward as it contains a one-hot encoding of the six cards in the player's hand consisting of 318 nodes, followed by 3 more inputs, one for the player's score, their opponents score, and a bit for determining whether the player is dealer or not. The hope with encoding the players scores and the dealer as information is to make the network consider the opponent's score with their decision. This is especially important later in the game, as you want to especially make sure you minimize the opponent's score if they're on the verge of winning and the dealer, as this is when you can best minimize their score if they're on the verge of winning. The dealer bit also helps the agent know whether it's fine to discard cards that give points to the crib, as if they are the dealer, they'll get to score those points regardless. The cards in the input will always be sorted to introduce some consistency in the way the output is represented.

The output for the network will be 15 nodes. Each of these 15 nodes will correspond to the assumed reward for a certain pair of cards being discarded. This functions as our Bellman Equation, however because in our case there are no effects of the current decision on future rewards due to the way the game is played, I completely ignore computing the estimated reward of future states and instead use the output nodes as simply

Because of the learning nature of this project, I tried a few different approaches and also made a few preliminary benchmark tests.

3.1 Benchmarks

Before moving forward, I wanted to assess some benchmarks that I could aim towards. Namely, I wanted to answer some questions about how good naive players are at the game, so I could better assess the performance of my model later on. It's interesting to create a model that can learn to play a game, but it's not very useful to evaluate it's metrics unless there's something to compare it to.

Unfortunately I don't have access to any known algorithms that can play the game of cribbage, but what I do have is two naive approaches to metrics, namely:

1. How an agent that plays completely randomly performs
2. How an agent that optimizes the hand score, ignoring the crib, can perform

This means I now have two other agents to compare the model to. From here, we can now evaluate some benchmarks, and set some goals for the model. Of course, before this, we need to know what metrics exactly we'll be evaluating. For this, I decided that I'd compare the average scores of hands between the agents. I also had the agents play matches against each other, and compared their scores. These results can be seen in Tables 1, 2 and 3.

Table 1: Player Average Scores Without Crib (1000 hands)

	Average Score
Random Player	2.682
Network Player	2.575
Naive Player	6.147

Table 2: Player Score Differences With Crib (1000 hands)

	Random Player	Network Player	Naive Player
Random Player		-0.055	-3.423
Network Player	0.055		-3.716
Naive Player	3.423	3.716	

As seen in the results, a completely randomized network seems to perform about as well as a player that picks cards completely at random, which is expected. What's more important is to see that the naive player achieves an average of about 6 points per hand without including a crib. This is a good benchmark for the network, as we now know what it looks like for the network to pick the optimal scoring hand without considering the crib.

Table 3: Player Match Scores (1000 games)

	Random Player	Network Player	Naive Player
Random Player		529-471	27-973
Network Player	471-529		20-980
Naive Player	973-27	980-20	

One thing to note is that the average hand scores were done without a crib. This was just to avoid any external randomness from affecting the benchmark.

4 Experiments

5 Results

6 Future Work

References

[1] B. Cards, "Learn to play cribbage," 2022.

[2] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*, pp. 3–42. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[3] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[4] B. Van Roy *et al.*, "An analysis of temporal-difference learning with function approximation," *Automatic Control, IEEE Transactions on*, vol. 42, no. 5, pp. 674–690, 1997.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] J. Paxton, "Six card cribbage," 1995.

[7] R. O'Connor, "Temporal difference reinforcement learning applied to cribbage," 2000.

[8] G. Kendall and S. Shaw, "Investigation of an adaptive cribbage player," in *Computers and Games* (J. Schaeffer, M. Müller, and Y. Björnsson, eds.), (Berlin, Heidelberg), pp. 29–41, Springer Berlin Heidelberg, 2003.