Learning to Play Euchre With Model-Free Reinforcement Learning

Eli Pugh
Stanford University
Mathematics & Computer Science
epugh@stanford.edu

Abstract

Euchre is a card game between two teams of partners compete in rounds. While once a popular card game, it has declined in popularity, and thus there haven't been any significant attempts to use modern reinforcement learning methods to solve it. In this paper, we explore different methods for learning to play Euchre, including Deep Q-Networks (DQN) (Mnih et al., 2013) and Neural Fictitious Self-Play (NFSP) (Heinrich and Silver, 2016). We find that while these methods often perform much better than random, they only beat smart rule-based agents just over 50% of the time, even with a large amount of training and clever augmentation. This might suggest that tree search methods will likely perform better on games like this, or possibly extensions made to the methods we present would improve learning. We make code available at https://github.com/elipugh/euchre.

1 Introduction

Euchre is a game dating back to the mid 1800's, and remains fairly popular in areas of England. It was considered the national card game of the United States in the late 1800's, but has since declined in popularity. It remains very popular in the Midwest, including Indiana, Michigan, Wisconsin, and also in Ontario.

The state space of Euchre is very large, consisting of several "tricks", where each trick consists of different stages. The game of Euchre is described more in detail in Section 2.

In Section 2 we discuss related work, in Section 3 we discuss modelling Euchre as a Partially Observable Markov Decision Process (POMDP), in Section 4 we discuss agents and training methods, and in Section 5 we discuss results.

2 Related Work

Reinforcement learning has seen significant success in game-playing problems from Atari games to Poker. Some games with smaller state spaces can be learned analytically with model-based reinforcement learning methods such as optimizing over maximum likelihood or using Bayesian methods (Ghavamzadeh et al., 2016). Euchre has a massive state space (further discussed in Section 3) and thus model-based methods are intractable, as they require too much storage and training examples to work well.

Games with significantly larger state-spaces often turn to Q-learning (Watkins, 1989) or Sarsa (Rummery and Niranjan, 1994). These methods can often be very slow, and thus benefit from eligibility traces (Sutton, 1988). Both work by keeping a Q-matrix that stores the inferred values of each $(state, action)$ pair. These are iterative algorithms over generated data, and the update values are below. (s, a) are state and action, a prime symbol denotes next state/action, α is a learning rate, r is the reward, and γ is the problem discount factor.

Q-Learning :

$$Q(s,a) += \alpha\Big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\Big)$$

Sarsa :

$$Q(s,a) += \alpha\Big(r + \gamma Q(s',a') - Q(s,a)\Big)$$

See Figure 1 for a diagram of the Q-Learning algorithm.

Q-Learning with eligibility traces (Watkins, 1989) have been explored extensively and show great results on many problems, including many card games such as Blackjack, Leduc Texas-Hold-Em, and others (Zha et al., 2020). These methods also struggle with massive state-spaces, since

we must store a Q matrix that stores values for each $(state, action)$ pair. To remedy this, often k-Nearest-Neighbors or interpolation works well to infer unseen $(state, action)$ pairs.

Deep Q-Learning (Mnih et al., 2013) is another remedy to this in cases where interpolation doesn't work as well. Deep Q-Learning instead trains a deep neural network to approximate the Q function, rather than storing the massive table of Q values. This is seen in Figure 2. Deep Q-Learning is discussed more in section 4.2.

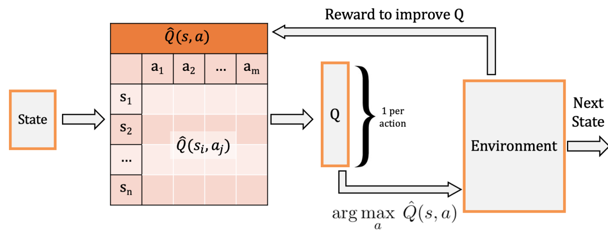

Figure 1: Q-Learning

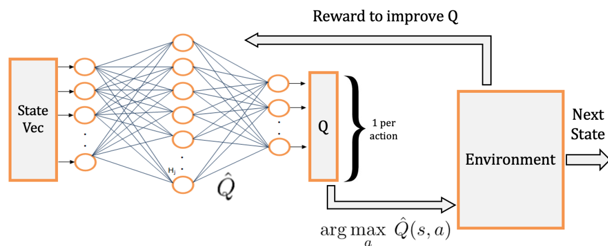

Figure 2: Deep Q-Learning

Though there have many demonstrations of these methods on many games, there has been little interest in Euchre. We hypothesize that this is due to the diminishing popularity of the game, and smaller AI community in the American Midwest where most players reside. One interesting approach by Marias Grioni was to use Monte Carlo Tree Search (MCTS) (Silver et al., 2016). This is published at https://github.com/matgrioni/Euchre-bot. All other Euchre AI attempts seem to be human-coded rule-based agents. Since MCTS has been explored, I decided to stray from tree-search methods in favor of model-free reinforcement learning methods.

3 Modelling Euchre

3.1 Gameplay

Euchre is a game where two teams of two partners compete to win 10 points. The game is played with 24 cards, including all suits of 9, 10, Jack, Queen, King, and Ace.

The game begins with one player dealing each person 5 cards, and then 1 card is placed face up. Then starting to the left of the dealer, each person may decide to "pass" or tell the dealer to "pick up" this card. When the card is "picked up", that it decides the "trump suit" for the rest of the hand. If all players pass, an there is one more round where each player has the option to "pass" or choose the choose the "trump suit".

Once the "trump suit" has been decided, all players have 5 cards, and there are 5 rounds. In each round, each player (in order) plays a card. If you have the same suit as the first card that was led, you must play the same suit. Otherwise, you are free to choose any card in your hand. After each player has played a card, the team with the best card is awarded a point. The best card is the highest card in the trump suit, or if no trump are played, the highest card of the same suit that was led. Each of these plays is called a "trick" and the team who wins 3 or more of the 5 tricks wins the round. If the winning team did not decide trump or they won all 5 tricks, they get 2 points. Otherwise, they get 1. To win, a team must get 10 points.

3.2 Euchre as a POMDP

In each stage of Euchre, there is some uncertainty from which cards each player holds. This can be partially observed by behaviour and playing strategy. In addition, you always know your own cards as well as some history from the past tricks. Thus we can model the game as a POMDP and use traditional model-free reinforcement-learning methods to learn to play this game.

3.3 Euchre State Space

We'll start by looking at the total (partially unobservable) state space. There are $(24!/(5! \cdot 4 + 3!))$ ways of dealing. Then there are additionally (4) possibilities if trump is chosen in the first round, or $(3 \cdot 4)$ possibilities if it's chosen in the second round through. This means before any card-laying starts, there are $(24! \cdot 48/(5! \cdot 4 + 3!))$ possible states, which is on the order of 10^{22}. Once card playing begins, there are round i will have up to $((5 - i) \cdot 4)$ plays. In total, there are up to roughly 10^{25} states.

3.4 States

When we represent this, we will need to separate based on what is observable to a given agent and

what is not. We can implement this roughly as follows:

```
State:
  Observable:
    Stage (0-1 picking trump,
           2-6 are tricks)
    Flipped Card ([Suit, Value])
    Your hand ([S, V]^5)
    Played card history ([S, V]^4·5)
    Current player (1-4)
  Unobservable:
    Opponent Hands([S, V]^3·5)
```

4 Methodology

4.1 Implementing Euchre

To model Euchre, we used RLCard, an open-source toolkit for reinforcement learning in card games (Zha et al., 2020). RLCard allows one to define environments using a simple interface. To implement a new game, one must design a structure for each of:

- `Player` to store the hand for each agent
- `Game` to define rules, states, and transitions
- `Round` to define how each round is played
- `Dealer` to define how cards are distributed
- `Judger` to assign scores at the end

By structuring the interface in this way, we were able to use some of the agents included in the RLCard package and evaluate their performance.

4.2 Agents

For comparisons, we used both a `RandomAgent` and `RuleAgent` as baselines. We experimented with different versions of both a `DQNAgent` and a `NFSPAgent`. We will define each in detail below:

`RandomAgent` is one that selects from each legal action uniformly at random. The set of legal actions are made available to agents during the game, so there are no issues with breaking rules.

`RuleAgent` was programmed using domain knowledge from the game of Euchre. After centuries of playing the game, humans have arrived at a generally optimal way of playing Euchre. While the optimal method of play depends heavily on the behaviour of other players as well as a more detailed history log, one can program a very performant agent of Euchre using rules conditioned on only the observable state as described in Section 3.4. This is a very strong baseline, as it closely mirrors human expert play.

`DQNAgent` is an agent that uses Deep Q-Learning to approximate the value of each action in a given state. This agent also used ϵ-greedy exploration with exponentially decaying ϵ. We also used a large replay memory. In some cases, we experimented with feeding the DQN agent data generated from the rule agents to start training with an initialization close to rule-based agent.

`NFSPAgent` is an agent that uses Neural Fictitious Self Play (Heinrich and Silver, 2016). This is where not only is a $\hat{Q}$ network trained to approximate values of each action given a state, but we also keep a network Π that maps states to action probabilities. Π is trained using supervised learning over the history of previous best responses. We switch between these two networks with a probability η to train Q and Π. Similarly to the `DQNAgent`, we also experimented with feeding the agent data generated from the rule agents to start training with an initialization close to rule-based agent.

5 Results

In order to benchmark these different agents, we played with teams of two trained agents versus teams of `RuleAgents`. We show results where the metric is win percentage over 50 averaged across 1000 games. Given the nature of Euchre, playing a game is computationally expensive, and thus we only run 1000 games per evaluation, though the long nature also makes for a lower-variance result. Also notice that since `RuleAgents` closely mimic human play, they are difficult baselines to beat.

Also notice that since we are measuring across win percentage instead of average score, a result of 10% can often mean that the average point margin is actually less than 10%. In addition, note that time and resource constraints meant we couldn't run as many times as we hoped, or for as long.

Results over training are shown in Figures 3,4,5,6. In these figures, initialization to rule-based approach means the network was trained first to imitate the `RuleAgent`.

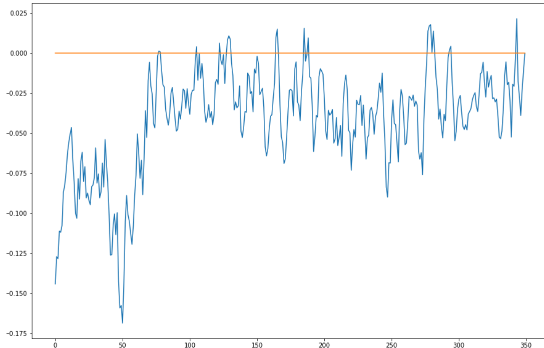

Figure 3: DQN Agent Win Rate Over Training

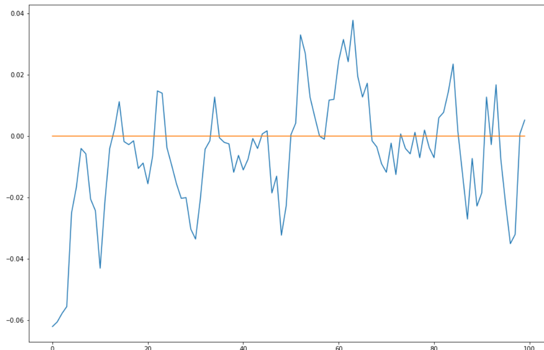

Figure 4: NFSP Agent Win Rate Over Training

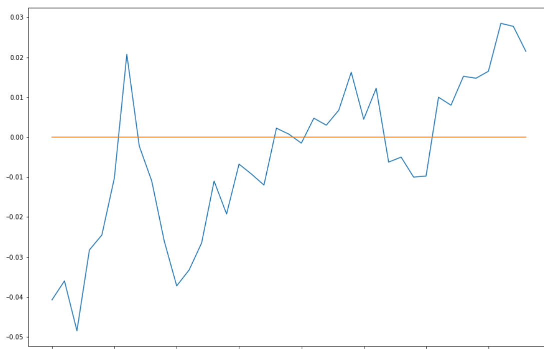

Figure 5: DQN (Init to Rule Agent) Win Rate

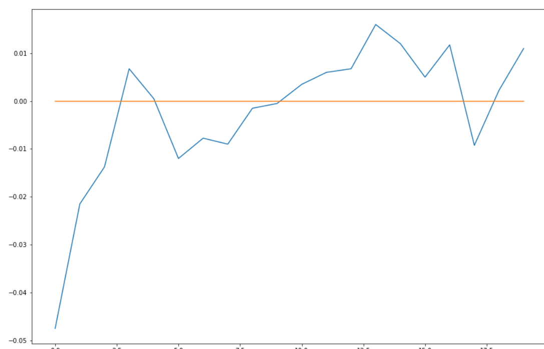

Figure 6: NFSP (Init to Rule Agent) Win Rate

5.1 Discussion

For this, we will discuss each training plot and the results in order.

We were able to train the DQN Agent over more epochs than any other agent, mainly because it was slightly faster than NFSP. Notice that the loss for the DQN Agent start much lower than the others. This is due to no pretraining on data generated from rule-based approach.

The NFSP performed similarly to the DQN Agent, but actually took less iterations to reach a good value. Notice that nearly immediately the relative win rate is above -5%, which takes the DQN agent over 50 iterations. This is most likely because NFSP reaches a Nash Equilibrium much more quickly than other methods when used for card games (Heinrich and Silver, 2016), though often it doesn't add any representational power over a DQN.

The DQN agent that is pretrained on `RuleAgent` data has a training curve that looks similar to the NFSP Agent. This is likely because the main advantage of NFSP is a faster start.

The NFSP agent that is pretrained on `RuleAgent` data doesn't really improve much over the original NFSP Agent. Both reach similar performance. We regrettably didn't have time to train this longer, though it seems like it still might hit a similar ceiling to the NFSP agent.

6 Conclusion

In this paper, we presented the first modern reinforcement-learning approaches to the game Euchre, a multiplayer card game that involves some cooperation between players and has a massive state space. We show that Deep Q-Networks and Neural Fictitious Self-Play approaches can rival rule-based agents programmed to mimic expert play.

While these approaches perform well, future directions into Monte-Carlo Tree Search (Silver et al., 2016) and related algorithms are certainly worth exploring.

In addition, we showed that while Euchre is very computationally challenging with many stages, many states, and slow computation, it's still possible to train large networks to approximate a Q function to develop policies.

References

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. 2016. Bayesian reinforcement learning: A survey. *CoRR*, abs/1609.04436.

Johannes Heinrich and David Silver. 2016. Deep reinforcement learning from self-play in imperfect-information games. *CoRR*, abs/1603.01121.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. 2013. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602.

G. Rummery and Mahesan Niranjan. 1994. On-line q-learning using connectionist systems. *Technical Report CUED/F-INFENG/TR 166*.

D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489.

Richard Sutton. 1988. Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44.

Christopher Watkins. 1989. Learning from delayed rewards.

Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. 2020. Rlcard: A toolkit for reinforcement learning in card games.