Tampere University

Emil Kattainen

Deep Counterfactual Regret Minimization in Continuous Action Space

Faculty of Information Technology and Communication Sciences (ITC)
Master's thesis
December 2021

Abstract

Emil Kattainen: Deep Counterfactual Regret Minimization in Continuous Action Space
Master's thesis
Tampere University
Master's Degree Programme in Information Technology
Dec 2021

Counterfactual regret minimization based algorithms are used as the state-of-the-art solutions for various problems within imperfect-information games. Deep learning has seen a multitude of uses in recent years. Recently deep learning has been combined with counterfactual regret minimization to increase the generality of the counterfactual regret minimization algorithms.

This thesis proposes a new way of increasing the generality of the counterfactual regret minimization algorithms even further by increasing the role of neural networks. In addition, to combat the variance caused by the use of neural networks, a new way of sampling is introduced to reduce the variance.

These proposed modifications were compared against baseline algorithms. The proposed way of reducing variance improved the performance of counterfactual regret minimization, while the method for increasing generality was found to be lacking especially when scaling the baseline model. Possible reasons for this are discussed and future research ideas are offered.

Keywords: Game Theory, Counterfactual Regret Minimization, Machine Learning, Deep Learning

Contents

ABBREVIATIONS AND SYMBOLS

Deep CFR Deep Counterfactual Regret Minimization.

CFR Counterfactual Regret Minimization.

CFR+ Counterfactual Regret Minimization+.

MCCFR Monte Carlo Counterfactual Regret Minimization.

NNCFR Neural Network Counterfactual Regret Minimization.

NN Neural Network.

ReLU Rectified Linear Unit.

ELU Exponential Linear Unit.

SGD Stochastic Gradient Descent.

MDN Mixture Density Network.

1 Introduction

Imperfect-information games is a model for strategic interaction between agents where only partial information is available. A typical example of such a game is poker. In contrast, perfect-information games such as chess, have all the information available for agents to use to make decisions. The lack of information in imperfect-information games makes them fundamentally more difficult in terms of finding equilibrium strategies where no agent can improve by deviating from the said strategy [1].

Counterfactual Regret Minimization (CFR) is a family of algorithms used for finding Nash equilibrium strategies for imperfect-information games. CFR has been used to reach milestones in many benchmark games such as heads-up limit Texas hold 'em [2] and no-limit Texas hold 'em [3]. Notably, these algorithms have used tabular form of CFR where the strategy is saved in a table with rows for all the possible situations the agent can find itself in. This table can become excessive when modeling real-world games. To compress the model, information abstractions are used to bucket different situations together. A problem with these abstractions is that they often require extensive domain knowledge, and the equilibrium within the abstracted game might not reflect the whole game's equilibrium accurately [4].

For reinforcement learning in perfect-information games, the need for tabular form has been eliminated by the use of deep neural networks as function approximators for the policy. This has been used to reach superhuman performance in Go [5]. The use of deep neural networks has been shown to reduce the need for domain knowledge greatly [6]. The reinforcement learning algorithms used in conjunction with deep neural networks are however mainly limited to perfect-information games as they do not converge to equilibria in imperfect-information games.

Deep neural networks have since been applied to CFR as well in the form of Deep Counterfactual Regret Minimization (Deep CFR) [4]. Deep CFR used deep neural networks to eliminate the need for information abstractions with the tabular CFR by having the networks approximate the regret value table for any state in the game. However, Deep CFR still requires the actions to be abstracted in the case of continuous actions. For example, in no-limit Texas hold 'em the player may bet any size between the minimum raise and the stack size. This range of actions has to be abstracted by choosing some number of discrete bet sizes within the allowed

sizes since the Deep CFR can only work with discrete actions.

In this thesis, different ways of handling the action abstraction in the context of Deep CFR are explored. Also, a way of reducing the variance of regret samples within Deep CFR is proposed. These different abstraction methods with and without the variance reduction are compared against each other in some benchmark games. The baseline for action abstraction is the traditional way of abstracting the continuous action space into a number of discrete sizes which cover the action space at wanted accuracy. This thesis proposes a new way of handling continuous actions by having a neural network predict a continuous distribution of values for each continuous action. This removes the last need for explicit abstraction within Deep CFR. The proposed method comes from the assumption that we can jump from discrete space to continuous by having the size of the abstraction tend to infinity. By assuming that small changes within the continuous action space lead to small changes in regret, the neural network can be trained by only sampling a number of discrete sizes within the whole continuous space. Essentially the neural network is given the job of inferring the values between the sampled ones to make handling the whole continuous action space without any abstraction feasible.

A problem within Deep CFR is the variance when sampling regret values. The proposed algorithm for reducing variance uses Monte Carlo roll-outs to get more accurate estimates of expected values for actions, thus improving the estimates for the expected value of whole information sets and regrets of individual actions within an information set.

This thesis is structured as follows. In Chapter 2 the theoretical background and related work is discussed. The relevant theory is within neural networks regarding mixture density networks and game theory about game solutions and counterfactual regret minimization and its variations. In Chapter 3 the proposed modifications to counterfactual regret minimization methods are discussed. Chapter 4 describes the test settings used to test the proposed methods against already established methods and goes over the results of these experiments. Finally, Chapter 5 concludes this thesis and discusses possible future work related to the work done in this thesis.

2 Theory and related work

In this chapter, the related theory and previous work are discussed. First neural networks and specifically the theory behind mixture density networks are reviewed. Then game theory and its basic theory are considered in more general and finally, we move toward the more specific problems and methods relevant to this thesis.

2.1 Neural networks

Neural networks (NN) are function approximators that have been found to perform well in practice from learning value functions for agents in reinforcement learning environments [6] to having superhuman performance in image recognition tasks [7]. NNs are inspired by the human brain by having artificial neurons communicate with each other. Each neuron has inputs and an output. The output of a neuron is calculated by multiplying the input by a weight, adding a bias to it, and then using an activation function on it. In practice, artificial neurons are grouped as layers and as such the output $\mathbf{y}$ of a layer can be calculated as

$$\mathbf{y} = \phi(\mathbf{W} \cdot \mathbf{x} + b), \tag{2.1}$$

where ϕ is the activation function, $\mathbf{W}$ is a matrix of weights of each neuron for each input, $\mathbf{x}$ is the input of the layer as a vector and b is the bias for the layer. This function for a single neuron is visualized in Figure 2.1. A full neural network is often constructed by first having an input layer that takes the input of the whole network. Then additional layers are stacked on top by passing the output of the previous layer as an input to the new layer. The topmost layer is called the output layer and its output is considered the output of the whole network. A network structured this way is called a feed-forward network and is the network type used in this thesis. [8][9]

2.1.1 Activation functions

Activation functions are used in-between layers as nonlinearities to enable the network to learn nontrivial functions. A common example of an activation function is rectified linear unit (ReLU). The output of ReLU is calculated as

$$y = \max(x, 0). \tag{2.2}$$

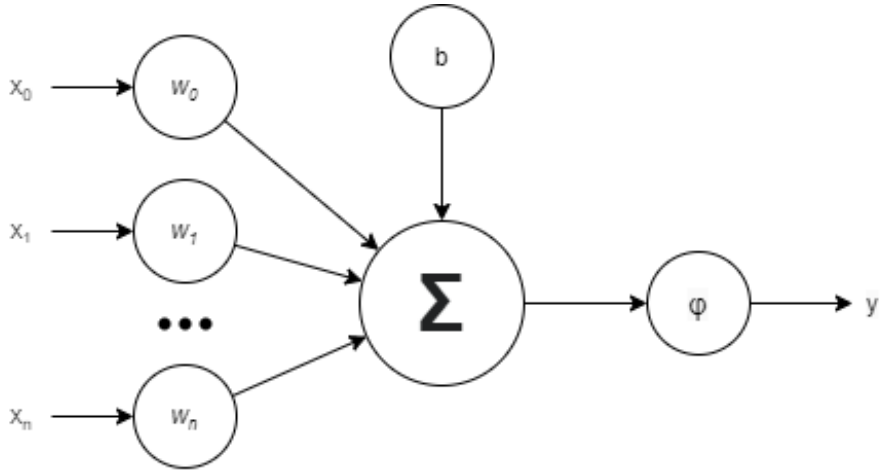


Figure 2.1 *Depiction of a single artificial neuron. Starting from the left there is the input vector visualized as x_0, x_1, and x_n. This input vector is multiplied by the weight vector $w_0, w_1, ..., w_n$ and then summed with the bias b shown at the top of the figure. Finally, an activation function is taken from this sum to produce the final output of the artificial neuron.*

Other activation functions used in this thesis are softmax and exponential linear unit (ELU). Softmax normalizes the inputs to become a discrete probability distribution with

$$y(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}, \tag{2.3}$$

where K is the number of values in the input vector. ELU is a modification to ReLU in the way that instead of having negative values be truncated to zero the output will follow an exponential curve for negative values of x. Formally

$$y = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0, \end{cases} \tag{2.4}$$

where α is a hyperparameter for controlling the value to which the output saturates to for negative inputs. [10][11]

2.1.2 Loss functions

A loss function describes the target metric that the neural network will be optimized for. The loss function is calculated per sample and then averaged over all samples. The loss function is used when training the neural network by trying to minimize the loss value over the training data. An example of a loss function is *negative log-likelihood* which is calculated as

$$L(\boldsymbol{x}, y, \boldsymbol{\theta}) = -log(p(y|\boldsymbol{x}; \boldsymbol{\theta})), \tag{2.5}$$

where $\boldsymbol{x}$ is the input vector, y is the output label and $\boldsymbol{\theta}$ are the network parameters. This loss function maximizes the probability of y given the input $\boldsymbol{x}$. [8][12] The specific loss functions used in this thesis are introduced in Section 3.2.

2.1.3 Training neural networks

To train a neural network the parameters of the network are optimized with respect to some loss function. This optimization often happens on a dataset that has input-output pairs defining desired output given an input. Optimizing a neural network with this kind of dataset is called supervised learning. [8]

Stochastic gradient descent (SGD) is an optimization algorithm used for training neural networks. SGD uses the gradient of the loss function over the neural network's weights to minimize the output of the loss function. SGD is an iterative algorithm. Each iteration a minibatch of m examples is sampled from the dataset. A gradient estimate is then calculated as

$$\hat{\boldsymbol{g}} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}), \tag{2.6}$$

where f is the output function of the neural network and $\boldsymbol{y}^{(i)}$ the desired output for the input vector $\boldsymbol{x}^{(i)}$ from the dataset. Finally, an update is applied to the network weights using the calculated gradient with

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}, \tag{2.7}$$

where ϵ_k is the learning rate at iteration k which is used to regulate the size of the updates. If the updates are too big the algorithm might diverge but if the updates are too small it might take unnecessarily long to converge. [13]

2.1.4 Mixture Density Networks

Normally neural networks predict just the maximum likelihood values for the outputs. Mixture density networks (MDN) expand on this by predicting an arbitrary distribution for the output values. The probability density function is represented as a linear combination of kernel functions in the form

$$p(\boldsymbol{t}|\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i(\boldsymbol{x})\phi_i(\boldsymbol{t}|\boldsymbol{x}), \tag{2.8}$$

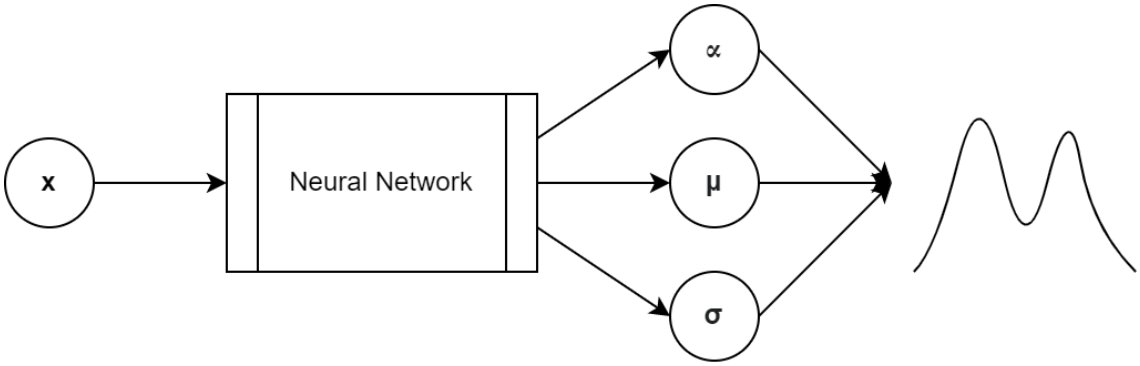


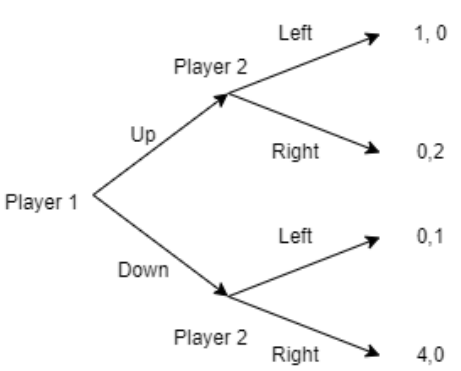
Figure 2.2 *Visualization of an MDN. Input vector $\boldsymbol{x}$ is fed into the NN. The NN outputs a vector of mixing coefficients α and multiple vectors of parameters for the component distributions. In this case, the component distributions are normal distributions so the output consists of means $\boldsymbol{\mu}$ and standard deviations $\boldsymbol{\sigma}$. Finally, these component distributions are combined to create the full mixture distribution.*

where m is the number of component distributions, $\alpha_i(\boldsymbol{x})$ is the mixing coefficient for the ith component and $\phi_i(\boldsymbol{t}|\boldsymbol{x})$ is the kernel function for the ith component. In this thesis, normal distributions are used as the kernel functions. This means that the network has to predict the mixing coefficients, the normal distribution means, and the normal distribution standard deviations for each component distribution. Figure 2.2 illustrates the whole pipeline from passing the input to the NN to combining the component distributions to the mixture distribution. The mixing coefficients must sum to zero so the softmax function is used in the network on the coefficient vector to ensure this property. Also, the normal distribution standard deviations must be positive so the ELU activation with an offset of one is used to achieve this. [14]

2.2 Game theory

Game theory is the study of games. A game describes the ways players can interact with each other and what the players ought to achieve in the game. In game theory, the players are assumed to be rational. This means the players are aware of the possible actions they can take, form expectations of unknowns, have clear preferences, and make choices after consideration. [15]

Games can be differentiated by being either strategic-form or extensive-form and being either perfect information or imperfect information. Strategic-form games model a situation in which the players choose their plan for the game once at the beginning, and the decision of each player is done simultaneously. In comparison, in extensive-form games actions are taken sequentially. A simple example of a game described in both strategic-form and extensive-form is shown in Figure 2.3. In perfect information games, every move done in the game is fully informed to all the



		Player 2			
		Left, Right	Right, Left	Left, Left	Right, Right
Player 1	Up	1, 0	0,2	1, 0	0,2
	Down	4,0	0,1	0,1	4,0

Figure 2.3 *Visualization in extensive-form and strategic-form of a simple game. Player 1 can choose to either go up or down. Player 2 can choose to go either left or right based on what player 1 chose. On the left is the tree visualization of this game as an extensive-form game. The tree starts with player 1's choice and continues with player 2's choice. On the right is the strategic-form visualization as a table. Player 1 can choose their strategy to be either going up or down. Player 2 can choose their strategy as a combination of what to do if player 1 went up and what to do if player 2 went down. For example, the left-most column labeled as "Left,Right" means player 2 chooses to go left if player 1 chooses up and otherwise player 2 chooses right. The values at the leaf nodes of the tree and in the cells in the table state the utility each player receives.*

players while in imperfect information games some of the actions may be only known by part of the players. This study focuses on extensive-form imperfect information games. It is described later in detail. [1]

2.2.1 Extensive-form imperfect information games

Extensive-form is a way to describe games as a history of sequential actions. An extensive-form game consists of the following.

- A finite set N of the players.
- A set H describing the possible histories h of the game. A history is described as a sequence of actions $h = (a^k)_{k=1,\dots,K} \in H$. A history can be called terminal if it is infinite or if there is no a^{K+1} such that $(a^k)_{k=1,\dots,K+1} \in H$. The set of available actions at a non-terminal history is denoted by $A(h)$ and the set of terminal histories is denoted by Z. An action maps the history to a child history h'.
- A player function P which outputs which player is the next to take an action after a non-terminal history. Denoted as $P(h)$. It is possible that $P(h) = c \notin N$ which means that chance determines the taken action.

- A chance function f_c which maps a probability for each action at a history if $P(h) = c$.
- For all players an information partition $\mathcal{I}_p$ of $\{h \in H : P(h) = p\}$ so that $A(h) = A(h')$ whenever h and h' are in the same member of the partition. The $I_p \in \mathcal{I}_p$ are called information sets.
- A utility function u which maps each terminal node to a payoff for each player. If there are exactly two non-chance players and holds that $u_1(z) + u_2(z) = 0$ for all $z \in Z$, the game is said to be two-player zero-sum.

In imperfect information games of important note is the fact that a player cannot know in which history they are. Instead, the player can only know in which information set they are taking an action. [1] For example, in poker there are private cards dealt to each player. A player cannot differentiate between histories where the opposing player has been dealt different cards but can differentiate between histories where they have been dealt different cards.

Player p can be said to play strategy σ_p. The probability of an individual action a is written as $\sigma(I, a)$ for information set I or as $\sigma(h, a)$ for history h. The strategies of all players in a game can be combined into a tuple of strategies called strategy profile σ. Using these definitions, reach $\pi^\sigma(h)$ can be defined as the probability that history h is reached if each player plays according to the strategy profile σ. Formally $\pi^\sigma(h) = \prod_{h' \cdot a' \sqsubseteq h} \sigma(h', a')$. Additionally, $\pi^\sigma(g, h)$ is used to represent the probability of reaching history h given history g is already reached. This is formally defined as $\pi^\sigma(g, h) = \prod_{g \sqsubset h' \cdot a' \sqsubseteq h} \sigma(h', a')$. The player reach $\pi^\sigma_p(h)$ is the product of the probabilities of the player p choosing all the actions which lead to the history h. Formally $\pi^\sigma_p(h) = \prod_{h' \cdot a' \sqsubseteq h | P(h') = p} \sigma(h', a')$. Player reach is also defined for information sets as $\pi^\sigma_p(I) = \prod_{I' \cdot a' \sqsubseteq I | P(I') = p} \sigma(I', a')$. External reach $\pi^\sigma_{-p}(h)$ is the product of the probabilities of actions leading to h which are external to player p. This includes the probability of players other than p choosing the actions which lead to h and the probability of the chance actions happening which lead to h. Formally, this is defined as $\pi^\sigma_{-p}(h) = \prod_{h' \cdot a' \sqsubseteq h | P(h') \neq p} \sigma(h', a')$. External reach is also defined for information sets as $\pi^\sigma_{-p}(I) = \sum_{h \in I} \pi^\sigma_{-p}(h)$. [16][17]

2.2.2 Game solutions

In a game, the goal for the player is to maximize the utility function. The utility of a strategy profile σ for player p can be written as $u_p(\sigma)$. The problem with an

individual player maximizing their utility is that the utility depends on the entire strategy profile. This means that what the other players choose as their strategies, changes the expected utility for the individual player. If we assume the opponent's strategy is not known and that the opponent can play any possible strategy, the player can approximate Nash equilibrium to find a good strategy. Nash equilibrium σ^* is a strategy profile in which any player cannot improve their utility by unilaterally changing the strategy. In two-player zero-sum games, this means that if one is playing a Nash equilibrium strategy the other player cannot decrease the utility gained by one's strategy. This makes playing a Nash equilibrium strategy optimal for two-player zero-sum games if one does not have any knowledge of the opponent. [18] Nash equilibrium is defined formally in Section 2.2.9.

2.2.3 Regret and regret-matching

Regret is a concept used in online learning for extensive games. Regret is accumulated over multiple times of playing an extensive game. Average overall regret [19] for player p at time T is

$$R_p^T = \frac{1}{T} \max_{\sigma_p^* \in \Sigma_p} \sum_{t=1}^{T} (u_p(\sigma_p^*, \sigma_{-p}^t) - u_p(\sigma^t)). \tag{2.9}$$

Non-negative regret is calculated as

$$R_{+,p}^T = \max(0, R_p^T). \tag{2.10}$$

Regret-matching uses the average overall regret at time T to create a strategy for time $T+1$. The strategy for each pure strategy s is calculated as

$$\sigma_p^{T+1}(s) = \frac{R_{+,p}^T(s)}{\sum_{s' \in S} R_{+,p}^T(s')}. \tag{2.11}$$

This regret-matching strategy is used in regret minimization algorithms [20].

2.2.4 Counterfactual regret minimization

In counterfactual regret minimization (CFR) the overall regret is decomposed into terms which can be minimized independently for each information set. These individual regrets are minimized iteratively. Counterfactual utility used in defining

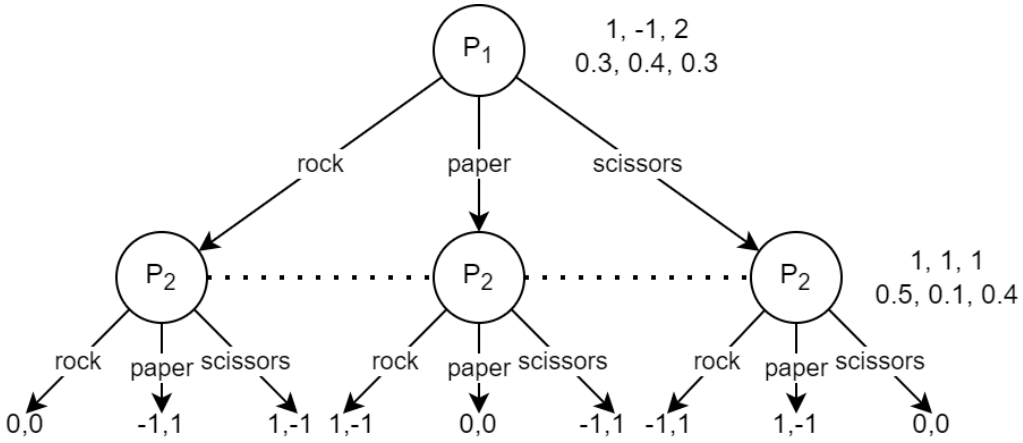


Figure 2.4 *Visualization of a hypothetical run of CFR on the game of rock paper scissors. Rock paper scissors can be formalized as an extensive-form game by having P_1 choose their action in secret and announcing the result after P_2 has chosen their action. This way after P_1 has chosen their action P_2 will be in an information set of three nodes—one for each choice P_1 has. This is visualized with the dotted line between P_2 nodes. On the right, there are multiple vectors of numbers to represent the cumulative regret and the average strategy tracked by CFR for each of the information sets. The top vector of three numbers is for the cumulative regret which is used to calculate the current strategy. The bottom vector is for the average strategy which will be the final strategy calculated by CFR.*

counterfactual regret is defined as

$$u_p(\sigma, I) = \frac{\sum_{h \in I, h' \in Z} \pi^{\sigma}_{-p}(h)\pi^{\sigma}(h, h')u_p(h')}{\pi^{\sigma}_{-p}(I)}. \tag{2.12}$$

This is used to calculate immediate counterfactual regret over all iterations up to T as

$$R^T_{p,imm}(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \pi^{\sigma^t}_{-p}(I)(u_p(\sigma^t|_{I \to a}, I) - u_p(\sigma^t, I))), \tag{2.13}$$

where σ^t is the current strategy at iteration t, $u_p(\sigma^t|_{I \to a}, I)$ is the utility of choosing action a at information set I and otherwise using the current strategy σ^t. This can be thought of as the regret for the player in each information set using the counterfactual utility weighted by counterfactual probability of reaching I if the player would have tried to do so. [21] The immediate counterfactual regret is tracked for each information set I and action a as

$$R^T_p(I, a) = \frac{1}{T} \sum_{t=1}^{T} \pi^{\sigma^t}_{-p}(I)(u_p(\sigma^t|_{I \to a}, I) - u_p(\sigma^t, I))). \tag{2.14}$$

Non-negative regret can be calculated for this in the same way as for ordinary regret in Equation 2.10. Regret-matching is then used to generate current strategy for $T+1$

as

$$\sigma_p^{T+1}(I,a) = \begin{cases} \frac{R_{+,p}^T(a)}{\sum_{a' \in A(I)} R_{+,p}^T(a')} & \text{if } \sum_{a' \in A(I)} R_{+,p}^T(a') > 0, \\ \frac{1}{|A(I)|} & \text{otherwise.} \end{cases} \tag{2.15}$$

This makes actions to be selected in proportion to the positive counterfactual regret for playing that action. If all actions have negative regret the strategy can be chosen arbitrarily, though normally the strategy is chosen to be uniform over all the actions. [19]

Average strategy is

$$\bar{\sigma}_p^t(I,a) = \frac{\sum_{t=1}^T \pi_p^{\sigma^t}(I)\sigma^t(I,a)}{\sum_{t=1}^T \pi_p^{\sigma^t}(I)}. \tag{2.16}$$

This average strategy approximates Nash equilibrium when updated through self-play using the current strategy based on the counterfactual regret. The self-play is done iteratively. Each iteration the whole game tree is traversed and the immediate counterfactual regret is updated for all actions in all information sets in the game tree. These iterations are done until an average strategy of wanted quality is achieved. The quality of the average strategy is often quantified using exploitability discussed in Section 2.2.9. A visualization of CFR is shown in Figure 2.4. [19]

Monte Carlo CFR

Monte Carlo CFR (MCCFR) is a version of CFR in which the whole game tree is not traversed at each iteration. Instead, only parts of the tree are sampled. The sampling policy used has to ensure that the immediate counterfactual regrets are unchanged in expectation. Let $\mathcal{Q} = Q_1, ..., Q_r$ be a subset of Z, such that the union of all Q_i spans the whole set Z. These subsets are called blocks. Each iteration, one of these blocks is sampled and only terminal histories in this one block are considered. Each block has a probability $q_j > 0$ associated with it. This probability dictates the chance of sampling this block on a single iteration. Let $q(z) = \sum_{j:z \in Q_j} q_j$ be the probability of sampling a terminal history z on a single iteration. The sampled counterfactual value on updating block j is

$$\tilde{v}_i(\sigma, I|j) = \sum_{z \in Q_j \cap Z_I} \frac{1}{q(z)} u_i(z)\pi_{-i}^{\sigma}(z[I])\pi^{\sigma}(z[I], z). \tag{2.17}$$

This counterfactual value is then used similarly to vanilla CFR to update the regrets using $\tilde{r}(I,a) = \tilde{v}_i(\sigma^t_{(I \to a)}, I) - \tilde{v}_i(\sigma^t, I)$. Notably, it can be seen that choosing $\mathcal{Q} = Z$, i.e., one block with all the terminal histories and sampling probability $q_1 = 1$ is equal to vanilla CFR. [22]

There are various ways of sampling used with Monte Carlo CFR. One is outcome-sampling MCCFR. Outcome-sampling MCCFR chooses the blocks so that each block has just one terminal history. The distribution for sampling blocks is defined using a sampling profile σ', so that $q(z) = \pi^{\sigma'}$. Each iteration, this sampling profile is used to first traverse the terminal history forward to calculate for each player i the probability of reaching each prefix of the history for that player $\pi_i^\sigma(h)$ and then backward to calculate each player's probability of reaching to the end of the sampled terminal history $\pi_i^\sigma(h, z)$. When doing the backward traversal the regret values are updated with

$$\tilde{r}(I, a) = \begin{cases} w_I \cdot (1 - \sigma(a|z[I])) & \text{if } z[I]a \sqsubseteq z, \\ -w_I \cdot \sigma(a|z[I]) & \text{otherwise,} \end{cases} \tag{2.18}$$

where

$$w_I = \frac{u_i(z)\pi_{-}^{\sigma} i(h)\pi_i^\sigma(z[I]a, z)}{\pi^{\sigma'}(z)}. \tag{2.19}$$

Another way of sampling used is external-sampling MCCFR. In external-sampling MCCFR the player's own actions are not sampled but the ones external to the player are. This way we have a block $Q_\tau \in Q$ for each pure strategy at the opponent and chance nodes. The block probabilities are defined by the chance function f_c and the opponent's strategy σ_{-i} as

$$q_\tau = \prod_{I \in \mathcal{I}_c} f_c(\tau(I)|I) \prod_{I \in \mathcal{I}_{Ni}} \sigma_{-i}(\tau(I)|I). \tag{2.20}$$

All terminal histories z consistent with τ are included in a block Q_τ, as in if an action taken at a point in history ha is a prefix of the terminal history z so that $h \in I$ for any $I \in \mathcal{I}_{-i}$ then $\tau(I) = a$. The point in using these block probabilities is that it results in $q(z) = \pi_{-i}^\sigma(z)$. Each iteration a block is sampled implicitly for each player $p \in N$ by traversing the tree in post-order depth-first fashion while sampling actions at h where $P(h) \neq p$. At each visited information set the sampled counterfactual regrets are computed as

$$\tilde{r}(I, a) = (1 - \sigma(a|I)) \sum_{z \in Q \cap Z_I} u_i(z)\pi_i^\sigma(z[I]a, z) \tag{2.21}$$

and the computed values are updated to the total regrets. [23]

CFR+

CFR+ is a modification to CFR which instead of using regret-matching uses regret-matching+ to improve convergence and enables better compression of the tabular data stored when executing the CFR algorithm. Regret-matching+ uses cumulative counterfactual regret+ instead of regular cumulative counterfactual regret. Using the definition of counterfactual utility $u_p(\sigma, I)$ from Equation 2.12 cumulative counterfactual regret+ can be defined as

$$R_i^{+,T}(I,a) = \begin{cases} \max\{u_p(\sigma^T_{I\to a}, I) - u_p(\sigma^T, I), 0\} & \text{if } T = 1, \\ \max\{R_i^{+,T-1} + u_p(\sigma^T_{I\to a}, I) - u_p(\sigma^T, I), 0\} & \text{otherwise,} \end{cases} \tag{2.22}$$

where T is the current CFR iteration, σ^T is the strategy at the current iteration and $\sigma^T_{I\to a}, I)$ is strategy taking only the action a. Regret-matching+ is calculated in similar fashion to regular regret-matching in Equation 2.15 by just replacing the non-negative regrets with the cumulative counterfactual regret+. This form of cumulative regret resets the regret to zero whenever it becomes negative. [24] In regular CFR the regret can grow to be negative indefinitely. In CFR+ having all these regrets set to zero reduces the entropy and allows for more efficient compression of the data. Also, a benefit of not having largely negative regrets is that if an action starts to get positive regret it will start seeing use faster when it does not have to first negate the negative regret accumulated so far. CFR+ also has a guarantee that the current strategy profile will converge or almost converge to an approximate Nash equilibrium on itself and the use of average strategy is not necessary. This can be used to save memory during run time at the cost of time to converge. [2]

2.2.5 Abstractions

Games which are desired to be studied using CFR can be really large because of the exponential growth that happens when the game is extended by a single node. Because of this studied games are often abstracted to fit in smaller spaces. Two ways of abstracting a game are information and action abstractions. Information abstractions are about merging multiple information sets into one. Action abstractions restrict the number of actions the player can make in each information set. [25] Abstractions often require extensive domain knowledge or can be made with domain agnostic abstraction finding algorithms. Extensive domain knowledge can be hard to come by and the domain agnostic abstraction finding algorithms can miss relevant aspects of the game in question and thus not be able to compress the game enough or produce abstractions that make the solution perform worse in

the full game. [4] Abstractions can also have abstraction pathologies which cause abstractions with strictly more information available to the player in any abstracted information set to perform worse in the full game than the one with strictly less information [25]. Thus not having to deal with abstractions can make it easier to apply CFR algorithms to a variety of games.

2.2.6 Reservoir sampling

The basic idea in reservoir sampling is to have a sample of size $S \geq n$ from which a random sample of size n is to be generated. The limitations are that we can only go through the original sample of size S once and that the size S is not known beforehand. This means we cannot just draw random records from the middle and instead we will have to process the records in a sequence. For an algorithm to be considered a reservoir algorithm it has to maintain the reservoir as a true random sample of the original sample after processing each new record from the original sample.

Initially, the n first records are put to the reservoir. After that when processing each new record the algorithm has to choose whether to replace a record in the reservoir with the new record or not. With the reservoir algorithm used in this thesis when considering the $(t+1)$st record when $t \geq n$, it has a $n/t+1$ chance of replacing a random record in the reservoir. [26]

2.2.7 Deep CFR

A problem in traditional CFR algorithms is the need to store the regret and average strategy values for each information set explicitly. This problem is normally relieved by the use of abstractions. These abstractions can be hard to come by as often domain knowledge of the game is required. There also exists abstraction finding algorithms but these are often too general for specific games and hence miss important nuances in the game. In deep CFR this problem is tackled by using neural networks to approximate the values while learning the abstraction implicitly within the neural network model.

On each iteration in deep CFR, there are K partial traversals of the game tree done for each player p. The player p is called the traversing player. The way the traversals are done is determined by external-sampling MCCFR. The strategy $\sigma^t(I)$ played at each information set is determined using regret-matching on the output of a neural network $V : I \rightarrow \boldsymbol{R}^{|A|}$ defined by parameters $\theta_p^{(t-1)}$. The input to the

neural network is the information set I and the output is $V(I, a|\theta^{t-1})$. The goal is to have $V(I, a|\theta^{(t-1)})$ approximately be proportional to the total regret $R^{(t-1)}(I, a)$ that traditional CFR would have computed.

At a terminal node, the value of the node is passed up. At chance and opponent nodes the value is passed up unchanged. At the traversing player's nodes, the value is passed up as the weighted average of all the actions weighted by the action probabilities of each action. Formally this value is defined as

$$\tilde{v}_i(\sigma^t, I) = \sum_{a \in I} u_i(a|I)\sigma^t(I, a). \tag{2.23}$$

The immediate regret for each action can be then calculated as

$$\tilde{r}(I, a) = \tilde{v}_i(\sigma^t_{(I \to a)}, I) - \tilde{v}_i(\sigma^t, I). \tag{2.24}$$

These immediate regrets are stored as samples in memory $M_{v,p}$ using reservoir sampling.

When these traversals are done for a player, a value network is trained to get the parameters $\theta_p^{(t)}$. The network is trained by minimizing MSE between the network's prediction $V(I, a|\theta(t-1))$ and the samples of immediate regrets in the memory.

In addition to the value network, a separate network $\Pi : I \to \boldsymbol{R}^{|A|}$ is used for approximating the average strategy over all iterations. This policy network's strategy is the one that converges to a Nash equilibrium. To train this network policy samples are stored whenever an information set belonging to a player other than the traversing one is visited. The policy sample is the vector of probabilities over all the actions in the information set given by the value network on that iteration. These samples are stored in policy memory M_Π with a weight t.

The training of this policy network is not necessary as to get the average strategy all the value networks can be stored and used at inference time by first sampling a value network and then computing the strategy using the sampled network. This eliminates the approximation error brought by the use of an extra network. In this thesis, the policy networks are not trained and during inference time the networks are sampled. This is because of the lack of extra approximation error and the lower memory usage by not having to store the policy samples. [4]

2.2.8 Linear CFR

Linear CFR is a modification to vanilla CFR where each iteration t is weighted by t [27]. This is used in deep CFR for faster convergence. With deep CFR when training the models the overall batch is weighted by $2/T$ to not have the weighted error grow infinitely. [4]

2.2.9 Exploitability

Exploitability of a strategy in a two player game is the amount the strategy loses against its worst case opponent as opposed to what a Nash equilibrium loses against its worst-case opponent. This worst case opponent is called best response and player p's best response to σ_{-p} is marked as $BR(\sigma_{-p})$. Formally, holds $u_p(BR(\sigma_{-p}), \sigma_{-p}) = \max_{\sigma_p'} u_p(\sigma_p', \sigma_{-p})$. In a Nash equilibrium σ^* every player is playing a best response to each other. Using the best response the exploitability e of player p can be written as

$$e(\sigma_p) = u_p(\sigma_p^*, BR(\sigma_p^*)) - u_p(\sigma_p, BR(\sigma_p)). \tag{2.25}$$

Exploitability is tracked as the total exploitability which is the sum of exploitabilities of all players

$$\Sigma_{p \in P} e(\sigma_p). \tag{2.26}$$

This is the value reported when talking about exploitabilities of different algorithms later on in this thesis.

While exploitability in a game tree with only discrete actions is possible to calculate given the tree is not too large, the same is not trivial when the game includes continuous actions. With continuous actions, one would have to sample an infinite number of actions in a single decision point involving a continuous action, which is not possible. Normally this is avoided by calculating the exploitability within the action abstraction used in the evaluated model. However, due to the proposed system in this thesis being abstractionless this is not possible. Because of this, the proposed system is only evaluated in relation to other models. [23]

3 Proposed system

The deep CFR algorithm removes the need for information abstractions as the neural network learns the abstraction implicitly instead. The proposed system in this thesis builds on the deep CFR algorithm and removes the need for action abstraction as well by the use of mixture density networks for continuous actions. This modified system is called continuous deep CFR. In continuous deep CFR nodes with continuous actions are split into two nodes. First is the initial node which will have all the discrete action options. For example, if we have a node in a poker game where the action options are fold, call and raise with raise being a continuous action, the initial node will have the action options fold, call and raise. Then the second node will follow the raise action. In this second node, the player has the option of choosing any available raise size. Splitting the node this way does not change the game in itself and allows isolation of choosing the value for continuous action independent of the rest of the tree which simplifies the model. For the initial node normal deep CFR is used, but for the second node, deep CFR is modified to fit continuous action space.

In addition to continuous deep CFR, a way of reducing variance in regret samples within deep CFR is proposed. It uses Monte Carlo roll-outs to get multiple samples of an action's value. Each time an action is chosen when traversing the tree some number of roll-outs are played and the values received from each roll-out are averaged to get a more accurate value for each action. These propositions are discussed in more detail in this chapter.

3.1 Continuous actions within CFR

Normally when using CFR-based algorithms for games with continuous actions, an action abstraction is used. This way, instead of having an action for which one can choose a value from a continuous interval, one will have a list of discrete actions each of which represents a single choice within the continuous interval. These have been shown to be potentially problematic by causing abstraction pathologies [25]. To combat this, a way of computing CFR over continuous actions without abstraction is proposed.

The base idea behind continuous deep CFR in this thesis is to think of having an action abstraction with n choices for the action value. Then have $n \to \infty$ and

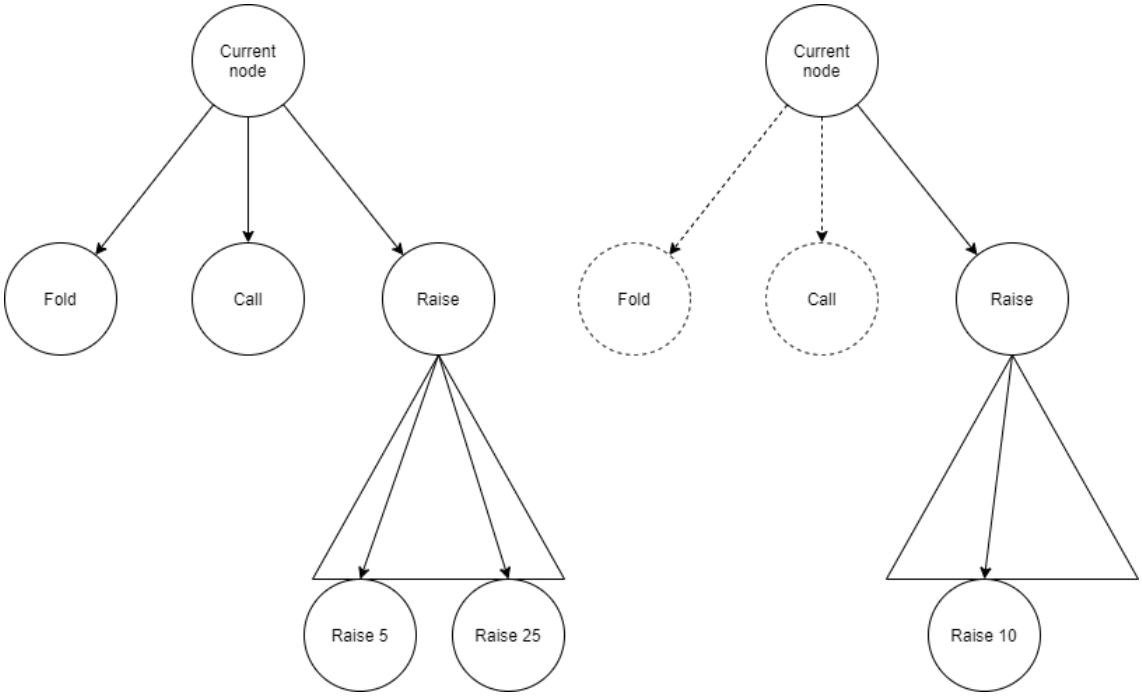


Figure 3.1 *Visualization of how a single node is traversed with continuous actions. On the left is an example of the traversing player choosing all actions and sampling two different sizes for the raise action. On the right, the non-traversing player chooses only the raise action and a single size for the raise size. The dashed lines and nodes mark that the action or node was not chosen.*

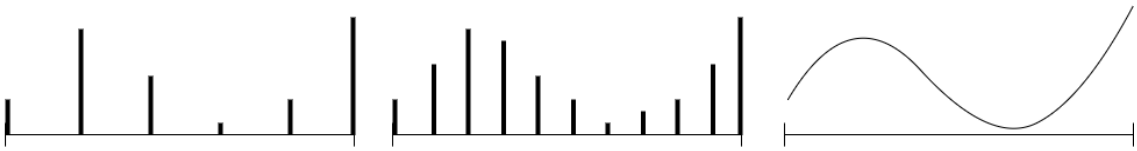


Figure 3.2 *Figure depicting cumulative non-negative regret for a continuous action with different levels of abstraction. The leftmost one shows six bars; one for each abstracted value within the actions value interval, the middle one shows a more fine-grained abstraction with 11 values within the interval. The rightmost one shows how the non-negative regret would look like if there were infinite values in the abstraction, or no abstraction was used.*

perform CFR for this theoretical abstraction. To perform CFR over this abstraction the nodes with continuous actions are split into multiple nodes such that for each continuous action a new node is created. In the original node, the player has to choose only over which type of action it has to perform. An example of this split can be seen in Figure 3.1 where the ”Current node” has the actions ”Fold”, ”Call” and ”Raise” with ”Raise” being a continuous action. Then the continuous action ”Raise” leads to a node where the player has to choose a single value within a continuous interval of values. Splitting the nodes this way avoids having to have probability distributions over a mix of discrete and continuous choices which simplifies the algorithm significantly.

While for the "Current node"'s actions we can use regular CFR-based algorithms, for the node after the "Raise" action a modification is needed. In CFR the regrets for all actions have to be tracked and then a current strategy has to be possible to be calculated from the regrets using regret-matching. With continuous action space, we can think of tracking the regrets as a function that will give the cumulative regret over the action space for any value. If we only keep track of non-negative regrets like in CFR+ we can have the function be non-negative as well. Then if we normalize the integral of this non-negative function to be one, we can use it as a continuous probability density function. If we compare this to the idea of having an action abstraction where the number of actions $n \to \infty$, we can see these two are analogous in terms of tracking the regrets and then performing regret matching to get the current strategy. To illustrate this, Figure 3.2 shows how a continuous action can be abstracted with increasing granularity. With finite choices within the abstraction, the regret matching is calculated by normalizing the sum of all the action choices to be one. This produces a discrete probability distribution over the action choices within the abstraction. With the abstraction with infinite choices i.e. no abstraction at all, the cumulative regret can be seen as a continuous function which—when normalized to have an integral of one over the action interval—will define a continuous probability distribution via a probability density function.

To collect cumulative regret within regular CFR, each iteration T an expected value for each action in each information set is calculated. Based on these values and the strategy used at iteration T, an expected value for each information set is calculated. By comparing the expected values of individual actions within an information set to the expected value of the whole information set, a regret value for each action is calculated. These regret values are added to the cumulative regret over all iterations until iteration T. When moving to an abstraction with infinite choices, it becomes impossible to calculate an expected value and consequently a regret value for each individual action choice. If we assume that a small change in the action choice brings only a small change in the expected value we can sample only a small set of action choices within the whole interval and then estimate expected values for the other action choices based on the samples. Also, we can estimate the expected value of the whole information set based on these samples. This way we can calculate estimates for the regret values using the estimated expected values over the whole action interval and the approximate expected value of the information set. The implementation of this will be discussed later in Section 3.2.

In regular CFR, traversals over the whole game tree are performed to execute the regret calculations. In continuous deep CFR, the traversals will be done over the modified game tree where the continuous actions are split into their own information

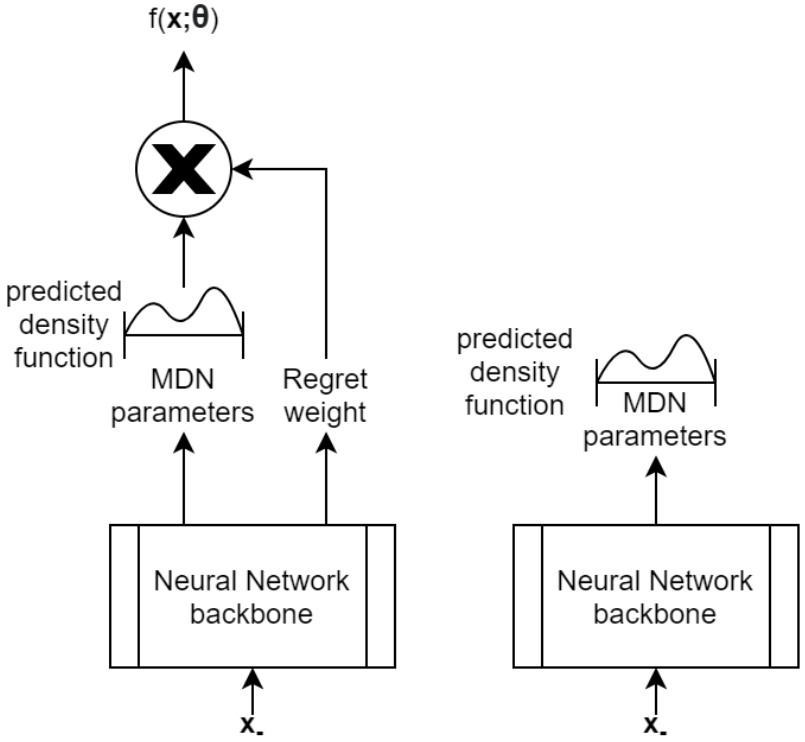


Figure 3.3 *Visualization of systems tried out for predicting the continuous probability distributions for actions. The system on the left includes the network predicting a regret weight which is used to multiply the density function so the regret can be predicted directly. The system on the right is simplified by removing that regret weight because the network is trained to predict the density function directly instead of explicitly approximating the cumulative regret.*

sets as described before. Also, the whole tree will not be traversed each time but a form of MCCFR will be used. For nodes that do not contain continuous actions, external sampling will be used. Figure 3.1 visualizes this for player nodes with dashed and non-dashed lines for non traversed and traversed actions respectively. In chance or nature nodes one of the actions is sampled each traversal based on the chance function f_c. For the continuous action nodes, the external sampling cannot be directly used as it requires the traversing player to choose all the actions at each node. Hence based on the previously discussed ideas of only sampling a few action choices and deducing the rest of the values based on the sampled values, the traversing player will just sample a number of action choices for each continuous action. The non-traversing player will be the same and sample just one action choice within the interval of the continuous action. All the sampled action choices are sampled based on the current strategy profile. This way, for the traversing player the expected values received for each action choice can just be averaged with arithmetic mean without any weighting to receive an estimate of the expected value of the whole node.

3.2 Neural network for predicting probability distributions for continuous actions

To implement continuous deep CFR, a way of predicting continuous functions over the action space is needed. These functions also need to be such that those can be normalized to have an integral of one over the action space and the normalized function has to be able to be sampled as a continuous probability distribution. For this job, two different systems based on MDNs were tried out. The first one was used initially during research but was found to be inferior to the other one and was discarded before the final experiments of this thesis. The first model adheres more strictly to the idea of first predicting the regrets over information sets and then subsequently getting the current strategy as a byproduct of the regrets. The second one tries to keep the prediction of the regrets as a more of an implicit idea behind how the current strategy is constructed and predicts the current strategy directly. Previously we discussed how only some samples of the whole action space were evaluated directly and the values for other parts of the space were extrapolated. In both systems, the neural network is let to do the estimation implicitly.

Figure 3.3 shows the structure of the neural network used in the first system. A core of MDN is used and the density value of the MDN is multiplied with a positive value predicted by the neural network to get the final predicted cumulative regret value. When the current strategy is to be predicted, the underlying MDN can directly be used. The full loss function to train this network is

$$L(\boldsymbol{x}, y, \boldsymbol{\theta}) = \frac{2t}{T}(f(\boldsymbol{x}; \boldsymbol{\theta}) - y)^2, \tag{3.1}$$

where $\boldsymbol{x}$ is the input vector, y is the sampled regret during traversal, $\boldsymbol{\theta}$ is the neural network parameters, t is the iteration the regret sample was created, T is the current iteration and $f(\boldsymbol{x}; \boldsymbol{\theta})$ is the predicted cumulative regret by the neural network. This network can only predict non-negative regret values. This comes from the idea that when generating the current strategy, the negative values will not be used anyways, and that it is easier to implement the probability distribution generically when there are no negative numbers. When training this network all negative values of regret are set to zero. This idea was from CFR+ where negative regrets are set to zero. Having to completely ignore the negative values with this system was problematic and hence it was discarded.

The structure of the second version of the system which was used in the final experiments is depicted in Figure 3.3. This system removes the final layer of scaling the density function with a positive value and instead is a bare MDN. The regret

values are not actually needed anywhere but when training the neural network, and hence the ability for the network to predict them could be removed. By removing the need to predict the regret values, the loss function could also be fit to take into account the negative regret values. The full loss function used for this system is

$$L(\boldsymbol{x}_-, x_{action_choice}, y, \boldsymbol{\theta}) = -\frac{2t}{T} p(x_{action_choice}|\boldsymbol{x}_-; \boldsymbol{\theta}), \tag{3.2}$$

where $\boldsymbol{x}_-$ is the input vector without the action value, $x_{actionvalue}$ is the action choice and $p(x_{actionvalue}|\boldsymbol{x}_-; \boldsymbol{\theta})$ is the probability of a certain action choice given the rest of the input vector and the network parameters. This loss function, in essence, is a negative log-likelihood function with weighting by the amount of regret in the sample. This loss function will make actions with positive regret more likely and actions with negative regret less likely. The function learned by this neural network is hypothesized to be analogous with doing regret matching on the regret-like function trained with mean squares error in regular deep CFR for discrete actions.

The probability functions of the neural networks described provide a probability function over all real numbers. Because to guarantee that a Nash equilibrium exists for a game the action space has to be non-infinite or compact [1], we limit ourselves to games where the action space is compact i.e. all the continuous actions limit the values to closed and bounded intervals. This means the probability distribution provided by the neural network has to be modified to have it span the same interval as the action space. Consider the sampling strategy for an information set with a continuous action a with values ranging $[a_{min}, a_{max}]$. First, the MDN is sampled for a value over all real numbers. This value is then truncated within the interval $[a_{min}, a_{max}]$ by just setting values larger than a_{max} to a_{max} and values smaller than a_{min} to a_{min}. This way the network will predict a distribution $p(t|\boldsymbol{x}_-)$ with the distribution truncated within the interval such that $p(t = a_{min}|\boldsymbol{x}_-) = p(t < a_{min}|\boldsymbol{x}_-)$ and $p(t = a_{max}|\boldsymbol{x}_-) = p(t > a_{max}|\boldsymbol{x}_-)$.

3.3 Combining the whole system together

In previous sections, the modifications to CFR required by continuous deep CFR were talked about in a general way and a way for implementing the function approximation by the use of MDNs was presented. In the following paragraphs, the implementation of continuous deep CFR will be presented concretely. Continuous deep CFR works very similarly to deep CFR with just slightly altering the game tree by splitting the continuous actions into their own nodes and then using the MDN proposed in Section 3.2 for handling the new nodes.

Algorithm 1 Continuous Deep Counterfactual Regret Minimization

function C(O)NTDEEPCFR
  Initialize each players strategy to be uniform over all actions.
  Initialize reservoir-sampled memories for regrets $M_{V,1}$ and $M_{V,2}$.
  for CFR iteration $t = 1$ to T **do**
    for each player p **do**
      for traversal $k = 1$ to K **do**
        TRAVERSE$(\emptyset, p, \theta_1, \theta_2, M_{V,p}, t)$
      end for
      Train θ_p
    end for
  end for
end function

In practice when handling a node during traversal the split is done very dynamically. First, the strategy for the node with the discrete actions is calculated using regret-matching on the output of the value network $V : I \rightarrow \boldsymbol{R}^{|A|}$ from regular deep CFR. Then if the current player in the node is the traversing player, for all the continuous action child nodes, a number of samples of the action choice are drawn using the MDN. Next, the actions in the node with the discrete actions are iterated over, and for non-continuous actions, the value of the action is calculated in a similar fashion to deep CFR. When a continuous action is iterated over the value of the action is calculated as the average of choosing the n samples previously drawn. In addition to estimating the value of the action this way, regret values for these samples are calculated. Finally, the value of the node is calculated as a weighted average of all the actions and the regret values for the discrete actions are calculated. The regrets for the actions in the discrete node and the samples in the continuous node are saved into memory and the value of the full node is returned. If the current player is not the traversing player, only one of the actions in the discrete node is chosen based on the probabilities given by the current strategy from regret-matching on the output of the value network. If the sampled action happens to lead to the continuous node only one sample within the action space is drawn. Also, the regret values will not be calculated nor saved into the memory.

Besides for the way the nodes are handled, the algorithm is very similar to deep CFR. Each iteration there are k traversals. The players take turns in being the traversing player. After all traversals for a player are done, a new value network and an MDN are trained for the traversing player. The full algorithm is detailed in Algorithm 1.

Algorithm 2 CFR Traversal with continuous deep CFR sampling

function T(R)AVERSE($h, p, \theta_1, \theta_2, M_V, t$)
  if h is terminal **then**
    return the payoff to player p
  else if h is a chance node **then**
    $a \sim \sigma(h)$
    return TRAVERSE($h \cdot a, p, \theta_1, \theta_2, M_V, t$)
  else
    Compute network outputs $V(I(h), a|\theta_{3-p})$ and
    get the strategy for discrete actions using regret-matching
    if $P(h) = p$ **then**
      Sample the MDN n times to get $\boldsymbol{c}_a = \{c_1, c_2, ..., c_n\}$ for
      each continuous action
      $\boldsymbol{a} \leftarrow A(h)$
    else
      Sample the MDN once to get $\boldsymbol{c}_a = \{c_1\}$ for each continuous action
      Sample an action from $A(h)$ according to $V(I(h), a|\theta_{3-p})$ and
      set it to $\boldsymbol{a}$
    end if
    for $a \in \boldsymbol{a}$ **do**
      if a is continuous **then**
        initialize $v_a(i)$
        for c_i in $\boldsymbol{c}_a$ **do**
          $v_a(i) \leftarrow TRAVERSE(h \cdot c_i, p, \theta_1, \theta_2, M_V, t)$
        end for
        $v(a) \leftarrow mean(v_a(i))$
        for c_i in $\boldsymbol{c}_a$ **do**
          $\tilde{r}_c(I, a) \leftarrow v_a(i) - v(a)$
        end for
      else
        $v(a) \leftarrow TRAVERSE(h \cdot a, p, \theta_1, \theta_2, M_V, t)$
      end if
    end for
    if $P(h) = p$ **then**
      for $a \in A(h)$ **do**
        $\tilde{r}(I, a) \leftarrow v(a) - \sum_{a' \in A(h)} \sigma(I, a') \cdot v(a')$
      end for
      Insert the information set and it's regrets to memory M_V
      return $\sum_{a \in A(h)} \sigma(I, a) \cdot v(a)$
    else
      return $\sum_{a \in A} v(a)$
    end if
  end if
end function

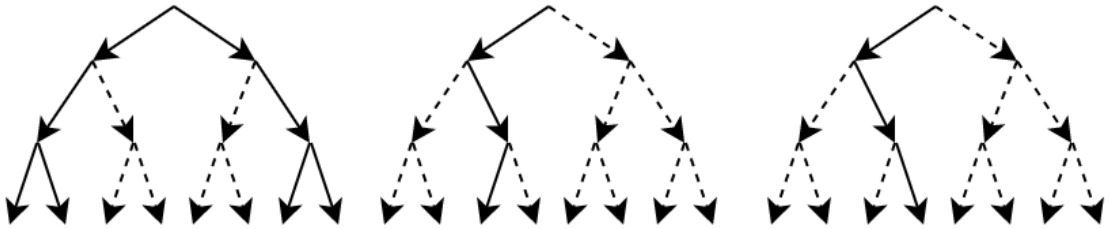


Figure 3.4 *Visualization of Monte Carlo roll-outs. All the graphs are rooted after the action for which the expected value is to be estimated. The left-most graph depicts the normal traversal when using external sampling. At the nodes where the traversing player takes an action, all the actions are sampled. At the other nodes, only one of the actions is sampled. The other two graphs show the additional roll-outs used for variance reduction. These roll-outs always sample just one of the actions because these are just for estimating the value of the action and no part of the CFR algorithm is run within these traversals.*

3.4 Monte Carlo roll-outs

Monte Carlo CFR suffers from variance and because of that for example requires special methods to make it viable when using CFR+ with it [28]. In this thesis, a way of reducing the variance of MCCFR when used within deep CFR is proposed. It uses Monte Carlo roll-outs starting from an action taken to better the estimated expected value for said action. Figure 3.4 illustrates how the roll-outs work.

When traversing a game tree at an information set using deep CFR with external sampling, the value of an action is estimated with a single traversal to the end of the game. While this traversal samples all the actions at the traversing player's nodes, only a single action is sampled at the other nodes. This causes the traversal to not touch all the nodes due to chance. In turn, the value estimates given by this sampling scheme will depend on which nodes happen to be traversed. By doing more traversals rooted after the action we can cover more of the tree and thus get a better estimate of the expected value. CFR is not run within these additional roll-outs and thus the additional roll-outs do not need to use the sampling scheme used with CFR. Because of this, these additional roll-outs can just sample one action at each node to have them run faster. When sampling an action during these additional roll-outs, for player nodes the player's current strategy is used, and for chance nodes, the chance function is used. Using this sampling, the expected value received from the roll-outs stays the same as with the traversal done with the external sampling. Once all the roll-outs are done, the sampled expected values can be averaged to get a lower variance estimate of the expected value for the action. This estimate can then be used for calculating the other values needed in CFR.

The proposed method is somewhat similar to Neural Network Counterfactual Regret

Minimization (NNCFR) [29] where Monte Carlo roll-outs are used to estimate the value of actions. The difference is that in NNCFR the estimations are used during the calculation of the strategy used at this iteration while in this thesis the proposed way is to just use the roll-outs to lower the variance of the samples recorded in the regret memory while keeping the strategy evaluation the same as original deep CFR.

4 Experiments and results

In this chapter, the experimental setup and the results from the experiments are presented. First, the games used to evaluate different algorithms are described in detail. Then the proposed systems are presented and the details of the systems are laid out. The baseline models, which the proposed systems are compared against, are presented. Finally, the results from the experiments are presented and discussed in detail.

4.1 Games

Two different games were used in the experiments. The first one is preflop no-limit hold'em which has a smaller game tree as all players' actions are limited to one round of betting. The second one is flop no-limit hold' em, which is expanded from the first game with an additional betting round. These two games were chosen to have games with a smaller and a bigger game tree to see how the proposed systems scale when the size of the game increases. There have been previous studies that have tried their proposed algorithms only in really small toy games and it has later been found out that the algorithms do not scale well to larger benchmark games [4].

Preflop no-limit hold 'em is a two-player zero-sum game. The rules of the game are the following.

- The game is played with a standard 52-card poker deck.
- At the start of the hand, both players have a stack of 100 chips. One of the players is the small blind who must place 1 chip in the pot, and the other player is the big blind who must place 2 chips in the pot.
- Both players are dealt two cards from the deck in private so that the other player may not see those cards.
- The small blind will start the action by choosing to fold, call or raise. After the small blind has taken an action the players will take turns in taking one of the actions until the game or betting has ended.
- Folding will result in the folding player losing whatever they put in the pot and the other player getting all the chips in the pot and ending the game.

- Calling means putting the same amount of chips as the other player has put in the pot and raising means putting more chips in the pot than what the other player has put in the pot.
- If a player calls and both players have acted, the betting will end.
- When a player raises, the raise must be at least for what was the difference between the last two raises or one big blind i.e. two chips, depending on which is bigger.
- The blinds act as the first two raises so if for example the small blind raises to six chips the big blind has to raise by at least four which is the difference between the previous raises six and two.
- Finally, the winner of the game is the player who can construct the best five-card poker hand when combining their two private cards and the five community cards. The winner will win the whole pot. If both players have equally strong hands, the pot is split evenly.
- After a game is played the players will change roles and the small blind will become the big blind and vice versa. Also, the chip stacks are reset to 100 chips.

Flop no-limit hold 'em is a slightly modified version of preflop no-limit hold'em. After the betting round, instead of dealing five community cards, only three cards are dealt. After the community cards are dealt there is an additional betting round starting with the big blind player. After the second betting round is done there are no additional cards dealt and the winner is determined by the best five-card poker hand.

4.2 Network architecture

All the neural networks in this thesis share the backbone used. A visualization of this backbone is shown in Figure 4.1. The output of this backbone is then fed to a head which is different between different networks. For neural networks used with regular deep CFR or for discrete actions, the head will be just a fully connected layer with the length of the output vector matching the number of actions used. For the network used to predict probability distributions for continuous actions, the output will be fed to a fully connected layer that outputs parameters for the MDN. The MDN is parameterized by the distribution parameters and the weights for individual distributions. The component distributions used are normal distributions

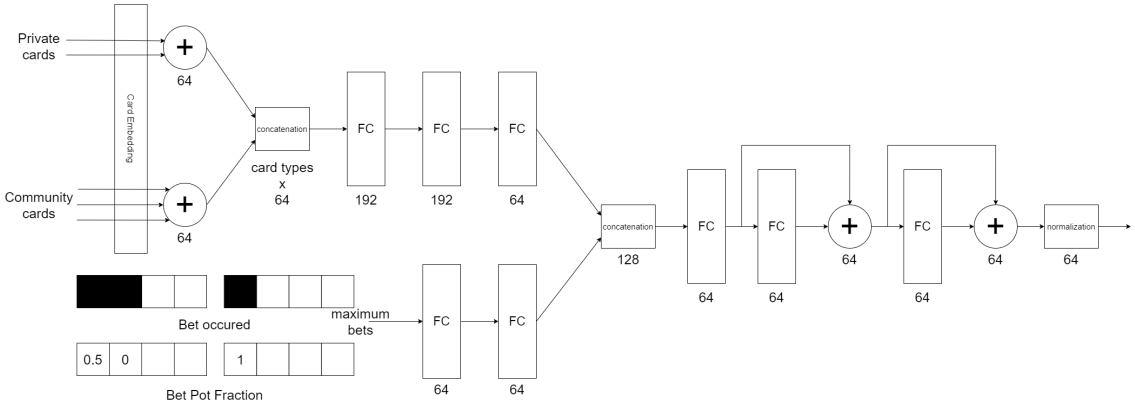


Figure 4.1 *Visualization of the backbone neural network architecture used for all different systems. The input consists of the cards dealt to players and the bet history. Embeddings are used for the cards. The network uses mainly fully connected layers with some of those having skip connections. The number under each module marks the length of the output vector for the module.*

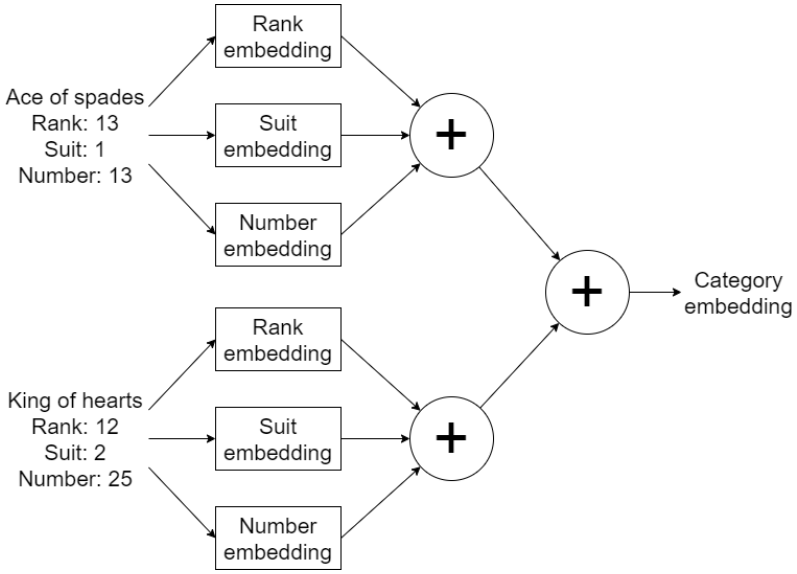


Figure 4.2 *A graph visualizing the calculation of representation for a single card category. A single card is represented as three numbers telling the rank, suit and the number of the card. An embedding is made out of each of these numbers and then the embeddings are summed over the cards in a category to get the final embedding for a single category.*

parameterized by their mean and standard deviation. This means that for each component distribution there are three parameters. In the experiments, the number of component distributions used is ten.

The information set is represented by the cards dealt and the betting actions happened so far. The betting history is given as two vectors. The first vector consists of zeros and ones marking whether the action at a point in history happened. The second vector tells the size of the bet as a fraction of the pot. If a player calls the size of the bet is set to zero. The maximum number of bets used is six. If a player

is considering their sixth action and a raise is possible, the raise will be forced to be all-in. The cards dealt are split into their categories based on the type of the cards. The private cards are in their category and the community cards are in their own. An embedding is made for each category. The embedding is made by summing card embeddings. The card embedding is made for each card by creating an embedding for the rank, the suit and the number of the card and summing these individual embeddings together. A graph for how this presentation is calculated for a single category is shown in Figure 4.2.

4.3 Model training

The neural network training was done using Adam optimizer. Adam optimizer is a modification to standard stochastic gradient descent which modifies the size of the updates based on previous updates. This has been shown to improve convergence speed in practice. [30] The learning rate used for Adam was $1e-4$ and the beta parameters were $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The batch size used for network training was $5,000$ and each network was trained for $3,000$ iterations. During training time the dataset was shuffled each time it was gone through in training.

The deep CFR algorithm was run for around 100 iterations for each model. The maximum memory size for reservoir sampling was $40,000,000$ samples. Each iteration there were $4,000$ traversals done for both players. For the models using Monte Carlo roll-outs, 2 additional roll-outs were done at each player action in addition to the regular traversal. When sampling continuous actions with the proposed modifications to deep CFR, there were three samples taken for each continuous action.

4.4 Baseline models

Two baseline models using regular deep CFR are used. These models differ only in the granularity of the abstraction used for the game tree. The abstraction was only used for the raise actions within the game tree and the abstraction was defined in relation to the pot size and the remaining stack size of the players. The main baseline model used an abstraction where the raise size was limited to a half pot, a full pot and an all-in size. When considering a raise action the pot size was calculated as what it would be if the raising player just called. For example, if the player had previously raised to 10, and the other player raised to 20, then the next raise would be calculated based on the pot being 40. This would make a half pot raise be a raise to 40. The all-in size means that the player raises all the chips

remaining in the stack. The model with the smaller abstraction had just the option of using the full pot size raise. The main baseline model is used for comparison in both games against all proposed systems and the smaller baseline model is used only in the preflop no-limit hold'em against the combination of all proposed methods.

In addition to these details, the baseline models use deep CFR with the same training hyperparameters as described in Section 4.3. Also, the network architecture used for the value network is the one described in Section 4.2 for regular deep CFR. The strategy network is not used and instead the average strategy is evaluated by sampling the value networks with linear weighting based on the iteration of the value network.

4.5 Results

The performance of continuous deep CFR with and without Monte Carlo roll-outs was compared against the baseline models by having the models play a million game match against each other. Continuous deep CFR with Monte Carlo roll-outs and deep CFR with Monte Carlo roll-outs are referred to as continuous deep CFR MCRO and deep CFR MCRO respectively. The performance was measured in milli big blinds per game (mbb/g) which is a standard measure of win rate in poker. Positive numbers correspond to the measured model to be better and negative numbers mean that the baseline model is better. The performance was measured at different points of training the proposed models so the performance is plotted over iterations of training. Also, the strategies of different models are looked at in a case-study fashion. In addition to the models described previously in this chapter strategy produced by CFR using [31] is shown and compared to the ones produced of variants of deep CFR.

In Figure 4.3 the performance of models in preflop no-limit hold'em. Both baselines with small and large action abstractions were used in this game. The small action abstraction was only compared to the combination of continuous action modifications and Monte Carlo roll-outs. The large action abstraction was compared to models with the Monte Carl roll-outs and with and without the continuous action modifications. Deep CFR with the continuous action modifications manages to beat the small abstraction but does not manage to learn a strategy good enough for facing the large abstraction. This is quite likely caused by the model with the small abstraction having trouble extrapolating responses against the single size in its abstraction. The large abstraction has multiple sizes and can thus interpolate responses in-between ones in its abstraction instead of extrapolating with only a single sample. Deep CFR with the Monte Carlo roll-outs added seems to just perform

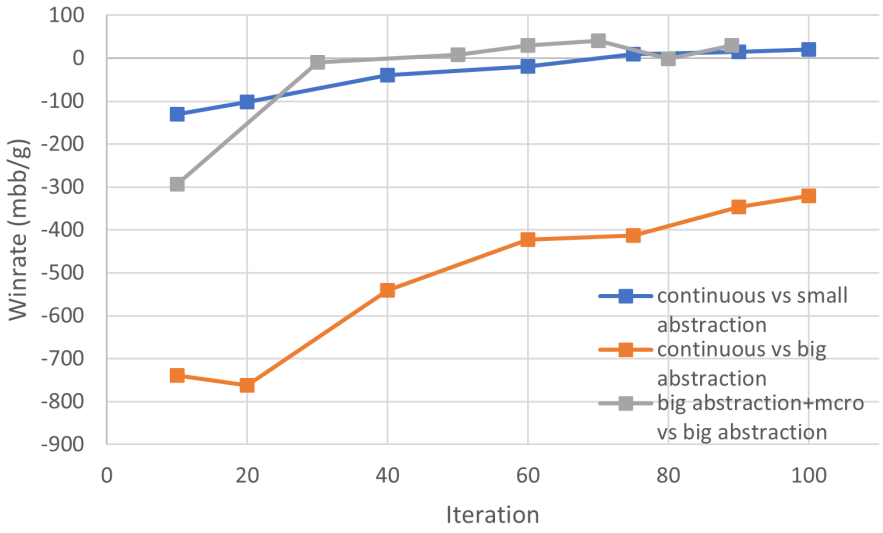


Figure 4.3 *Performance of continuous deep CFR MCRO and deep CFR MCRO plotted as a function of iterations. Performance is measured as win rate in milli big blinds per game. The performance is compared against deep CFR with large and small action abstractions. The blue line corresponds to continuous deep CFR MCRO against deep CFR with the small abstraction, the orange line corresponds to continuous deep CFR MCRO against deep CFR with the large abstraction and the grey line corresponds to deep CFR MCRO with large abstraction against deep CFR with large abstraction. A negative win rate means the baseline model is performing better while positive values mean that the measured model is performing better.*

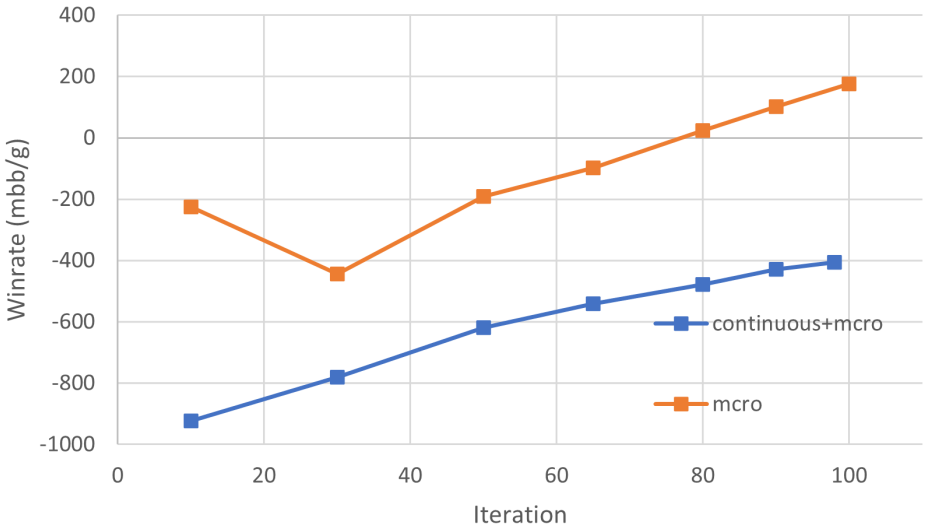


Figure 4.4 *Performance of continuous deep CFR MCRO and deep CFR MCRO against deep CFR plotted in milli big blinds per game as a function of iterations. The blue color corresponds to continuous deep CFR MCRO and the orange color corresponds to deep CFR MCRO.*

strictly better than the regular deep CFR. Note that the version with Monte Carlo roll-outs was only trained for 89 iterations because of memory issues in the training system so it did not quite have the time needed to completely converge.

In Figure 4.4 are performance graphs of the models in flop no-limit hold'em. Deep CFR with the continuous action modification and Monte Carlo roll-outs and regular deep CFR MCRO were compared to regular DCFR without any enhancements. Both regular deep CFR models used the bigger action abstraction. Can be seen that continuous deep CFR does not manage to reach the level of deep CFR with just the large action abstraction for continuous actions. Adding Monte Carlo roll-outs, however, does seem to make the deep CFR algorithm converge to a better strategy.

The addition of Monte Carlo roll-outs possibly improved the performance of the model when trained for the same number of iterations. The continuous action modification was good enough to beat deep CFR with the small abstraction at least in the smaller game but giving deep CFR an action abstraction comparable in branching to the modified deep CFR the regular deep CFR managed to out scale the modified version. Reasons and future research ideas related to this performance difference are discussed in Chapter 5.

Figures 4.5, 4.6 and 4.7 visualize strategy produced by continuous deep CFR MCRO, deep CFR MCRO and CFR. Can be seen that there are similarities but the deep CFR-based algorithms mix the actions more than the regular CFR. This can probably be explained by the use of neural networks adding noise to the training process. Also, the expected values of raising or calling the hands which are pure raises are relatively close in the strategy produced by regular CFR. This causes the noise coming from using neural networks to make the differences in strategies look much bigger than it actually most likely is in terms of expected value.

When looking at the strategy produced with the continuous action modifications, it can be seen that the minimum size is very often used in comparison to the other models for both of the visualized nodes. Especially in the second node, the all-in size is used very often by the other models but for some reason, continuous deep CFR doesn't find the all-in size. This difference seems to be the biggest difference between what the strategy would be expected to be and what the proposed model produces. Possible reasons for this and future research ideas on fixing this are discussed in Chapter 5. Of interesting note is the fact that the deep CFR systemically does not learn to fold any hands in the second node. This could be related to the relatively small amount of traversals used per iteration in this thesis.

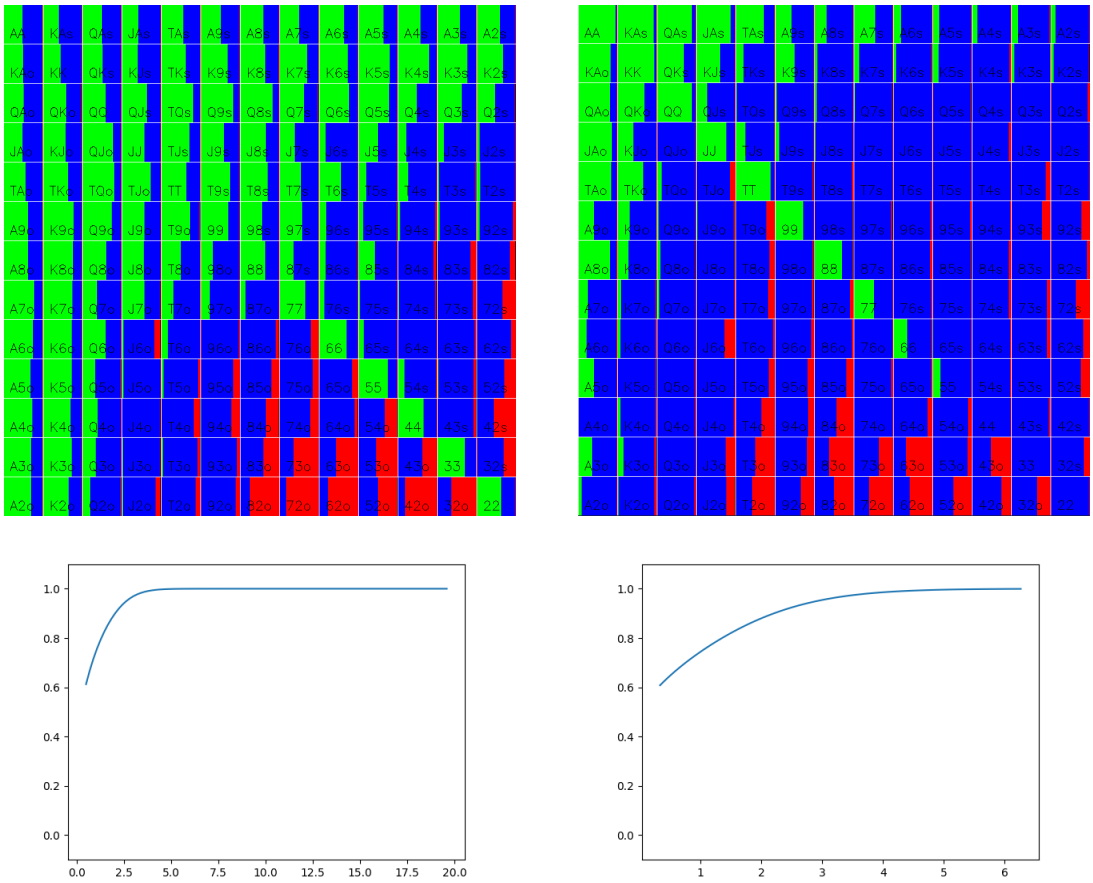


Figure 4.5 *Visualization of strategy produced by continuous deep CFR MCRO for two nodes in preflop no-limit hold'em. The strategy over the discrete choices is visualized for each individual starting hand type. The starting hands are grouped by the ranks of the two cards and by whether the cards share their suits or not. This a standard way grouping poker hands when there are no community cards dealt because the strategies of hands grouped this way are identical. The nodes visualized are the first player node and the node that follows when the small blind raises pot in this node. The colors green, blue and red correspond to the raise, call and fold actions respectively. For each hand type there is a square which is colored with these colors in proportion to the strategy used of that hand type. In addition to visualization of the discrete choice strategy, a weighted average over all hands of the cumulative density function for the raise action is shown. The cumulative density function is plotted as a function of the raise size in relation to pot size.*

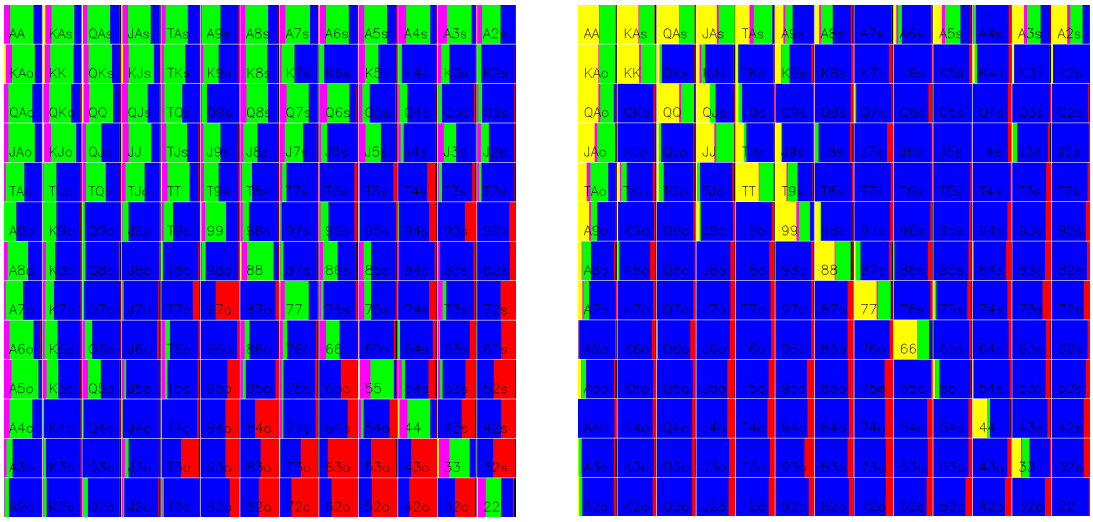


Figure 4.6 *Visualization of strategy produced by deep CFR MCRO. The nodes visualized are the same as in Figure 4.5 and the format is similar but with more action types. The colors yellow, pink, green, blue and red correspond to actions raising all-in, raising half pot, raising full pot, calling and folding respectively.*

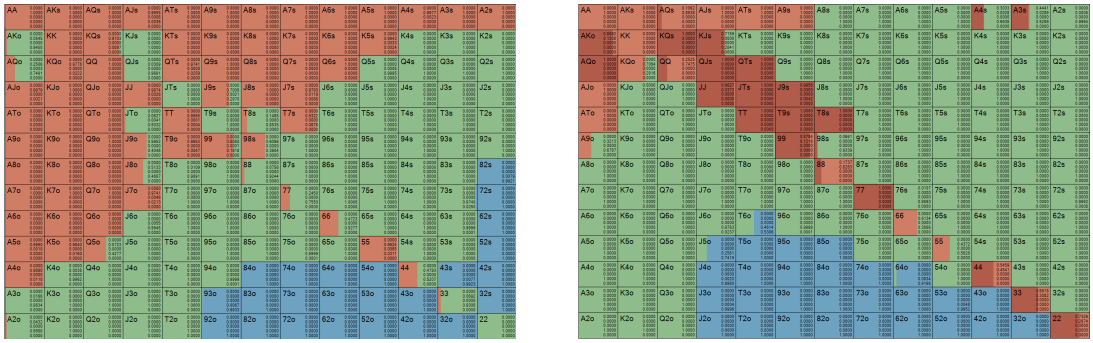


Figure 4.7 *Visualization of strategy produced by regular CFR in the software PioSOLVER for the same nodes as in Figure 4.5 with similar format of visualization. The colors dark brown, light brown, green and blue correspond to actions raising all-in, raising full pot, calling and folding.*

5 Conclusion and future work

Counterfactual Regret Minimization based algorithms have been used to achieve superhuman performance in many benchmark games. Deep learning has been applied to CFR in deep CFR to achieve more generality. This allows easier application of these algorithms to different problems where domain knowledge is not as available. The proposed Monte Carlo roll-outs were used to enhance the deep CFR method while not fundamentally changing it. The continuous action modifications were used to fundamentally change deep CFR, to reduce the required domain knowledge, when applying the algorithm to new problems.

The proposed continuous action modifications did not surpass the performance of regular deep CFR. Researching the reasons for this would be a good continuing point for future research. It is possible that some of the assumptions in the proposed changes are not fundamentally sound. Some of the changes could be such that they would break the convergence bounds proven to be there for regular deep CFR. The proposed changes were motivated by the idea of having a system that seemed logically analogous to regular deep CFR while changing the action space from discrete to continuous. If some of these changes were proven to not be analogous under closer scrutiny or to be otherwise mathematically unsound, some other way of achieving the same changes could be thought of and tried out. Another reason for the failure of the proposed modifications could be that the implementation had problems. The experiments were done with limited resources. For example, the number of traversals per iteration was quite small, and the number of iterations was also smaller than what was used in the research originally done for deep CFR. The implementation could have been more optimized or the hardware could have been faster to allow using a greater number of traversals and iterations. A concrete example of optimizing the algorithm would be implementing vector-form CFR. This would reduce the generality of the algorithm but would allow for reduced variance, possibly enabling the model to converge to a better strategy.

When inspecting the strategy learned by continuous deep CFR and comparing it to the other models, the action choices used for the continuous actions were very different. Studying the reasons for this difference could be an idea for future research as well. The model was not learning to use the whole action space like the models with abstractions did. This could be due to the mentioned limitations from hardware. Also, the parameters used for the neural network could cause this. Perhaps the

network would need to be trained to overfit the predicted distributions more closely to the data produced by the traversals. Other options would include the theoretical soundness of the algorithm. The traversal and sampling of the continuous actions could be unsound. A more likely problem in the algorithm could be the way the neural network is trained. The loss function could need modification to ensure the required convergence bound for CFR.

References

[1] Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

[2] Michael Bowling et al. "Heads-up Limit Hold'em Poker is Solved". In: *Commun. ACM* 60.11 (Oct. 2017), pp. 81–88. ISSN: 0001-0782. DOI: 10.1145/3131284. URL: https://doi.org/10.1145/3131284.

[3] Noam Brown and Tuomas Sandholm. "Superhuman AI for multiplayer poker". In: *Science* 365.6456 (2019), pp. 885–890. DOI: 10.1126/science.aay2400. eprint: https://www.science.org/doi/pdf/10.1126/science.aay2400. URL: https://www.science.org/doi/abs/10.1126/science.aay2400.

[4] Noam Brown et al. "Deep Counterfactual Regret Minimization". In: *CoRR* abs/1811.00164 (2018). arXiv: 1811.00164. URL: http://arxiv.org/abs/1811.00164.

[5] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[6] David Silver et al. "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *CoRR* abs/1712.01815 (2017). arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

[7] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[9] Sonali B Maind, Priyanka Wankar, et al. "Research Paper on Basic of Artificial Neural Network". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 2.1 (2014), pp. 96–100.

[10] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation Functions in Neural Networks". In: *International Journal of Engineering Applied Sciences and Technology* 4.12 (2020), pp. 310–316.

[11] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *arXiv preprint arXiv:1511.07289* (2015).

[12] Noboru Murata, Shuji Yoshizawa, and Shun-ichi Amari. “Network information criterion-determining the number of hidden units for an artificial neural network model”. In: *IEEE Transactions on Neural Networks* 5.6 (1994), pp. 865–872.

[13] Léon Bottou. “Stochastic Gradient Descent Tricks”. In: *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436.

[14] Christopher M Bishop. *Mixture Density Networks*. Tech. rep. Aston University, 1994.

[15] Robert Gibbons et al. *A Primer in Game Theory*. Harvester Wheatsheaf New York, 1992.

[16] Noam Brown. “Equilibrium Finding for Large Adversarial Imperfect-Information Games”. PhD thesis. Carnegie Mellon University, 2020.

[17] Branislav Bosansky et al. “An Exact Double-Oracle Algorithm for Zero-Sum Extensive-Form Games with Imperfect Information”. In: *Journal of Artificial Intelligence Research* 51 (2014), pp. 829–866.

[18] Roger B Myerson. “Refinements of the Nash equilibrium concept”. In: *International Journal of Game Theory* 7.2 (1978), pp. 73–80.

[19] Martin Zinkevich et al. “Regret Minimization in Games with Incomplete Information”. In: *Advances in Neural Information Processing Systems* 20 (2007), pp. 1729–1736.

[20] Sergiu Hart and Andreu Mas-Colell. “A Simple Adaptive Procedure Leading to Correlated Equilibrium”. In: *Econometrica* 68.5 (2000), pp. 1127–1150.

[21] Michael Johanson et al. “Efficient Nash equilibrium approximation through Monte Carlo counterfactual regret minimization.” In: *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2012, pp. 837–846.

[22] Richard G Gibson et al. “Efficient Monte Carlo Counterfactual Regret Minimization in Games with Many Player Actions.” In: *Advances in Neural Information Processing Systems (NIPS)*. 2012, pp. 1889–1897.

[23] Marc Lanctot et al. “Monte Carlo Sampling for Regret Minimization in Extensive Games”. In: *Advances in Neural Information Processing Systems*. Ed. by Y. Bengio et al. Vol. 22. Curran Associates, Inc., 2009. URL: https://proceedings.neurips.cc/paper/2009/file/00411460f7c92d2124a67ea0f4cb5f85-Paper.pdf.

[24] Oskari Tammelin. "Solving Large Imperfect Information Games Using CFR+". In: *CoRR* abs/1407.5042 (2014). arXiv: 1407.5042. URL: http://arxiv.org/abs/1407.5042.

[25] Kevin Waugh et al. "Abstraction Pathologies in Extensive Games." In: *International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*. 2009, pp. 781–788.

[26] Jeffrey S Vitter. "Random sampling with a reservoir". In: *ACM Transactions on Mathematical Software (TOMS)* 11.1 (1985), pp. 37–57.

[27] Noam Brown and Tuomas Sandholm. "Solving Imperfect-Information Games via Discounted Regret Minimization". In: *CoRR* abs/1809.04040 (2018). arXiv: 1809.04040. URL: http://arxiv.org/abs/1809.04040.

[28] Martin Schmid et al. "Variance Reduction in Monte Carlo Counterfactual Regret Minimization (VR-MCCFR) for Extensive Form Games using Baselines". In: *CoRR* abs/1809.03057 (2018). arXiv: 1809.03057. URL: http://arxiv.org/abs/1809.03057.

[29] Huale Li et al. "NNCFR: Minimize Counterfactual Regret with Neural Networks". In: *CoRR* abs/2105.12328 (2021). arXiv: 2105.12328. URL: https://arxiv.org/abs/2105.12328.

[30] Diederik P Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[31] *PioSOLVER*. https://www.piosolver.com. Accessed: 2021-11-29.