Evaluating Reinforcement Learning Agents playing Leduc Hold'em Poker

CS7IS2 Project (2019/2020)

Dhruv Ishpuniani, Sakina Vohra, Sujit Jadhav

ishpunid@tcd.ie, vohras@tcd.ie, jadhavs@tcd.ie

Abstract. Reinforcement Learning(RL) allows agents to learn from a feedback loop without any supervised signals making it one of the most promising solutions for solving complex problems like Poker Games where the state of the game at every step is unknown. In this paper we discuss, three novel algorithms of RL, Counterfactual Regret Minimization, Deep-Q Learning and Neural Fictitious Self-Play (NFSP) and evaluate them against each other to find out which gives the best results for the Leduc Hold'em problem.

Keywords: poker, counterfactual regret minimization, neural fictitious self-play, deep q-network, reinforcement learning, imperfect information games

1 Introduction

In 2016, Lee Sedol, a 18 time world champion was defeated by Google DeepMind's AlphaGo, an Artificial Intelligence based computer program designed for the board game Go setting a new benchmark for recreational games. Many successful AI agents or bots have been programmed throughout the years that were able to beat humans in games like Chess, Checkers, Go etc, these games are considered to be Perfect Information Games, as both the players are aware of the state of the game at every new step. In 2017, an Artificial Intelligence Poker bot, Libratus[1] made a breakthrough by winning against a team of four professionals in Heads-up no-limit Texas Hold'em game raising many eyebrows and opening doors to more innovation and research in applications of AI in imperfect information games.

Imperfect Information games like card games have most game states hidden from the player. Researchers are trying to find an ideal strategy to solve these problems by building AI agents that mimic human behaviour. As the gaming agents do not have complete information, they have to rely on other factors like behavioural or psychological aspects. This increases the complexity in building such an agent. Poker is a perfect example of an Imperfect Information game as it is played with little information and highly relies on the psychological aspect of the game. Different players have different strategies and artificial intelligence can be used to understand these strategies, find the loop holes in them and use it to win the game.

Due to the uncertain nature of the Imperfect Information game, we require a paradigm that allows the agents to learn by themselves by trial-and-error. In recent years, Reinforcement Learning(RL) has witnessed major breakthroughs in the field of gaming as here the agents learn from evaluative feedback without any supervised signals [2], making RL a good solution for Poker. In simplest terms, an RL agent first receives a state from the state space and it selects an action from action space using a policy. The policy here is the behaviour of an agent which is used to associate every state to an action. On every action, the RL agent receives a reward which is used for further modelling. In this paper, we discuss three RL algorithms viz. Counterfactual Regret Minimization, Deep-Q learning and Neural Fictitious Self-Play to understand which one yields best solution to the problem of Leduc Hold'em Poker.

2 Related Work

In this section we review techniques that are widely used to build near perfect poker agents.

2.1 Rule-based Systems

The first approach we review is that of rule-based systems. Domain specific human experts are needed to assist with the development of rule-based systems. Numerous simple poker playing agents are rule-based. They are a set of multiple conditional statements which can be visualised as a hand-made decision tree. One such agent is SoarBot[4], a rule-based expert system for Poker built using the SOAR rule-based architecture[5]. Rule-based agents have mediocre performance and may be used as baseline players.

2.2 Evolutionary Algorithms

The application of evolutionary algorithms to build game playing agents had been an interesting topic of research in the first decade of the century. The idea behind an evolutionary algorithm is based on the theory of evolution. In our case, it means that the strongest strategies survive and reproduce to form superior strategies and the weaker strategies eventually become extinct. An example of such algorithm is presented by Noble in [6]. Noble uses Pareto Coevolution to evolve poker agents, where each poker agent is represented as an artificial neural network(ANN). Coevolution algorithms maintain multiple genetic populations where members of the different populations are allowed to compete with each other, however, the evolution is restricted to their own population. The input of an ANN is the current state of the game and the output nodes correspond to actions. This particular approach improved the state-of-the-art at the time of publication(2002). An earlier approach by the same author uses a less refined rule-based representation[7].

2.3 Reinforcement Learning

Reinforcement Learning is a semi-supervised machine learning technique commonly used to build game playing agents. It consists of an agent exploring and exploiting an environment. Based on the agent's actions, it may either be rewarded or penalised. Using a value function, it learns to identify the most desirable action in that given game state.

Counterfactual Regret Minimization(CFR) CFR is a RL algorithm that analyses the effect of each move played by the agent and attempts to minimize the loss. Using this algorithm, a Texas Hold'em Poker agent was created in [14] having the information set in the order of 10^{12}. The agent was trained by using 3 layers of abstraction in the available infrastructure. These abstractions included reducing the information set, measuring the winning hand by using squared hand strength metric and partitioning the hands based on hand strength.

Unlike other RL algorithms, CFR considers all the possible trajectories in the game and calculates the rewards accordingly. In addition, this algorithm can be used to reduce the size of information set of problems with massive information sets. These attributes of the CFR algorithm led to it being one of the approaches to be evaluated.

Deep-Q Learning Network(DQL/DQN) Most RL applications require handlabelling the data however it is expected that the model learns by itself from reward signals. Now there could be a significant delay between the action and the resulting reward which makes it complex to map inputs with it's targets. Apart from this, RL looks for correlation in the states but it's difficult to find it in independent data. For this, in the DeepMind paper 2013 [17], Deep Q learning was introduced. The paper discusses how CNN can learn control policies from raw video gaming data that is trained on a type of Q-learning algorithm. This algorithm used raw pixels as input for 2006 Atari games and performed better than humans in games like Pong, Enduro and Breakout.

Another paper [9] proposes a new agent called deep-Q network which combines reinforcement learning with deep neural networks using several layers of nodes for more data abstraction to make the neural network able to categorise objects from raw data. Here the agent interacts with the environment using actions, rewards and observations and select an action that can maximise the expected reward. This technique is evaluated against the best techniques of Reinforcement Learning by playing Atari games. Here, it outperforms all other techniques in most cases and human in 75 percent of cases.

We choose this algorithm because it reduces the variance of the updates and these weight updates is used in each step of experience for greater data efficiency in a heavy game like Poker. It also uses Experience replay that averages the behaviour distribution over it's previous states and avoids divergence in parameters.

Neural Fictitious Self-Play (NFSP) The NFSP[10] agents learns by playing against itself, without any explicitly defined prior knowledge. It uses DQN[9] to replay experience with Q-Learning[8]. When applied to Leduc Poker, the NFSP agent learns a strategy which challenges the performance of state-of-the-art methods based on agents embedded with prior knowledge.

We have selected this paper as one of the approaches to be evaluated since an NFSP agent internally uses DQN to build an agent and it would be interesting to compare them. Secondly, the agent does not need to exhaustively iterate over the game's states and may not even need a model of the game's dynamics.

3 Problem Definition and Algorithms

3.1 Problem Definition

We choose to train our agent on Leduc Poker, a two player zero sum game, which follows the same concepts and strategies of a normal Poker game and also due to its tractable information set size. Leduc hold'em Poker is played as follows:

– The deck consists of only 3 cards from 2 suits which are (J, J, Q, Q, K, K).
– Each player gets 1 card.
– There are two betting rounds, and the total number of raises in each round is at most 2.
– In the second round (preflop), one card is revealed on the table and this is used to create a hand and another card is revealed in next round(flop).
– There are two types of hands: pair and highest card.
– There are three moves: call, raise, and fold.
– Each of the two players starts with betting amount of 1.
– In the first round, the betting amount is 2. In the second round, it is 4.

3.2 CFR

– **Counterfactual** : This means what if given a chance to undo something and then do it the next time in a better way i.e. "What if I had done this. . . "
– **Regret** : This tells us the effect of doing something wrong i.e. "How much would it have impacted if I had chosen to do this instead of what I did?"
– **Minimization** : What actions and strategies can be followed to decrease the overall loss i.e. "What are the strategies to reduce regret?"

CFR algorithm is applied to those games in which information state is not fully known i.e. they are non-deterministic in nature. In case of any poker game, it is impossible to know the opponent players' cards. This type of algorithm is played and trained against other intelligent poker agents. The methodology to obtain Nash Equilibrium in [15] is as follows:

1. Generate the strategies for actions[Fold, Call, Raise] in the format *fcr(P(F), P(C), P(R))* and combine them to form a strategy profile.

2. Calculate best response strategy using CFR and generate the best response strategy profile.
3. Compute the rewards using the utilities function with the help of strategy profile used against each player.
4. Compute the rewards using the utilities function with the help of the best response strategy profile used against each player.
5. Compare the rewards obtained from utilities using strategy profile and best response strategy profile.
6. ϵ-Nash Equilibrium is achieved if the rewards obtained from utilities using the best response strategy profile do not change more than ϵ.

3.3 Deep-Q Learning (DQL)

This algorithm is built upon Tesauro's TD-Gammon Architecture where parameters of the network are updated from policy samples drawn from the interaction with the environment. This approach outperformed many human professionals. For more significant results, it is combined with deep neural network and scalable RL algorithms. In this algorithm [17], a technique called Experience Relay is used to store agent's experience at each time-step in a Data set D taken from episodes into replay memory. In the inner loop, E-learning is applied to stored samples drawn randomly. After this the agent selects and implements an action as per ϵ-greedy policy. This algorithm is called Deep-Q learning.

```
1: function DQNAgent($\Gamma$)
2:     Initialize replay memory D to capacity N
3:     Initialize action-value function Q with random weights
4:     for episode = 1, M do
5:         Initialise sequence $s1 = x_1$ and preprocessed sequenced $\phi_1 = \phi$(s1)
6:         for t = 1, T do
7:             With probability $\epsilon$ select a random action $a_t = max_a Q*(\phi(s_t), a; \theta)$
8:             Execute action $a_t$ in emulator, observe reward $r_t$ and image $x_{t+1}$
9:             Set $s_{t+1} = s_t$,$a_t$,$x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
10:            Store Transition ($\phi_t$, $a_t$, $r_t$,$\phi_{t+1}$) in D
11:            Sample random minibatch of transitions ($\phi_t$, $a_t$, $r_t$, $\phi_{t+1}$) from D
12:            Set policy $\leftarrow \begin{cases} r_j, \text{ for terminal } \phi_{j+1} \\ r_j + \gamma max'_a Q(\phi_{j+1}, a'; \theta) \text{ , for non terminal } \phi_{j+1} \end{cases}$
13:            Perform a gradient descent step on ($y_j$ - Q($\phi_j$, $a_j$; $\theta$))$^2$
14:        end for
15:    end for
16: end function
```

3.4 Neural Fictitious Self-Play (NFSP)

The training process starts with multiple agents initialized with random strategies. An agent strengthens itself by improving its own strategy, based on the knowledge of the opponents' strategies. An NFSP agent has two kinds of memories and is made up of two neural networks. The first memory stores the agent's playing experience against other agents and is used by RL to train a network that predicts expected values of actions. The other memory stores the experience of the agent's own behaviour and is used by supervised learning to predict the agent's own average behavior.

The agent chooses the strategy for each episode from a mix of its best-response strategy and average response strategy. The parameter η is called the anticipatory parameter and it helps regulate this mixture.

The NFSP algorithm from [10] is defined below:

```
1: Initialize game $\mathit{\Gamma}$ and execute an agent via RunAgent for each player in
   the game
2: function RunAgent($\mathit{\Gamma}$)
3:     Initialize replay memories $M_{RL}$ (circular buffer) and $M_{SL}$ (reservoir)
4:     Initialize average-policy network $\mathit{\Pi}(s, a|\theta^{\mathit{\Pi}})$ with random parameters $\theta^{\mathit{\Pi}}$
5:     Initialize action-value network $Q(s, a|\theta^{Q})$ with random parameters $\theta^{Q}$
6:     Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
7:     Initialize anticipatory parameter $\eta$
8:     for each episode do
9:         Set policy $\sigma \leftarrow \begin{cases} \epsilon\text{-greedy}(Q), \text{ with probability } \eta \\ \mathit{\Pi}, \text{ with probability } 1 - \eta \end{cases}$
10:        Observe initial information state $s_1$ and reward $r_1$
11:        for $t = 1, T$ do
12:            Sample action $a_t$ from policy $\sigma$
13:            Execute action $a_t$ and observe reward $r_{t+1}$, next state $s_{t+1}$
14:            Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in RL memory $M_{RL}$
15:            if agent follows best response policy $\sigma = \epsilon$-greedy $(Q)$ then
16:                Store behaviour $(s_t, a_t)$ in supervised learning memory $M_{SL}$
17:            end if
18:            Update $\theta^{\mathit{\Pi}}$ with stochastic gradient descent on loss
19:            $L(\theta^{\mathit{\Pi}}) = E_{(s,a)\sim M_{SL}}[-\log \mathit{\Pi}(s, a|\theta^{\mathit{\Pi}})]$
20:            Update $\theta^{Q}$ with stochastic gradient descent on loss
21:            $L(\theta^{Q}) = E_{(s,a,r,s')\sim M_{RL}}[(r + max_{a'} Q(s', a'|\theta^{Q'}) - Q(s, a|\theta^{Q}))^2]$
22:            Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
23:        end for
24:    end for
25: end function
```

4 Experimental Results

4.1 Methodology

We are using RLCard[11] to evaluate the multiple approaches described in Section 3. RLCard is a python based framework designed for reinforcement learning in card games developed by the DATA Lab at Texas A&M University. The main idea behind its development is to bridge the gap between reinforcement learning and imperfect information games, and advance research in the field of Reinforcement Learning[12].

This framework is perfect for our evaluation since we can model environments for numerous games, particularly Leduc Poker, using it. We can train and plug in the trained agents to simulate games and tournaments. Another advantage of using this framework is that it has well defined interfaces to design CFR, DQN and NFSP agents.

We train each agent for the same number of episodes, 100,000. We set the initial size of the replay memory to 1000. The specific training parameters and techniques for the algorithms are as follows:

– We use the CFR Agent given right out of the box in RLCard. We train two CFR agents, one by playing against the random agent and the other by playing against a rudimentary rule based agent. The training environment is initialized with the allow_step_back flag set to true. This allows the agent to traverse back along the game tree and identify "regrets".
– We use a DQN Agent with memory size selected in the range of 1000 to 20000, discount factor of 0.99, Adam optimizer with learning rate of 0.00005 and the network structured as an MLP with size 128-128. We train two DQN agents with these parameters. Similar to the CFR agent, one with a random agent and the other with a rule-based agent.
– For NFSP, the anticipatory parameter is chosen to be 0.1. Memory size for supervised learning and reinforcement learning are 10^6 and 3 x 10^4, respectively. The size of the hidden layer is set to 128-128. The learning rates for supervised learning and reinforcement learning are 0.005 and 0.1, respectively. The remaining parameters specific to the DQN used by NFSP are the same as above. We train the NSFP agent by initializing two NSFP agents in a game environment and letting them play against each other.

The evaluation metric used by RLCard Toolkit is the average payout per game(rewards) which is calculated on the basis of 'big blind' per hand. In Leduc hold'em Poker, the big blind is 1 as the game starts with amount of 1. We initialize the Leduc poker game environment in RLCard with the agents needed to be compared. We then conduct a tournament where the two selected agents play 10,000 games against each other. The results of the tournament are reported as one agent having a positive payout and the other having a negative payout of the same magnitude.

Competing with Baseline The random agent has been selected as the baseline. This agent is equally likely to select an action from the set of all legal actions available to it at that particular state of the game. We evaluate an algorithm by making it compete with the random agent in a tournament.

Competing amongst Agents Another evaluation approach is to make the agents compete with each other. The idea behind this evaluation is that a better

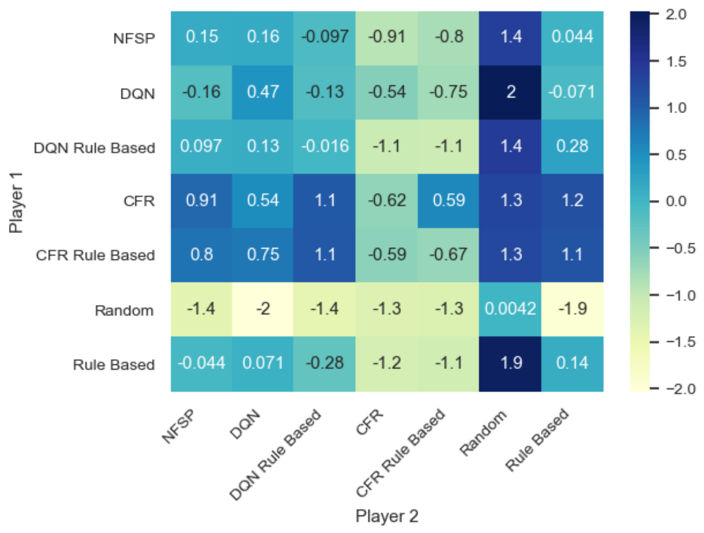

Fig. 1. Average payout per game for Player 1 against Player 2

agent would have a higher average payout per game than the other agents on competing in a tournament.

Competing with Benchmark To benchmark the agents' performance, we make the agents compete with a rule-based agent. The rule-based agent plays aggressively to 'raise' and 'call'.

4.2 Results

The results of the performed experiments are visualised in the form of a Heatmap in Fig 1. The figure shows the average payout per game for Player 1 against Player 2. The payout value of the agent is mapped with the intensity of the color as per the scale. For instance, when the random agent plays against the NFSP agent it gets an average payout of -1.4. Similarly, when NFSP agent plays against the random agent it gets an average payout of 1.4. From the figure, it is evident that all trained agents perform well against the random agent, while CFR agent has the best performance overall.

4.3 Discussion

All the selected approaches perform well against the baseline, random agent, especially DQN. The DQN agent learns a deterministic, greedy strategy. This implies that it would "call" or "raise" in most situations. This strategy works well against random agents since they may easily choose to "fold". However, the

DQN agent struggles to compete with other agents since they are able to exploit it's aggressive strategy.

DQN achieves human-level performance when playing against single agent games, with fixed opponents like Atari. However, it struggles to learn an optimal strategy where the opponents are adapting dynamically. This is because DQN stores its experience in a replay memory and learns against the opponents' average strategy. The NFSP agent uses a slowly changing average policy in self-play. Hence, its experience does not vary as much leading to a more stable strategy. We can see how the NFSP agent performs better than the DQN agent, albeit by a small margin.

Finally, it is evident that CFR is the best performing agent in our experiments. This is so because it uses memory to store the regret by accessing further possible game states. However, for games like Mahjong or No Limit Texas hold'em Poker, the information set is quite large which makes it hard to train as memory needed to store strategies against regrets and rewards would be large.

5 Conclusions

In this research, we evaluated poker agents trained on three algorithms viz; **CFR**, **NFSP** and **DQN** using RLCard[12] along with rule-based and random agents. It is apparent that the CFR agent outperforms all other agents. This is because it stores regrets and constantly minimises them to get maximum rewards. The trained agents are fitting opponents to play against any good poker player. However, there will always be some scope to exploit them by playing out of the box moves. Agents can further be trained by using other CFR algorithms like Deep CFR, Monte Carlo CFR or CFR+ and can be evaluated against each other. Due to limited computing power, the evaluation was carried out on Leduc Poker. Training a Limit hold'em poker agent or any other card game agent with a large information set can be carried out in future. Future work could entail changing the evaluation metric to milli big blind per hand(mbb/h) [16] which would help us compare trained agents' performances with the state-of-the-art. Research and development of techniques to visualise various game playing strategies could also be a promising research topic.

References

1. Brown N. and Sandholm T(2017). Libratus: The Superhuman AI for No-Limit Poker.in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, pp. 5226-5228. [online] Available at: https://www.cs.cmu.edu/~noamb/papers/17-IJCAI-Libratus.pdf
2. Jeerige A., Bein D. and Verma A. (2019). Comparison of Deep Reinforcement Learning Approaches for Intelligent Game Playing.in: 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC). [online] Available at: https://ieeexplore-ieee-org.elib.tcd.ie/stamp/stamp.jsp?tp=&arnumber=8666545

3. Niklaus, J., Alberti, M., Pondenkandath, V., Ingold, R. and Liwicki, M. (2019). Survey of Artificial Intelligence for Card Games and Its Application to the Swiss Game Jass. arXiv:1906.04439 [cs]. [online] Available at: https://arxiv.org/abs/1906.04439.
4. Follek, R. (2003). SoarBot: A Rule-Based System For Playing Poker. [online] Available at: https://support.csis.pace.edu/CSISWeb/docs/MSThesis/FollekRobert.pdf.
5. Laird, J.E., Newell, A. and Rosenbloom, P.S. (1987). SOAR: An architecture for general intelligence. Artificial Intelligence, 33(1), pp.1–64.
6. J. Noble, Finding robust Texas hold'em poker strategies using Pareto coevolution and deterministic crowding, in: Proceedings of the 2002 International Conference on Machine Learning and Applications – ICMLA 2002, 2002, pp. 233–239.
7. J. Noble, R.A. Watson, Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for Pareto selection, Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001, Morgan Kaufmann (2001), pp. 493-500
8. Watkins, C.J.C.H. and Dayan, P. (1992). Q-learning. Machine Learning, 8(3–4), pp.279–292.
9. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, [online] 518(7540), pp.529–533. Available at: https://web.stanford.edu/class/psych209/Readings/MnihEtAlHassibis15NatureControlDeepRL.pdf.
10. Heinrich, J. and Silver, D. (2016). Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. arXiv:1603.01121 [cs]. [online] Available at: https://arxiv.org/abs/1603.01121.
11. RLCard: A Toolkit for Reinforcement Learning in Card Games https://github.com/datamllab/rlcard
12. Zha, D., Lai, K.-H., Cao, Y., Huang, S., Wei, R., Guo, J. and Hu, X. (2020). RLCard: A Toolkit for Reinforcement Learning in Card Games. arXiv:1910.04376 [cs]. [online] Available at: https://arxiv.org/abs/1910.04376.
13. Neller, T. and Lanctot, M. (2013). An Introduction to Counterfactual Regret Minimization. [online] Available at: http://modelai.gettysburg.edu/2013/cfr/cfr.pdf.
14. Zinkevich, M., Johanson, M., Bowling, M. and Piccione, C. (2007). Regret Minimization in Games with Incomplete Information. [online] Available at: http://papers.nips.cc/paper/3306-regret-minimization-in-games-with-incomplete-information.pdf.
15. Risk, N. and Szafron, D. (2010). Using Counterfactual Regret Minimization to Create Competitive Multiplayer Poker Agents. [online] Available at: http://www.ifaamas.org/Proceedings/aamas2010/pdf/01\%20Full\%20Papers/03_01_FP_0089.pdf.
16. Bowling, M., Burch, N., Johanson, M. and Tammelin, O. (2015). Heads-up limit hold'em poker is solved. Science, [online] 347(6218), pp.145–149. Available at: https://science.sciencemag.org/content/sci/suppl/2015/01/07/347.6218.145.DC1/Bowling.SM.pdf.
17. Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D. and Riedmiller M. (2013). Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602v1 [cs.LG]. [online] Available at : https://arxiv.org/pdf/1312.5602v1.pdf