Deep GXFP

Neural Network Implementation of Generalized Extensive-Form Fictitious Play

Viktor Giordano

April 2024

1 Introduction

In the study of game theory, the strategic interaction between opposing agents has long been a subject of interest. An example of such models are two-player zero-sum games. Such games are characterized by a fixed set of strategies where the gains of one player are directly proportional to the losses of the other. Applications appear in various domains, ranging from economics and military strategy to sports and beyond. Understanding such games has been a longstanding challenge, requiring innovative methodologies that blend mathematical proof with computational power. Recently, applications of artificial intelligence (AI) have risen to the forefront of the field.

When confronted with large games featuring imperfect information, such as poker variants like Texas Hold'em, the challenge becomes even more daunting. In these settings, players possess incomplete knowledge about the state of the game, such as opponents' cards. As a result, traditional approaches struggle to devise optimal strategies in the face of uncertainty and unsolvable size.

Algorithms such as Fictitious Play (Brown 1949), Counter-Factual Regret Minimization (CFR) (Zinkevich et al. 2007), and Generalized Extensive-Form

Fictitious Play (GXFP) (Schulze 1993) provide frameworks to derive solutions to many games. Deep learning adaptations of Ficitious Play, Neural Fictitious Self Play (Heinrich, Silver 2016), and CFR, Deep CFR (Brown et al. 2019), allow large games to analyzed. The use of neural networks allows for computationally efficient approximations of game values or strategies based on the game's state.

In this paper, a deep learning framework that applies notions of GXFP is introduced. Firstly, a background of key algorithms are presented. Secondly, an explanation of neural networks is provided. Finally, the novel algorithm, *Deep GXFP*, is outlined, explained, and computationally exercised in the scope of poker.

2 Background

2.1 Game Theory Algorithms

2.1.1 Fictitious Play

In an analogous way to how a human may generate a strategy for a game, fictitious play's algorithm determines the best possible response based on an opponent's average strategy and the expected reward or loss for each given action. At each time step, the acting player calculates a *best response* based on the expected utility $u^i(\sigma^i, \sigma_n^{-i})$, from both players' strategies.

$$\beta^i(\sigma_n^{-i}) \in arg\max_{\sigma^i} u^i(\sigma^i, \sigma_n^{-i})$$

The acting player then iteratively updates their average of play as,

$$\sigma_{n+1}^i = (1 - \frac{1}{n+1})\sigma_n^i + \frac{1}{n+1}\beta^i(\sigma_n^{-i})$$

Upon enough iterations, each player's average of play will converge to a Nash Equilibrium.

2.1.2 CFR

The Counter Factual Regret Minimization algorithm utilizes a notion of *regret* to update each player's strategy. At each time step, a player's regret is measured as the difference between the utility of an action, a in information set, versus the utility of a player's strategy, b^i.

$$U^i(I,a) - U^i(I, b^i(I,*))$$

Where the *action-utility* is computed as follows.

$$U^i(I,a) = \sum_{x \in I} P^{-i}(x|I) \sum_{\ell \in L_{x,a}} P(\ell|x,a)\mathcal{U}^i(\ell)$$

Where,

$$P^{-i}(x|I) = \frac{\prod_{j=1, I(a_j^x) \notin \mathcal{I}^i}^{J_x} b(I(a_j^x), a_j^x)}{\sum_{x \in I} \prod_{j=1, I(a_j^x) \notin \mathcal{I}^i}^{J_x} b(I(a_j^x), a_j^x)}$$

$$P(\ell|x,a) = \prod_{j=J_x+2}^{J_\ell} b(I(a_j^\ell), a_j^\ell)$$

The average regret weighted by the probability that the opponent reached that game state is calculated.

$$R_n^i(I,a) = \frac{1}{n} \sum_{k=1}^{n} P^{-i}(I; b_k)[U^i(I,a; b_k) - U^i(I, b_k^i(I,*); b_k)]$$

Then, the acting player's strategy is iteratively updated as,

$$b_{n+1}^i(I,a) = \begin{cases} \frac{\max(R_n^i(I,a),0)}{\sum_{a \in A(I)} \max(R_n^i(I,a),0)} \; if \; \sum_{a \in A(I)} \max(R_n^i(I,a),0) > 0, \\ \frac{1}{|A(I)|} \; otherwise \end{cases}$$

The average strategy weighted by reach probabilities will converge to a Nash Equilibrium strategy.

2.1.3 GXFP

GXFP (Schulze 1993) is similar to fictitious play; however, the strategy is updated via *best decisions* which are based off both player's strategies rather than just average plays. These best decisions are found via the action-utility found in CFR.

$$d^i(I, a; b_n) \in arg\max_a U^i(I, a; b_n)$$

At each step, the acting player's strategy is updated as,

$$b^i_{n+1} = (1 - \frac{1}{n+1})b^i_n + \frac{1}{n+1}d^i$$

It has been shown that these updates to the behavior strategy create convergence to Nash Equilibrium.

2.2 Deep Learning

In recent years, advances in AI, and specifically deep learning, have come to the forefront of our society. Large language models and computer vision are two examples that have become increasingly visible in modern life, be it in the form of self-driving cars or generative writing software. To provide a background on how the Deep GXFP algorithm functions and provides benefit over traditional algorithms, an understanding of neural networks is paramount.

2.2.1 Model Representation

When creating a neural network, a structure must be developed to receive inputs and properly classify or regress a meaningful output. The nodes, or neurons, of the network do not necessarily represent anything in the context of a game or problem. They simply exist to create connections within data that may be meaningful in predicted outputs. Input data, X, can be represented as a matrix of size $m \times n$, where m is the number of training examples and n is the number of features. Focus will be placed on fully connected linear networks, which means

each neuron in one layer is connected to every neuron in the next layer. If there are L layers in a network (including the input and output layers), denote the activations of the neurons in the i-th layer as $A^{(i)}$. The parameters of our network are the weights $W^{(i)}$ and biases $b^{(i)}$ for each layer i. $W^{(i)}$ is a matrix of size $n^{(i)} \times n^{(i-1)}$, where $n^{(i)}$ is the number of neurons in the i-th layer, and $n^{(i-1)}$ is the number of neurons in the $(i-1)$-th layer. $b^{(i)}$ is a vector of size $n^{(i)}$. Since only linear networks are considered, the output of a neuron in layer i, denoted $z^{(i)}$, is calculated as

$$z^{(i)} = W^{(i)} A^{(i-1)} + b^{(i)}$$

. These weights and biases are randomly assigned upon initialization. Deep GXFP utilizes a Xavier Uniform initialization, which randomly assigns values from $[-x, x]$ where x is defined as follows for each layer.

$$x = \sqrt{\frac{6}{n_{inputs} + n_{outputs}}}$$

Herein lies the benefit of neural networks: if the network's weights and biases can be trained to approximate a given output, then approximations on new inputs can be made using solely linear calculations and basic matrix arithmetic.

2.2.2 Forward Propagation and Activation Functions

The first step in a network's training process begins with forward propagation. Given some input value x, the activation values are calculated via the rectifier function.

$$ReLU(x) = \max(0, x)$$

$$Z^{(i)} = W^{(i)} A^{(i-1)} + b^{(i)}$$

$$A^{(i)} = \text{ReLU}(Z^{(i)})$$

Calculating this activation value for each node and layer creates a unique mapping through the network.

2.2.3 Loss Functions

To measure the performance of the model, loss functions are used on the output of the training data. Common ones that will be alluded to later include Mean Squared Error loss (MSE) and Cross Entropy loss.

$$\text{MSE}(Y, \hat{Y}) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

$$\text{CrossEntropy}(Y, \hat{Y}) = -\frac{1}{m} \sum_{i=1}^{m} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right)$$

where Y represents the true output, $\hat{Y}$ represents predicted output, and m represents the number of samples.

2.2.4 Stochastic Gradient Descent

The goal is to minimize the loss function by adjusting the weights and biases of the network. Stochastic gradient descent iteratively updates these parameters. At each iteration, the gradients of the loss function, J, with respect to the parameters are computed. These gradients are used to update the parameters with α being the learning rate between steps.

$$W^{(i)} := W^{(i)} - \alpha \frac{\partial J}{\partial W^{(i)}}$$

$$b^{(i)} := b^{(i)} - \alpha \frac{\partial J}{\partial b^{(i)}}$$

2.2.5 Backpropagation

Backpropagation allows for all parameters to be updated with one loss function. The chain rule is applied each layer to provide a gradient which is used to update the weights of each layer.

$$\frac{\partial J}{\partial (W^{(i)} \cdot A^{(i-1)} + b^{(i)})} = \frac{\partial J}{\partial z^{(i)}} \cdot \frac{\partial z^{(i)}}{\partial (W^{(i)} \cdot A^{(i-1)} + b^{(i)})}$$

Then, the following partial derivatives can be used in the SGD updating function.

$$\frac{\partial J}{\partial W^{(i)}} = \frac{\partial J}{\partial (W^{(i)} \cdot A^{(i-1)} + b^{(i)})} \cdot \frac{\partial (W^{(i)} \cdot A^{(i-1)} + b^{(i)})}{\partial W^{(i)}}$$

$$\frac{\partial J}{\partial b^{(i)}} = \frac{\partial J}{\partial (W^{(i)} \cdot A^{(i-1)} + b^{(i)})} \cdot \frac{\partial (W^{(i)} \cdot A^{(i-1)} + b^{(i)})}{\partial b^{(i)}}$$

Upon completion, the network has been trained.

2.3 Deep CFR

In games where the number of infosets (state's indistinguishable to the acting player) are beyond computational limits, deep learning methods can provide approximations much faster than tabular calculation. Deep CFR utilizes a network that attempts to predict the regret in a given infoset.

2.3.1 Algorithm

function DEEPCFR

Initialize each player's regret network $V(I, a|\theta_p)$ where θ_p are random nonzero input parameters such that all outputs are 0

for CFR iteration t=1 to T **do**

for each player p **do**

for traversal k=1 to K **do**

Perform CFR calculation with external sampling

end for

Train θ_p on MSE loss between the calculated regret and the network's predicted regret.

end for

end for

Train a strategy network, $\Pi(I, a|\theta_\Pi)$ on the MSE loss between a CFR calculated strategy and the network's predicted strategy.

end function

2.3.2 Regret Network

The regret network, $V(I, a|\theta_p)$, aims to predict an infoset's regret. The output is a measurement that is proportional to the actual regret at the given game state. This network is trained through the use of CFR calculations to collect training data. The benefit of this trained network is that regret can be approximated without the need for traversing the game tree. In games such as No Limit Texas Hold'em, game tree traversal is computationally taxing; so much so, that none of the common algorithms can solve for the optimal strategy. A linear approximation as given by the neural network allows for fast computation of a strategy.

2.3.3 CFR with External Sampling

CFR with external sampling, or Monte Carlo CFR (MCCFR), is a version of the CFR algorithm that only traverses a portion of the game tree upon each iteration. In traditional CFR, the entire game tree traversed from the acting infoset. By randomly generating a value in $[0, 1]$, the following action in the game tree is selected by the probability distribution created by each player's strategy as approximated by the strategy network. Each iteration only reaches one terminal node, so multiple iterations of the algorithm are performed and an average regret is returned.

2.3.4 Strategy Network

Recall that the average strategy over all iterations converges to a Nash Equilibrium in CFR. Thus, to create an average strategy using strictly neural network approximations each training iteration's network must be placed into memory.

In large games or experiments where multiple runs of the algorithm take place, this quickly produces storage issues. To avoid this, a separate strategy network is established and trained.

After the each run of the algorithm, the strategy network trains itself on whatever average strategy that the CFR loop calculates.

Note: The strategies used in the random decision making process in the CFR with external sampling come from this strategy network. Strategies based upon the regret that is tabulated in CFR are then used to train the strategy network.

3 Deep GXFP

Deep CFR provides a framework for applying neural networks to extensive form games such as Flop Hold'em poker. To build on this application and further test the notion of *best decisions* brought about by GXFP, an analogous algorithm is created for GXFP. The only differences between the two being the networks' targets. In Deep GXFP, the regret network is replaced by an *action-utility network*. Rather than approximating regrets, the network aims to approximate the action-utility of any action. Using this action-utility, a best decision can be deduced.

Since an optimal strategy would make the best decision in any given state, the strategy network aims to predict which action provides the highest utility. The strategy network is trained on whatever best decision is deduced from the CFR iterations. This training uses a Cross Entropy loss function, one that is specialized for classification-type models.

3.1 Algorithm

function DEEPGXFP

Initialize each player's utility network $V(I, a|\theta_p)$ where θ_p are random nonzero input parameters such that all outputs are 0

for iteration t=1 to T **do**

for each player p **do**

for traversal k=1 to K **do**

Perform calculation of utility with external sampling

and current player strategies

end for

Train θ_p on MSE loss between the calculated utility and

the network's predicted utility.

end for

end for

Train a strategy network, $\Pi(I, a|\theta_\Pi)$, on the cross-entropy loss between a calculated best decision and the network's predicted strategy.

end function

3.2 Network Architecture

Both the action-utility and strategy networks utilize the same network structure to make their respective approximations. The structure used mimics that of Deep CFR identically.

There are seven fully connected linear layers which have rectifier (ReLU) activation functions between them. All weights and biases for each node are randomly initialized according to Xavier Uniform initialization. The input is received as an *embedding*, which encodes which cards are visible to the acting player and the percentage of the pot that was just bet by the opposing player. From these two pieces, the infoset can be deduced.

The goal of the neural network is to 'learn' connecting aspects of the data through loss function and bias minimization, so to understand and draw conclusions about the game state, the card state and betting history are kept separate until layer 5 of 7. The intuition behind this separation in structure is that it

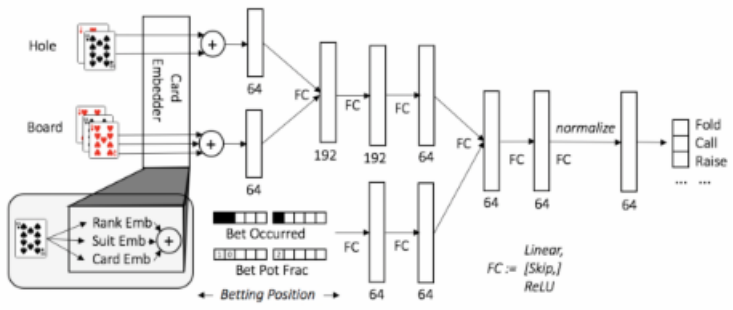


Figure 1: Deep GXFP Network

allows the network to classify the betting habits (such as potential bluffs, aggressive betting, limps or passive bets) and the card state (made hands, bad hands, and potential draws) before analyzing the overall game. The separation between these two aspects allows certain scenarios of the other to be refined. I.e.: if the opposing player's bet is weak, it may be possible to neglect the possibility that they have strong cards.

Each layer has between 64 and 192 neurons. Since these neurons do not represent anything in the game setting, the size of each layer was chosen to optimize the output. More neurons were used when understanding the game's card state, yet less were used for the betting state. There is no explicit reasoning given by Brown et al. as to how the number of layers or nodes were chosen, other than it provides more accurate results compared to other setups. A more thorough investigation as to how network architecture affects approximation accuracy for different games should be investigated.

4 Experiments

Hardware: All experiments were run on 13th Gen Intel i9-13900K @ 3.00 GHz with 96GB of RAM.

Examples with intuitive results such as values and strategies are given. Further

computations of *exploitability* are made. Exploitability measures how close a suboptimal strategy is to Nash Equilibrium.

$$Exploitability(\pi) = \frac{1}{n}\sum_{i}(\max(u_i(\pi_{-i})) - u_i(\pi))$$

4.1 Kuhn Poker

The first point of investigation of the Deep GXFP algorithm is applied to a small game, *Kuhn Poker* (Kuhn 1950). In Kuhn Poker, only three cards are used. Both player's ante 0.5 units, are dealt one card, then one round of betting ensues. Player 1 may choose to bet 1 unit or pass. If player 1 bet, then player 2 may fold (terminal) or call their bet (terminal). If player 1 passed, then player 2 may pass (terminal) or can bet 1 unit (continuation). Following a bet from player 2, player 1 may fold (terminal) or call their bet (terminal).

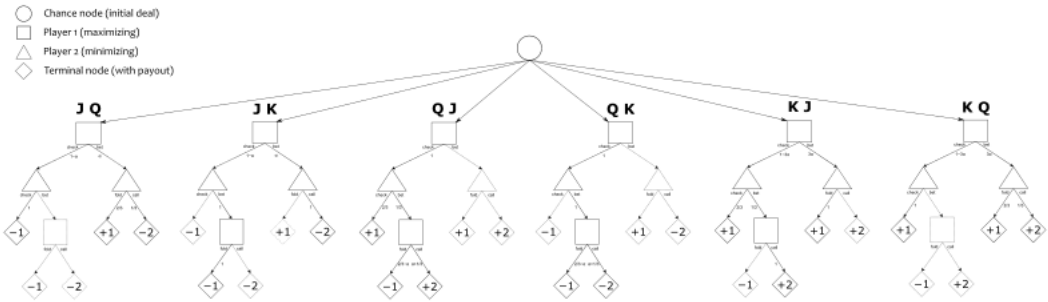
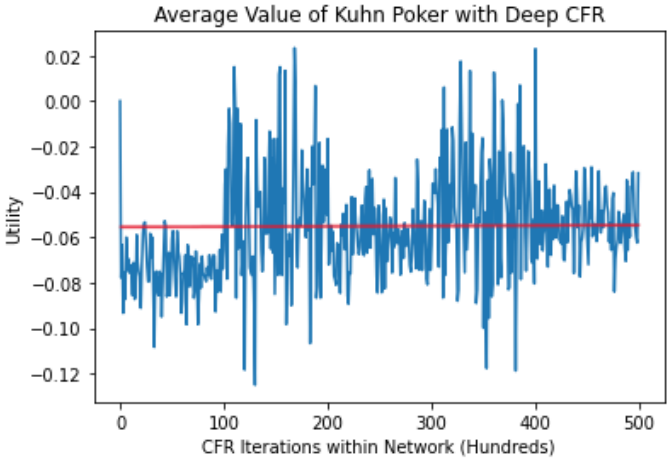


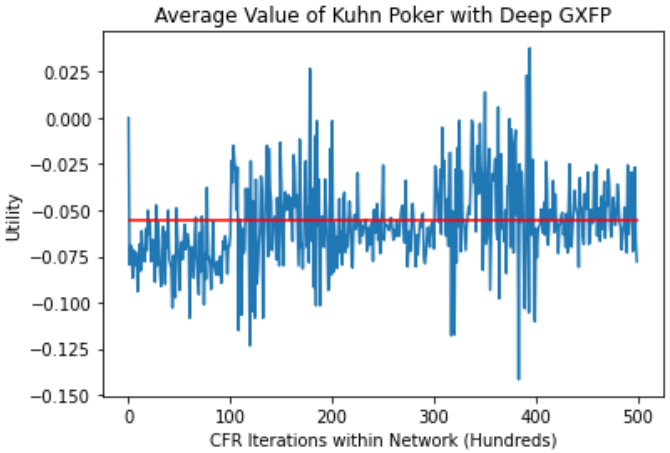
Figure 2: Kuhn Poker Tree

At Nash Equilibrium, Kuhn Poker has an average value of -0.055 from the perspective of player 1. Meaning, that if optimal strategies are used by both players, player 1 loses 1/18 of a unit each play. In such a small game, neural networks tend to over fit, or over correct their parameters, when finding solutions; however, they do approximate this Nash Equilibrium value.

For the experiment, 100 stochastic gradient descent iterations are performed per training and 1000 CFR iterations collect data for the network to train on.

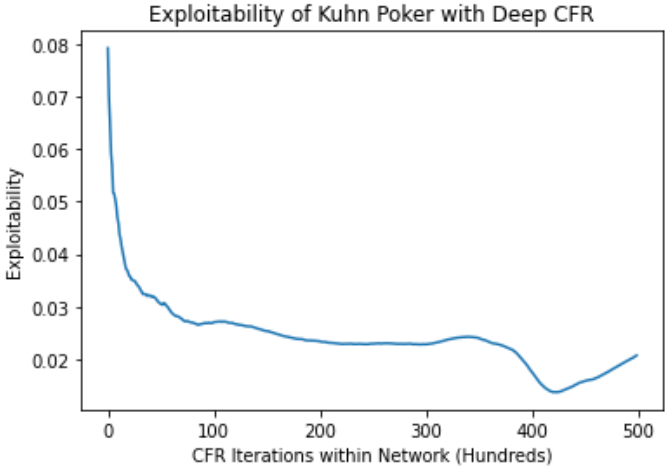


(a) DeepCFR

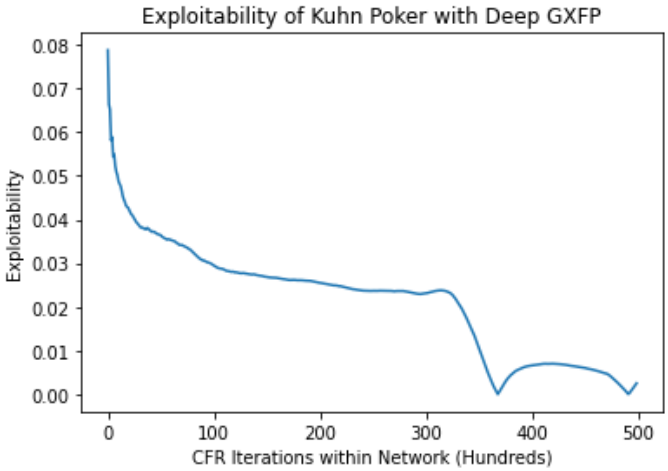


(b) DeepGXFP

Figure 3: Kuhn Poker Value



(a) DeepCFR



(b) DeepGXFP

Figure 4: Kuhn Poker Exploitability: Exploitability is measured in the game's unit. Within 50,000 iterations of the CFR training, less than 1% of an ante per game can be exploited.

4.2 Heads Up No Limit Texas Hold'em

To test the algorithm's capabilities on a large game, Heads Up No Limit Texas Hold'em (HUNLH) is used. In this alteration of Texas Hold'em, two players face each other, bet sizes have no bound, and all other rules of Texas Hold'em are assumed. To limit the number of possible actions, only 3 bet sizes are allowed: call, 1/2 pot, and 1x pot (the option to check/fold is always present). First, an intuitive example will be presented, then the *exploitability* will be determined of this strategy as it were trained. To train the network, 10,000 CFR iterations are performed each loop, then 500 stochastic gradient descent iterations follow to train the network.

In such a large game, comparing results for meaningful strategies can be rather difficult. To intuitively investigate this solver in the scale of HUNLH, a specific example is solved for strategy. Using a modern poker GTO solver, GTO+ [4], to also calculate an optimal strategy. GTO+ utilizes a version of CFR to calculate optimal strategies in specific game examples. To compare the two, the following example is established: In a \$1/\$2 small/big blind game, two players face the flop (first three community cards revealed) with \$10 in the pot. Both players have stack sizes of \$200. The acting player, the one being analyzed, is in-position (or second to act once the flop cards are revealed). The acting player holds Ace of diamonds and Queen of hearts. The flop comes out as Queen of spades, Jack of hearts, and 2 of hearts. The opposing player bets \$5, or half pot. The strategies produced are as follows:

	GTO+	DeepGXFP
Fold	0.0%	0.4%
Call	8.3%	11.2%
Raise 50%	21.0%	19.1%
Raise 100%	70.7%	69.3%
Minimal Exploitability (mbb/g)	51.2	31.9
Calculation Run Time (s)	227.36	0.001

Table 1: HUNLH Flop Example

Investigating the exploitability of the model on HUNLH, measured in milli big blinds per game (mbb/g). This measure is equivocal to $0.002, but mbb/g are typical to game theory literature so the measure is used.

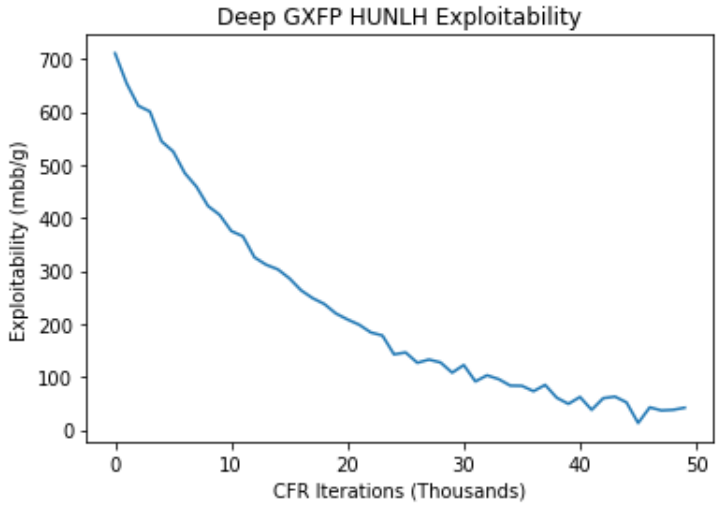


Figure 5: Exploitability of HUNLH

5 Conclusion

In the paper, a new algorithm utilizing GXFP and the neural network structure of Deep CFR is introduced. Meaningful results are given by solving the basic game of Kuhn poker, providing results equivocal to that of Deep CFR. When applied to HUNLH, intuitive strategies that align with those of a modern poker strategy optimizer are found. That strategy was then shown to converge in terms of exploitability as training of the network proceeded.

Future points of study include the optimization of the network architecture for different games. All calculations were produced on a machine without a GPU, future training with access to parallel computing or more powerful hardware could further the model's performance.

References

[1] Brown, George W. Some Notes on Computation of Games Solutions. RAND Corporation, 1949.

[2] Brown, Noam, et al. 'Deep Counterfactual Regret Minimization'. arXiv [Cs.AI], 2019, http://arxiv.org/abs/1811.00164. arXiv.

[3] Bubbabjh. 'Bubbabjh/Headsup_deepcfr'. GitHub, https://github.com/bubbabjh/HeadsUp_deepCFR.

[4] GTO+, https://www.gtoplus.com/.

[5] Heinrich, Johannes, et al. 'Fictitious Self-Play in Extensive-Form Games'. International Conference on Machine Learning, 2015, https://api.semanticscholar.org/CorpusID:13937012.

[6] Heinrich, Johannes, and David Silver. 'Deep Reinforcement Learning from Self-Play in Imperfect-Information Games'. CoRR, vol. abs/1603.01121, 2016, http://arxiv.org/abs/1603.01121.

[7] Kuhn, H. W. (1950). "Simplified Two-Person Poker". In Kuhn, H. W.; Tucker, A. W. (eds.). Contributions to the Theory of Games. Vol. 1. Princeton University Press. pp. 97–103.

[8] Schulze, Tim. 'A Generalized Extensive-Form Fictitious Play Algorithm'.

[9] Timbers, Finbarr, et al. 'Approximate Exploitability: Learning a Best Response'. Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, July 2022, https://doi.org10.24963/ijcai.2022/484.

[10] Zinkevich, Martin, et al. 'Regret Minimization in Games with Incomplete Information'. Advances in

Neural Information Processing Systems, edited by J. Platt et al., vol. 20, Curran Associates, Inc., 2007, https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf.