Deep CFR for HUNL

Nima Chitsazan, Foad Khoshouei, Ridhi Mahajan
nc2806@columbia.edu
fk2377@columbia.edu
rm3601@columbia.edu

Abstract

In this project, our goal is to implement the Deep CFR method for the game of No-Limit Texas Holde'm Poker (HUNL) and assess the effect of changes in the hyper-parameters of Deep CFR method on the performance and training iterations of the algorithm. Counter-Factual Regret (CFR) minimization has been widely used in imperfect information games. Multiple variations of CFR, including CFR+, Monte Carlo CFR, Discounted CFR, Deep CFR and Single Deep CFR have been proposed to improve the efficiency. Poker (a large imperfect information game) has been widely used to test the performance of these methods. In this project, we implemented Deep CFR interactively and tested its performance on the games of Leduc and HUNL. We then assessed the effects of changing the hyper-parameters, including the number of training iterations, number of traverses per iteration and the neural network architecture, on the performance of the model. To quantify this, we measured the performance of the agents playing Leduc in an AI versus AI setting. Agents with higher number of iterations have lower exploitability and achieve better results against agents with less training iterations. For HUNL, we implemented action abstraction (Discretized bets), adaptive traversals per iteration and Local Best Response (LBR) to train the agents. The trained agents play poker against humans reasonably. With more iterations the exploitability of the agent can be further improved.

Introduction

A perfect information game in game theory refers to a game in which the decision-maker or the agent has complete information regarding all the events that have previously occurred (Obsborne and Rubinstein 1994). In contrast, in an imperfect information game, the information available to the agent is limited (e.g. in card games such as poker) (Meyerson 1997). Solving an imperfect game problem is equivalent to finding an equilibrium such that there can be no improvement to any player by deviating from the equilibrium.

Linear Programming methods have been widely used for solving imperfect game problems. However, these methods are computationally very expensive for extensive incomplete games such as poker. Zinkevich et al. (2007) proposed the Counter-Factual Regret (CFR) technique to find the approximate solution for imperfect information games by minimizing counterfactual regret. Tammelin (2014) introduced a variant of CFR method called CFR+ and empirically showed that CFR+ has a better performance in comparison to CFR. CFR and its variants were some of the most successful algorithms for solving imperfect information games. However, it has been shown that in cases of highly costly mistakes, CFR+ performs relatively poorly. To address this weakness, a variant of CFR (Discounted CFR) was introduced by Brown et al. (2019) that does not assign uniform weight to each iteration. Instead, earlier iterations are discounted. Some combinations of this method perform significantly better than CFR+. One of the drawbacks of CFR is that it requires abstraction, and often an in-depth knowledge of the game. Deep Learning began has emerged as a method for solving imperfect information games by using function approximations rather than a tabular representation. Deep-CFR (Brown et al. 2019) uses deep neural networks, hence approximating the behaviour of CFR on an un-abstracted game. Deep CFR also performs better than Neural Fictitious Self Play (NFSP) (Heinrich and Silver, 2016), which had been one of the leading algorithms for imperfect information games. Up until now, the major breakthroughs were mostly limited to settings involving two players. However, very recently an AI agent capable of defeating elite human professionals in six-player no-limit Texas holdem poker, was released (Brown et al. 2019). The AI agent (Pluribus) was trained using a form of Monte Carlo CFR (MCCFR) that samples actions in the game tree rather than traversing the entire game tree on each iteration.

Related Work

To apply the above methods on extremely large games, abstraction is normally used. Abstraction needs domain-specific knowledge, is often manual and requires previous knowledge of the equilibrium of the game. Deep Counterfactual Regret Minimization (Deep-CFR) has been used to deal with the abstraction issue in strategic interaction

between multiple agents with only partial information in extremely large imperfect-information games. Deep-CFR (Brown et. al 2019) uses neural networks to approximate the behavior of CFR in the full game. The goal of Deep CFR is to approximate the behavior of CFR without calculating and accumulating regrets at each infoset, by generalizing across similar infosets using function approximation via deep neural networks. The network used to train the model, consists of seven (7) layers and 98,948 parameters. Steinberger (2019) introduced Single-Deep CFR (SD-CFR), which is a simplified variant of Deep-CFR that has a lower overall approximation error. It has been shown that SD-CFR improves upon the convergence of Deep-CFR in poker games and outperforms it. In addition to CFR based methods, other methods in Deep Learning have been used to model imperfect information games as well. Yakovenko et. al (2016), introduced Poker-CNN, which uses a CNN based learning model that can learn the patterns in three different poker games. The model is self-trained and competitive against human expert players. The authors provide a novel representation of the game, which is extendable to different poker variations. Heinrich et. al (2016) introduced a Deep Reinforcement Learning framework for imperfect information games. Neural Fictitious Self-Play (NFSP) approaches a Nash equilibrium when applied to Leduc poker.

Methodology

Problem setup: Let the set of players that play the imperfect information game be denoted by P. Game of Leduc is being played by two players; thus, for the game of Leduc P = $\{P_1, P_2\}$. Let S denote the game tree. An information set (infoset), I, is defined as the set of all the possible game states in the game tree S, given the revealed information to playerP_i. In perfect information games the size of the information set is one (1) regardless of the game state. However, for an imperfect information game, such as Leduc, given the revealed information to each player, the game can be in multiple states. Let A_h be the set of all available actions for player P_i at the node h of the game tree, S , and W_{Ah} be the set of weights, showing the probability that player P_i plays the action $a \in A$ at the node h of the game. If W_{ah} is the probability of action $a \in A$ then:

$$\sum\nolimits_{a \in A} W a_h = 1$$

A game strategy is to assign W_{Ah} for each node h in the game tree. For a game like poker, it is imperative to avoid deterministic weight set (set one weight to 1 and all others to 0), because in this case the opponent would be able to learn the strategy and exploit it.

CFR: CFR algorithm (Zinkevich et al. 2007) aims to solve the imperfect information game problem by finding the Nash Equilibrium strategy. Nash Equilibrium is an approximate equilibrium in which no player can improve by deviating from it (Nash 1950). Zinkevich et al. (2007) proposed finding the Nash equilibrium by minimizing the counter factual regret, defined as:

$$R_{A_h,i} = \max_a \sum_{t=1}^{T} [U_a - U(A|W_A)]$$

where $R_{A_h,i}$ is the regret for player i at node h, U_a is the return that player I could receive by playing action a, and $U(A|W_A)$ is the expected return for player i given the current set of weights W_A. After updating the regrets the CFR moves forward to update the weights by using regret matching as below:

$$\mathrm{R}_{A_h,i} = \begin{cases} 1, & x = 1 \\ 0, & otherwise \end{cases}$$

Intuitively, this value is the average of regrets of all actions in the game tree, given the weight set W_{Ah}. It is proven that by iteratively updating the weights using the proposed regret matching process, that the game will converge to Nash equilibrium.

Deep-CFR: Solving extensive imperfect information games such as poker (even in simplified form e.g. Leduc) using CFR is not computationally practical due to requirement of calculating the game returns U_a for all the branches of the game tree. To resolve this issue, instead of traversing the entire game tree in each iteration, Deep CFR partially traverses the game tree for K number of times, where depth of each traverse is determined by external sampling. When traversing the tree, the Deep CFR determines the action a to be played, by regret matching on the outputs of a Neural Network (NN) that takes the infoset (I) as the input, and outputs a variable V that is proportional to the approximate regret $R_{A_h,i}$. When the search reaches a terminal node, the results are propagated backward throughout the tree. In traverser infosets, the weighted average of all actions are added to a memory M. The values in memory M are then used for calculating the instantaneous regret R. This process of traversing the game tree is done iteratively for each player. When the K traverses for one player is finished, a new network is trained to minimize the MSE between the calculated regrets from the previous run and the output of neural network (V).

Innovation and Contribution

Network Architecture: This study uses the feed-forward network architecture proposed by Brown et al. (2019) as the baseline model. The network architecture is shown in Figure 1. The inputs to the Neural Network are the revealed card information and the betting history. The information from private cards and the community cards are embedded separately into two vectors with dimension 64. Then, the information from these two vectors are transformed and passed through three dense layers with size 192 (Card Layers). The betting information is passed through the network with two dense layers of size 64 (Betting Layers). The information from the last betting layer and the last card layer are combined and conveyed through three (3) layers of size 64 (Comb- Layers). The outputs of the last Comb-Layer are normalized to form the network outputs.

Innovation and Contribution: Multiple studies have used Neural Networks to facilitate the problem of solv-

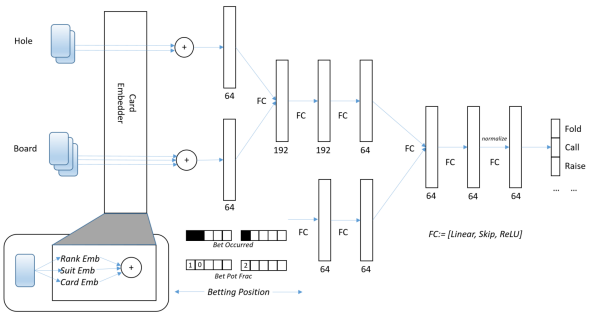

Figure 1: DEEP-CFR Network Architecture. Source: Deep Counterfactual Regret Minimization (Brown et al. 2019)

ing imperfect information games by Counter Factual Regret minimization (Moravck et al. 2017, Steinberger 2019, and Brown et al. 2019). However, the impact of the network hyper-parameters on the performance and effectiveness of these solutions is rarely investigated. This study aims to assess the impact of network hyper-parameters such as network architecture, tree traversals per iteration as well as number of iterations for which an agent is trained, in producing an AI agent using the Deep-CFR method.

We created multiple agents to play the game of Leduc (described below) to quantify these effects. For the baseline model, we used the architecture shown in Figure 1, with hidden layer of size 64. To assess the effect of changes in the size of the hidden layers on performance of the model, we implemented neural networks with hidden layer sizes of 16, 64 and 128. The results of the head to head games between AI agents with different network structures are presented in the results section.

HUNL is a large game with 10^{161} states. In order to train an agent with Deep-CFR for HUNL, multiple issues arise due to the size of the game. Since the betting size is unlimited, there are infinitely many choices an agent can make at each round. This is, of course, infeasible, as we cannot train a infinite tree model using a amount of time and memory. One possible solution is to only allow integer amount betting up to a very large integer. However, this too creates a high branching factor in the game tree with many choices for the agent, and the resulting tree, though finite, is still too large. To resolve this issue, we used action abstraction in order to reduce the size of the tree. The method works based on discretizing the best sizes and letting the agent chose from few different bet size options available. In the real game, this will not impact the strategy of the agent significantly and the agent can round the bet size to the nearest option available on the tree to make a decision.

The other issue in solving the game with the CFR method is the number of traverses per iteration of the search. With more traverses the agent converges more quickly (Brown et al. 2019), but this increases the running time of the algorithm. Another key observation is the fact that exploitability improves faster in the early stages of the training and then slows down in later iterations (Figures 2,3,4,8). In order to improve the running time of the algorithm, we implemented adaptive number of traversals per iteration, starting with a smaller number of traversals in early stages and gradually increasing the number as the agent is trained. By doing this, we were able to still get good performance, while reducing the running time quite a lot.

Finally, in order to compute the actual exploitability of the agent, a method called Best Response (BR) is used which is computationally expensive. This effect is even more pronounced in HUNL, because the large size of the game exponentially increases the computational complexity. For this implementation, we used LBR (Local Best Response) method instead, which approximates a lower bound to the BR method. This improves the run time by quite a lot. With all these hyper-parameter changes to the model, the run time improves and we were able to train an AI agent on the game of HUNL.

Test Problem: Leduc and HUNL

The game of poker has become the common measure for evaluating the proposed algorithms for solving the imperfect information game problems. In this project, the game of Leduc was fisrt used to test the performance of Deep-CFR, as a simplified version of the Texas Holdem poker. Leduc is a variant of poker played with a deck of six cards, which have two suits of three ranks each. (ref: https://github.com/lifrordi/DeepStack-Leduc) The game starts with each of the players being dealt a private card. This is followed by a round of betting. Then, a new card is dealt publicly as a community (or board) card, and there is another betting round. Finally, the players reveal their private cards. If one player's private card is the same rank as the board card, they win the game; otherwise, the player whose private card has the higher rank wins.

Heads-Up No-Limit Texas Holdem (HUNL) is a two player zero-sum game. The game is played with 52 cards (four suits, 13 ranks each). Each player receives two cards (private cards). This is followed by a round of betting, but there is no limit to the size and number of bets per round. Then three cards are dealt publicly on the board (Flop) and a round of betting follows. Then another card is dealt publicly (Turn), followed by a round of betting, another card is dealt publicly (River) and a final round of betting. The game has an extensive state size of 10^{161}, which makes it super complex for AI agents to search and possibly solve.

Implementation and Results

We used Eric Steinberger's (2019) github repository for the initial code needed to start experimenting with Deep-CFR. We set up docker to run some smaller experiments locally, and a Google Cloud high-CPU virtual machine to run larger, time consuming experiments. We were also able to run our code on Amazon Web Service(AWS) virtual machines. The agents were trained with modifications to the Deep CFR Algorithm - running a number of experiments with different hyper-parameters, including iterations trained, number of tree traversals per iteration, and metric for algorithm evaluation.

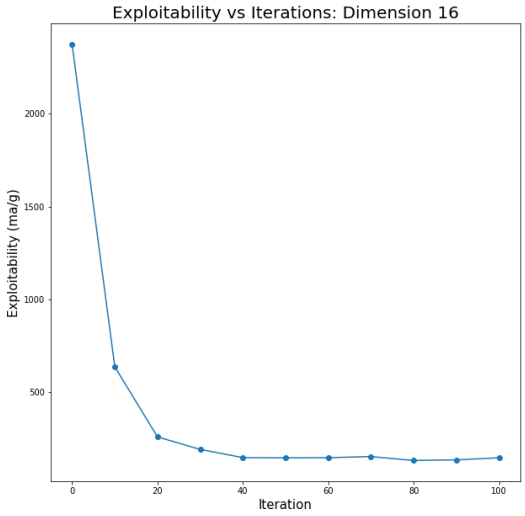

Figure 2: Exploitability- Network of 16 Nodes per Layer

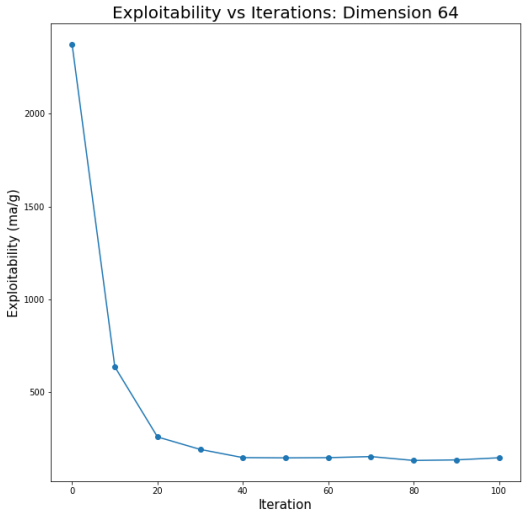

Figure 3: Exploitability- Network of 64 Nodes per Layer

As shown in Figure 2, the agent with 16 nodes per layer (details in Figure 1) was trained using Deep-CFR for 100 iterations to converge. The figure shows exploitability of the agent versus number of iterations. Exploitability is measured in milli-ante per game (mA/g), a linear metric.The agent improves drastically during the first 40 iterations and continues to improve at a slower rate. Figures 2 and 3 show the exploitability of the agent with 64 and 128 nodes. The agents with more nodes converge to the same exploitability levels.
After the agents were trained, we used a framework (Poker

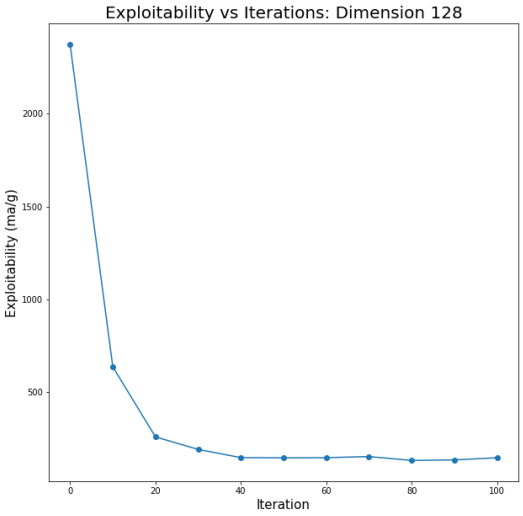

Figure 4: Exploitability- Network of 128 Nodes per Layer

RL) which allows a human to play Leduc interactively with a trained agent. We extensively studied the PokerRL python library, and decided to use the PokerRL as the framework for setting up an interactive system by modifying the code for the trained agents as well as the type of information we cache. This modification enabled us to seamlessly transfer information from the Deep CFR Trained agents to the Poker RL framework.
As a result, we trained an agent to play interactively with a human. The base code we built on top of, as well as the additional code we wrote to set up Deep-CFR, train an agent, and have it play interactively with a human, is in a private github repo (ref:https://github.com/rmahajan14/deep-learning).

We first demonstrate the effect of the number of training iterations has on the performance of the agents. We compare agents trained for 0, 10, 100 iterations. The results are plotted in Figures 5,6,7. A color of red shows represents that agent 1 won, while blue represents a win by agent 2. To evaluate these wins, and make them statistically significant, each dot represents not just 1, but 1000 games played between 2 agents, and the average number of wins and score for the games. The darker the scatter point is, the higher is the score of the x-axis agent. So, for e.g. (x, y) = (80, 20) in the graph represents 1000 game between agent 1 (trained for 80 iterations) and agent 2 (trained for 20 iterations). From the figures we can clearly see that, as expected, agents trained for a higher number of iterations perform better, and on average are able to win more games. Further, an interesting point to note is the color shift in the diagonal of the graph. The diagonal represents agent pairs with a constant difference between the number of training iterations. For e.g. (10, 20) and (80, 90) are both on the same diagonal. From the exploitability graph, we know that the amount learnt is much higher in the initial iterations, and gradually declines and saturates later.

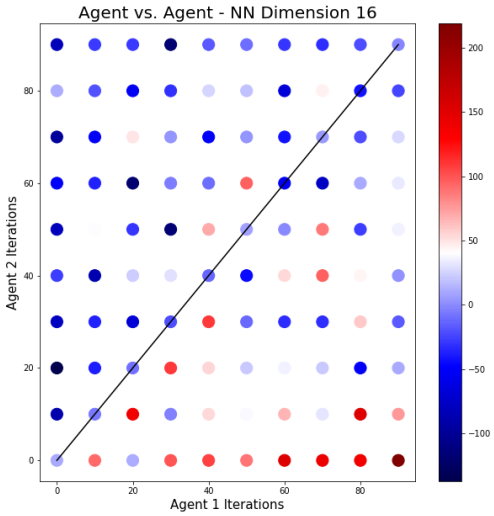

Figure 5: AI vs AI- Network of 16 Nodes per Layer

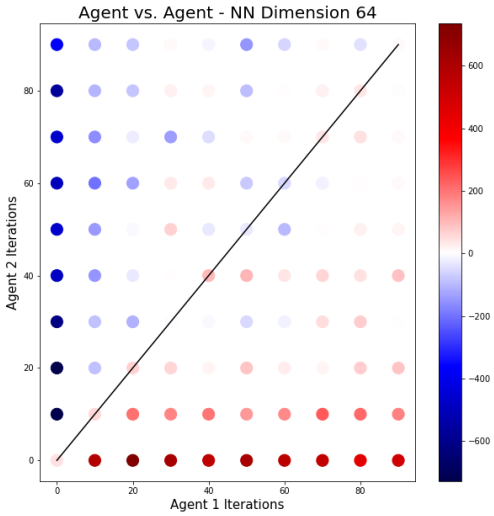

Figure 6: AI vs AI- Network of 64 Nodes per Layer

Hence, we should expect the difference between agents with iterations 10 and 20 to be much more pronounced than the difference between agents with iterations 80 and 90. In fact, we are able to verify this experimentally - going from left to right along a diagonal gradually reduces the darkness of the color, verifying the theoretical expectation.

In Figure 8, it is clearly shown that exploitability reduces with the number of iterations. In the beginning, the exploitability is quite high, and the decline is steep. As the number of runs progresses, the exploitability steepens,

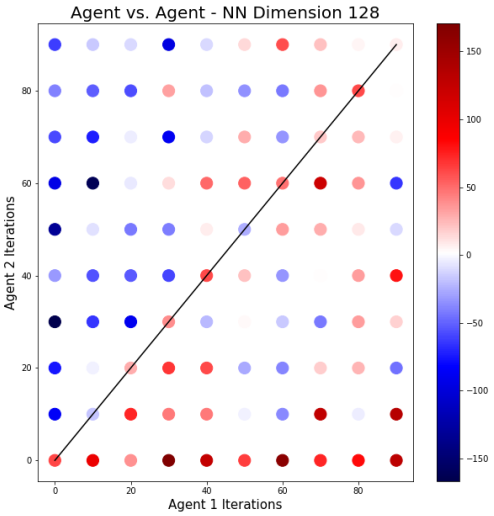

Figure 7: AI vs AI- Network of 64 Nodes per Layer

meaning that we are not learning as much anymore. Based on this, we got the idea for adaptive traversals - that is, instead of using a constant number of traversals, have a variable number of tree traversals. When we are in the initial region of the training, we can set the number of traversals to be very small, and still expect to learn a lot. When we are in the later stages of the training, we increase this parameter, to be able to improve more from one iteration.

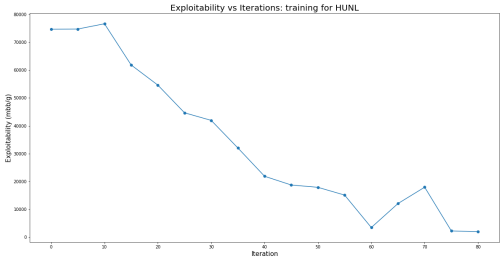

Figure 8: Exploitability of the agent for HUNL

Future Work

As mentioned in the previous sections, the main limitations for training a model for HUNL is the size of the game. So, in order to reduce the size of the game, card abstraction can be implemented. By grouping similar cards together, the state space size can be significantly reduced, while still not sacrificing performance by too much. In performing the card abstraction, One other idea is to implement tree pruning. Tree pruning is to remove, or prune parts of the tree which have low probability of being reached with prior knowledge about

the game. By implementing these two ideas, the game tree will be smaller, which is a valuable contribution to any large-size game, specifically the game of HUNL.

References

Osborne, M. J.; Rubinstein, A. 1994. "Chapter 6: Extensive Games with Perfect Information". A Course in Game Theory. Cambridge, Massachusetts: The MIT Press. ISBN 0-262-65040-1.

Roger B. Myerson. 1997. Game theory: analysis of conflict. Harvard University Press.

Brown, N. and Sandholm, T. 2018. Solving imperfect information games via discounted regret minimization. arXiv preprint arXiv:1809.04040, 2018b.

Steinberger, E. 2019. Single Deep Counterfactual Regret Minimization. arXiv preprint arXiv:1901.07621, 2019.

Yakovenko, N., Cao, L., Raffel, C., and Fan, J. 2016. Poker-cnn: A pattern learning strategy for making draws and bets in poker games using convolutional networks. In 30th AAAI Conference on Artificial Intelligence

Gibson, R., Lanctot, M., Burch, N., Szafron, D., and Bowling, M. Generalized sampling and variance in counterfactual regret minimization. In Proceedins of the Twenty-Sixth AAAI Conference on Artificial Intelligence, pp. 13551361, 2012.

Moravck, M., Schmid, M., Burch, N., Lisy, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Jo-hanson, M., and Bowling, M. 2017. Deepstack: Expert-level artificial intelligence in heads-up no-limit poker. Science.

Schmid, M., Burch, N., Lanctot, M., Moravcik, M., Kadlec, R., and Bowling, M. Variance reduction in monte carlo counterfactual regret minimization (VR-MCCFR) for extensive form games using baselines. In AAAI Conference on Artificial Intelligence (AAAI), 2019 Zinkevich, M., Johanson, M., Bowling, M. H., and Piccione, C. 2007. Regret minimization in games with incomplete information. In Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS), pp. 17291736.

Farina, G., Kroer, C., Brown, N., and Sandholm, T. Stablepredictive optimistic counterfactual regret minimization. In International Conference on Machine Learning, 2019.

Oskari Tammelin. 2014. Solving large imperfect information games using CFR+. arXiv preprint arXiv:1407.5042.

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. 2018. Deep Counterfactual Regret. Minimization. arXiv preprint arXiv:1811.00164.

Kroer, C., Waugh, K., Klnc-Karzan, F., and Sandholm, T. Faster algorithms for extensive-form game solving via improved smoothing functions. Mathematical Programming, pp. 133, 2018b.

A. Gilpin, T. Sandholm, and T. B. Srensen. A heads-up no-limit texas holdem poker player: Discretized betting models and automatically generated equilibrium-finding programs. In Proc. 7th AAMAS, pages 911:918, 2008.

Bowling, M., Burch, N., Johanson, M., and Tammelin, O. Heads-up limit holdem poker is solved. Science, 347 (6218):145149, January 2015.

Brown, N. and Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. Science, pp. eaao1733, 2017.

Noam Brown, Tuomas Sandholm. 2019. Superhuman AI for multiplayer poker. Science. DOI: 10.1126/science.aay2400.

Heinrich, J., Silver, D. 2016. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games arXiv preprint arXiv:1603.01121 (2016).

Guy Aridor, Natania Wolansky, Jisha Jacob, Iddo Drori. 2018. Implementation of Training Data Poisoning for Imperfect Information Games.
https://github.com/rawls238/LeducTrainingPoisoning

Chitsazan N. Khoshouei F. Mahajan R. 2019. Interactive Deep CFR.
https://github.com/rmahajan14/deep-learning

Steinberger, E. 2019. Scalable Implementation of Deep CFR and Single Deep CFR.
https://github.com/EricSteinberger/Deep-CFR

Steinberger, E. 2019. 2019. Framework for Multi-Agent Deep Reinforcement Learning in Poker.
https://github.com/EricSteinberger/PokerRL

NFSP

Acknowledgments

We are especially grateful to Iddo Drori for his invaluable contributions throughout the process.