University of Applied

Ravensburg-Weingarten a
SciencesTechntk | Economics | Social Welfare

SciencesFaculty of Electrical Engineering

Bachelor Thesis

in Applied Computer Science - Focus on Robotics andTechnology Automation

to obtain the academic degree
Bachelor of Science

Topic: Deep Reinforcement Learning from Self-Play in Imperfect Information Games

Author: David Joos
david.joos@hs- Weingarten.de
MatNr. 24316.

Version from: 21. May 2018

1. Supervisor: Prof. Dr. med. Ertel
2. Supervisor: Prof. Dr. med. White

Summary

In this work, the implementation of the article by Heinrich and Silver (2016) and the resulting "Neural Fictitious Self-Play" algorithm is addressed. It will on the implementation of neural networks, Reinforcement, Supervised Learning and the "Neural Fititious Self-Play" algorithm. The necessary basics to work with these technologies will also be dealt with and in the respective Subchapters described. For the implementation was Python, Keras - which on TensorFlow and uses Numpy. It has been shown that with the implementation This work results similar to those achieved in the specification can.

INTRODUCTION

Artificial intelligence is becoming more and more popular and we are increasingly seeing it in everyday life Situations, be it the voice-controlled assistant or the autonomously acting one Vehicle. Also media-related events, such as the victory of Alpha Go over the world's best human player Lee Sedol Silver and Hassabis (2016)) are advancing the development artificial intelligence more and more in the focus of the public. The possible Areas of application are complex and attractive for many industries, be it the Financial industry or in the care. But as useful as this technology may seem, it is also discussed as controversial - headlines about face recognition in football stadiums, burdened the respectable citizens, as well as the AI supported recovery Personal data from corporations like Google or Facebook raises questions about the meaningful use of this technology. The advance of the AI is present everything in the intensive research of the last years. In the recent past especially the research in the field of artificial neural Attract attention to networks. Many of these innovative technologies will be first explored in a game environment, see AlphaGo by Deepmind. After this AI in various games like Chess, Go and the Atari Arcade games Meanwhile, there is still a domain that presents some big challenges contains: Games in which the AI does not have the full information of the game state. Poker, for example, is one of those games. This work should also dedicated to learning in games with incomplete information. there The most up-to-date technology is used and an executable program is developed learn a simple form of poker in which it plays against itself.

1.1 Task

This work deals with the article V011 Heinrich and Silver (2016). The necessary technologies for implementing the described "Neura1 Fictitious Self-Play" algorithm are shown and explained.

- Explanation of the necessary technologies needed to implement a workable Program are necessary.
- Development of the game environment Leduc Hold'em, which also described in the article becomes.
- Development of an agent and the neural networks for function approximation. The method used is explained in Heinrich and Silver (2016).

The aim of the work is to reproduce the authors' findings. Since the article does not go into the direct implementation, but just outlines a rough structure, a way is to be found, despite missing information an executable program develop on this basis.

1.2 Delineation

The bachelor thesis concentrates on the technical implementation of the NFSP algorithm and discusses all necessary technologies with an executable program to be able to develop. For this, common technologies and frameworks such as Keras, TensorFlow, Python and Numpy are used. These technologies are set as accepted and thus is the comparison to similar technologies of other offerers not part of this work. Furthermore, all parameters, e.g. the learning rate are taken directly from the article Heinrich and Silver (2016). Is from the specifications in the article is marked, this is marked and is used for easier implementation. It is not part of this work, the guideline to optimize in any way or question.

Notes on handling:
In this work, various words are used based on their English spelling . This is to facilitate the coping with related work, which is mainly written in English. To further simplify the language below some points defined

- Deep learning is when a neural network has a number n of hidden layers. Where n> 0, $\forall n \in N$
- An agent is a program that acts as an independent entity in it's environment. In this case, a program through the implementation of machine Learning process makes learned or randomized decisions.
- Reinforcement Learning is hereafter referred to as Reinforcement Learning. Furthermore, the abbreviation RL is used at various places.
- Supervised Learning is hereafter referred to as Supervised Learning. Furthermore, the abbreviation SL is used at various points.
- The programmatic implementation was done in Python.

2 Game Theory

Often, games are used to develop and study machine learning techniques, There is a simple reason for this: In a real-world scenario, the amount is often too great for possible states and causing the states to be too laborious. In a simplified simulated environment it can be both faster and easier to train, as exploring the state space faster and artificially limits the number of leaves in a game tree. Furthermore, games offer many features based on real-world scenarios. This work will also deal with artificial intelligence in the environment of a game. In this chapter the basics of game theory are conveyed to the reader.

2.1 Zero Sum Games

A game is then a zero-sum game when the sum of the winnings and losses of all players adds exactly to zero. In a two-player scenario, this is given if the gain of one is the loss of the other (see Holler and Illing, 2006).
Win (Player1) + Win (Player2) = 0

2.2 Games with incomplete information

It is possible to distinguish between games with incomplete and complete information. For example, chess and drafts are representatives of the games with complete information, which means in plain language that all players have at all times complete information on the score. Of course, this is not the case in card games, for example. In the course of this work, will be referred to a poker variant later on, and of course poker will be part of games with incomplete information. At the beginning of each game, private cards are issued which only the respective player knows. This means that players their own cards and the know cards on the table, but have no information about the private cards opponent's. The expected profit is therefore not deterministic. A player can therefore use the same card in different rounds orcard's V011 win, depending on the opponent. If there is no complete information, the player must discard probabilities and include them in his decisions.

2.3 Nash equilibrium

A Nash equilibrium, named after Nash et al. (1950), if it is no worthwhile for any player to deviate from his strategy. Suppose all players act rationally, which means that everyone will try to maximize their profits, then each player will try to find the best possible answer to the opponent strategy. When this is achieved, there is a Nash equilibrium and it is for no player worthwhile to deviate from his strategy.

2.4 Extensive and Normal Form

Extensive and normal forms are terms from game theory. The difference between the two lies in their representation. The normal form is commonly called a table or matrix, the extensive form mostly as a tree diagram. Look at one Two-player zero-sum game will be made clear what the differences make two representations. As an exemplary game was "scissors, stone, paper" selected.

2.4.1 Normal form

The game "scissors, stone, paper" is played by two players Anna, Otto. Everyone the player can choose from the actions {Scissors, Stone, Paper}. The chosen one Action is called strategy. If Anna wins, Otto loses. To ever the strategy of a player, the expected profits are registered, depending on the chosen opponent strategy. So Anna plays strategy "stone" and Otto strategy "Scissors" can be derived directly from the table:

GAnna (AnnaStein, OttoSchere) = 1

Heinrich and Silver (2016) define each of the actions as a pure strategy. each pure strategy Δip is part of the strategy amount Δib , Depending on the reward function *R* and the chosen opponent strategy becomes the reward feature for each player
i defined as follows:

$R\ i\ (\Delta i\ p\ ,\ \Delta{-}i\ p\ )$

Any pure strategy can therefore be considered as a game plan that can be used for any situation which a player may face, determines the reward. Will be a game played in multiple iteration, then each player will choose a blending strategy, which means that before each iteration, a game plan is chosen and this one for this one an iteration is played. The choice of pure strategy, depends on the mixed strategy Π, which is a probability distribution is about all pure strategies. The set of all mixed strategies is Δi For Player i and thus is $\Pi \in \times\ i \in N\ \Delta i$ with N = {1, ..., n} as the set of all players. In order to the expected reward can be further formalized:

$Ri : \times\ i \in N\ \Delta i \rightarrow R$

In the further course of this work, as also by Heinrich and Silver (2016) uses mixed strategies and pure strategies with big Greek letters described. During behavioral strategies, explained in the next subchapter, get small Greek letters.

2.4.2 Extensive Form

In contrast to the normal form can be with the extensive form sequential and not represent deterministic earnings expectations (see Holler and Illing, 2006). Applied on the "scissors, stone, paper" game, this results in a tree representation 2. In this presentation it can be seen that just as complex

sequential processes as well as non-deterministic profits. For example it is conceivable that Otto wins with "scissors" against Anna with "Stein", on the condition that Otto has previously lost with "scissors" against "stone". However, it is also easy to see that "scissors, stone, paper" only makes sense if it does not follow a sequential course, otherwise Otto would have an advantage He already knows what strategy Anna plays. That's why the Extensive form expand so that there is no sequential process - but that is mentioned for the sake of completeness 3.

Figure 3: Game theory: Extensive form. The fact that the OttO nodes are circled means that Otto does not know in which of these nodes he is located. He does not know what strategy Anna Wahlt has. With that , the game "scissors, stone, paper" can then also be represented as meaningful.

The requirements for the Extensive form representation, as described by Heinrich and Silver (2016) are the following components:

- N = {1, ..., n} describes the amount of players
- S is the set of states and thus the set of all states in the game tree.
- $\forall s \in S$ defines a set of actions A (s) that each player in the node s are available.
- For each player i there is an associated amount of information Ui and one Information function Ii: $S \rightarrow Ui$, This determines which of the states for the Players are indistinguishable if they have a state at the same informational state $u \in Ui$ maps. If from a game with incomplete Information is gone, then that means that player i to two states sk1 and sk2 the same u i k has available, though the conditions differ. By the example of poker, this would mean that the state of the game is equal to the private card of the opponent of player i

In this work it is assumed that every player has complete information has all the actions and states which uik guided to have. Each player knows the sequence: ui1 , ai1 , ui2 , ai2 , ...uik , If this is given, it is a game with perfect recall

- With the reward function R: S → R Finally, the final states become A vector is shown representing the reward for each player.

Each player now chooses a strategy. Similar to the mixing strategy in the normal form this strategy will map different states to different actions. The The difference is that the normal form after playing the action once assumes that the game is scheduled and a reward is distributed. In However, in Extensive Form, a game may involve multiple actions by the respective player before the game is scheduled. This is how we define the behavioral strategy, the each state in the tree represents an action:
πi (u) ∈ Δ (A (u)), ∀u ∈ Ui , Δib as the set of all behavioral strategies of player i. This determines the probability distribution about all possible actions for a given state. On Strategy profile π = {π1 , ..., πn} contains all strategies of all players and π-I refers thus all strategies in π except for πi, Following this reasoning becomes the expected reward of player i is defined as follows: Ri(π) under the condition that all players follow the strategy profile π4, Since it can be assumed that every player wants to win, every player becomes try to play a strategy that maximizes its expected profit - with it the best response can be defined. Player i will if possible in any state to choose an action that is under the condition that player -I the strategy π-i plays R maximizes:
bi(π-i) = arg maxπi∈ΔibRi(πi, π-i). OnNash equilibrium is reached when every player has the best response to the strategy of the game Opponent plays. Formalized this means: πi ∈ bi(π-i) for all i ∈ N 5

2.5 Leduc Hold 'em

Heinrich and Silver (2016) refer in their article to a poker variant named "Leduc Hold'em" (LHE). Because LHE is a special variant of the widespread Texas Hold'em, for the time being, on some terms and rules of the poker game to discuss the differences between LHE and Texas Hold'em.

Concepts:

- **Dealer**: The dealer is the player at the table who issues the cards. The dealer changes after each round. As a rule, the player to the left of the dealer becomes the new dealer.
- **Smallblind**: The player to the left of the dealer must provide the small blind. Since money is usually played, the small blind is a certain amount of money, this player has to bet.
- **Bigblind**: The player to the left of the Smallblind has the duty of providing the Bigblind. The amount of the bigblind is usually twice the size of the smallblind. Like the smallblind, the big blind is compulsory for each player.

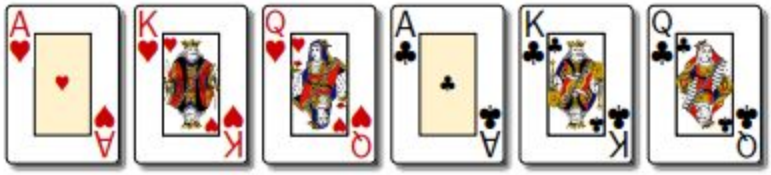
Texas Hold'em is played in rounds. At the beginning of the first round, the dealer mixes and hands out the private cards to each player. In conventional Texas Hold'em gives each player at the table two face-down cards. Are all cards dealt, begins the first round of betting. The small and big blind are obligatory, the However, players who are not affected by the blinds, can at this point already get out, if your private cards promise no profit. In the betting round. Each player has the opportunity to choose one of three actions:

- **Fold**: Folden means to quit the current game. The player gives his cards back to the dealer and is out for this game. If the player has already set an amount at this time, this is lost.
- **Call, Check**: This means a player does not want to raise but stay in the game. Has already been raised in the current round of betting, then the player must at least bring the bet already brought by the increasing player was (call). Has not been increased in the betting round, the player has nothing do (check).
- **Raise**: Raise stands for raise. If a player has good cards or even only bluffed, then he raises by an amount. This amount is at the discretion of the increasing, the difficulty with it is not to scare too many opponents, if the raiser has good cards or does not bet too low if bluffed becomes.

.

A betting round ends once all players have bet or folded the same amount. Each player can only raise once per turn. Increase Player 1 and then Player 2 increased again, then player 1 can

only call or fold. When the first betting round is completed, the table cards are revealed. there Three cards are placed face up on the table. Each player can now get out of the two private and the three open cards make a hand. Whether a hand wins or it does not depend on the rank of the hand. There are certain combinations the cards higher than others. For example, a couple is lower than one Triplet and so on. Then the second round of betting is played, the blinds are eliminated after the first round and thus, each player can decide freely whether to continue plays or leaves the game with fold. Subsequently, two more rounds are played, each time another card is placed open on the table. Too good Finally there are five open cards on the table. In case of two or more players the amount of the hand decides who wins the pot. The pot is the LHE is a limited edition of Texas Hold'em. At LHE are the insert heights Firmly prescribed, a player wants to raise, can only by the amount of the Bigblinds increase. Assuming the Bigblind equals "1" increase each player by only 1 each turn. Heinrich and Silver (2016) Texas Hold'em will usually come with one However, LHE in their article is played on a 6-card hand reduced. This also means that the number of laps must be limited. In this work, the number of rounds is limited to 2. Besides, get each player just a face down card at the beginning. In the first round of betting is after the Rules of Texas Hold'em played. If this round is completed, only one will Card laid open on the table. After the second betting round is completed and no player has folded, the winner is determined. Given a 6-card hand:



The player who has the same rank (for example, ace) with his face down card the face-up card is the winner. Because of each rank only two copies in the game are (three duplicates), it is impossible for both players to open the card to meet. If neither player meets the face-up card, the winner is the one who has the higher-value card in hand. Both players have the same rank on hand, it is a draw and each player gets his bet back. One last peculiarity is that the game heads-up is played. It means that there are only two players in the game and thus the dealer must provide the small blind and the other each the big blind.

2.6 Implementation

The game is implemented as a "black box". That means the players have none Knowledge about the rules of the game or reward function. Basically, the player can just interact with the game via two functions:

```
import leduc.env as env

# Initialisiere Spieler
spieler = spieler.Spieler()

# Initialisieren der LHE Umgebung
LHE = env.Env()

# Ruecksetzen der Umgebung nach Beendigung eines Spieles
# Hierbei muss der Dealer Index mituebergeben werden
LHE.reset(spieler.getIndex)

# Abgreifen des Spielzustandes als MDP codiert
s, a, r, s_1, t = LHE.get_state(spieler.getIndex)

# Im Spiel einen Zug ausfuehren
LHE.step(action, spieler.getIndex)
```

Listing 1: Basic interaction with the game environment LHE.

If a player asks the current state of the game via the function "get_state" then this must be given to the player's index as this is a game with incomplete Information is and every player may only get the information that is his due. The same applies to the "step" function. The environment lays in the background set a separate state for each player in the game and use the executed ones actions, always with regard to the index of the respective player. This leads to, that no player has to pay special attention to the score, since the game environment in the Background for each player done. The players interact with the game environment So only by querying the game status, because of which then one select action and hand it back to the game.

LHE is an Extensive Form game and so in state sk all the previous ones are executed actions and states s1, a1, s2, a2, ..., sk a decision criterion for the Player. Because of this, the state queries about the "get_state" function can be a betting history. The betting history describes what has been done actions of each player, each round. The state can be as:

$$2 \times 2 \times 3 \times 2$$

Tensor - by name (player, round, raises, actions) can be represented as it a two-player game, with a maximum of two rounds, 0 to 2 Raise per round and each is two possible actions. However, the action space consists of three actions { Fold, Call, Raise}, but as Fold leads to the demolition of the game, it will not included in the betting history. Furthermore, the cards still have to be:

$$2 \times 3$$

Tensor passed - in particular (round, maps) shown. The two tensors are concatenated together and passed as an array of length {30}.

3 Neural networks

The human brain consists of about 10 to 100 billion nerve cells in their interconnection and mutual activation define our intellect. After this The science has become more and more secure that all our skills are connected to the network originating from our nerve cells, then became the McCulloch and Pitts (1990) first mathematical model of a neuron as a central switching element of our brain presented. Based on this article could now first artificial neural networks be reproduced. Today, this technique is versatile and it also approximates nonlinear complex functions see Ertel (2009).

The structure may differ greatly depending on the nature of the task, therefore limited this work, only the necessary modules for the implementation of the To describe NFSP's.

3.1 Structure of an Artificial Neural Network

In a neural network, the neurons are connected in layers, here Different connections are conceivable. In this work should be a fully networked Network can be used. As already mentioned, the neuron is the building material of a every neural network. Whether a neuron becomes "active" (ie a signal is sent to it), will depend on whether the previous neurons have been activated or not. The top layer, which is just the input signal In the further course of the thesis, the term "input layer" is used. The neuron of the input layer thus act as the incoming connections for the following Layer, called Hidden Layer. Basically, in a fully networked Net, each neuron of a layer, with each neuron of the following or the previous one Layer connected. Thus, the activation potential of the neuron i be presented as a mathematical model (see Ertel, 2009.

$$\sum_{j=1}^{n} w_{ij}\ x_j$$

However, the activation potential does not say anything about whether the neuron is activated or not. Therefore, an activation function becomes the activation potential applied:

$$x_i = f(\sum_{j=1}^{n} w_{ij}\ x_j)$$

There are a variety of activation features and will be chosen according to your needs. A neural network thus consists of the input layer, n hidden layers and the last layer (referred to as Output Layer in this work). The output layer follows the same principles as all other layers. Furthermore it is of a discrete Time scale is assumed, that means the information per input signal will be passed through the network sequentially. Say, neuron i has the initial values at time t all preceding neurons of the input signal Inputt available. The data is processed sequentially and can not be used in parallel become. A neural network thus works in which the neurons within the network work activate each other and thus map the input values to the output values.

3.2 Activation Functions

As far as the NCCR is concerned, there will be two different activation functions later used, on the one hand the Rectifier and on the other hand the Softmax function. Both should be presented to the reader in this area.

3.2.1 Rectifier

A rectifier, first introduced by Hahnloser et al. (2000) corrected the function values in the only the positive values as part of its output be used. Formally expressed:
f (x) = x + = max (0, x) - that is for each Value that is not part of the positive set of x, a 0 is returned, otherwise x see Figure 4.

3.2.2 Softmax

The Softmax function calculates the probability distribution over all neurons
(see Christopher, 2016):

$$F(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}}$$

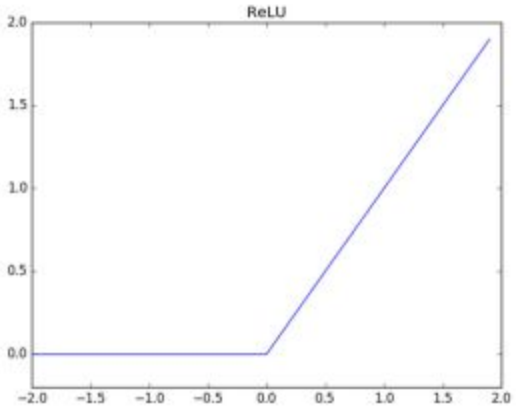


Abbildung 4: Aktivierungsfunktion: ReLu

It applies Pki = 0 xi = 1 and thus is a probability distribution over all neurons from x1, ..., xk. With classification problem this is useful, since the Softmax Function for each output neuron indicates a probability with which this neuron corresponds to the expected output.

3.3 Backpropagation

If you imagine the network in layers, see Figure 5, and the underlying underlying activation of the individual neuron and the edges as weights understands, can be easily understood how the input signal to the output (output Layer) is mapped. A net that has no previous knowledge, so it was randomly initialized

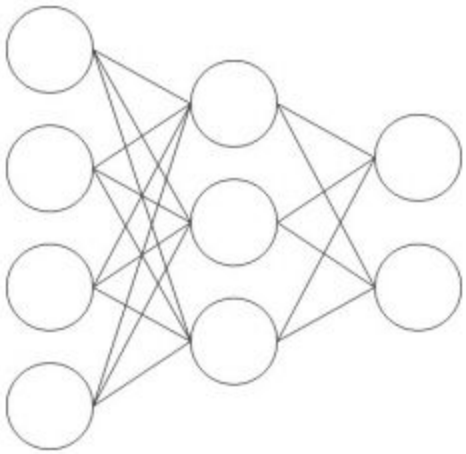


Figure 5: Fully connected neural network. The circles represent the neurons The edges are the weights between the neurons.

(random weights), will map each input signal to a random output. If you want to change this behavior and let the net learn, then you need the output of the network with the expected Ouput compare and accordingly correct. The comparison between the output of the network and the control data can take various forms and is commonly called an error or cost function C, see more in Basic Machine Learning on p. 18. The value This feature is an indication of how well the net's evaluation function has approximated. A network that has already learned about many training examples should have a low value of the cost function. Because it always applies, the cost and thus to minimize the cost function. Because only the weights of the network can be influenced are, the weights should be changed so that the output layer activates the neurons it is supposed to activate. To achieve this is the backpropagation algorithm used. The weights are in accordance with the negative gradient, the cost function summed over the output neurons.

$$E_p(w) = \frac{1}{2} \sum_{k \in Ausgabe} C(t_k^p, x_k^p)$$

changed for the training pattern p:

$$\Delta_p w_{ji} = -\eta \frac{\partial E_p}{\partial w_{ji}}$$

Now replace Ep with (2) and xk with (1), then recursively the Issues xi the next lower layer before and in turn the next and so on continue. By repeatedly applying the chain rule (see Rumelhart et al., 1985) results in the backpropagation learning rule (see Ertel, 2009, p. 292). As from the documentary Keras implements the backpropagation algorithm without any intervention the user Chollet et al. (2015) so do not go further to the exact details will be received.

4 Basics Machine Learning

When we learn, both on purpose and not, we define ourselves and ours New environment. A change of our thinking and / or action has occurred and leads to a different interaction with what we perceive and what we do. This ability is the basic requirement for the animal and the human being adapt to their environment in due course and thus survive. Especially the ability to associate, that is to link different sensory impressions and the resulting amplification of neural links, too called synapses, allows people to learn. To the perception of environment, man has his sensory impressions and emotions at his disposal.

After the world and your own position in it has been evaluated, this leads to a Plot. Once the action has been carried out, it is again evaluated how your own Position has changed depending on the action. Is the new position more desirable? than the previous one, that means the act in such an environment and to the prevailing condition can be considered as meaningful and thus wants to be repeated. What is natural for humans is heavy on machinery transferred to. The "perceive, act, evaluate" pattern, as in the previous paragraph does not represent the full spectrum of human learning heard but always to learn about it. As a result, the idea of reinforcing emerged Learning. The term "learning through reinforcement" covers a number different methods, but all are based on "perceiving, acting, evaluating" Template. The "rating" part plays the biggest role, it is evaluated based on the reward function and always maximize the reward. What the agent and how it can act depends on the implementation. Is the world little

complex and the amount of possible actions small, it may be possible to derive a unique action for each state, but most often it is World more complex and the set of possible states and action very big and the Reward function must be approximated. For example, this will be neural Networks used - as described in the previous chapter. On the other hand but it is also possible to learn from already known. Is that what you learn should already be known, the own learning progress on the basis of correctly classified data be checked - this also applies to machine learning. This is called from supervised learning. Both approaches are part of this work and are becoming Implementation of the NFSP.

In summary, basic learning methods can be divided into two classes:

- Supervised Learning
- Reinforcement Learning

As the NFSP uses both supervised and unmonitored learning, the reader should be introduced to the principles of both methods.

4.1 Supervised Learnin

Supervised learning serves to derive regularities from existing knowledge and apply them to new data. The existing knowledge can be found in this trap be either expert knowledge or tagged data. This procedure is called monitors because artificial intelligence is learning data that is already known and the AI can control its own statements and correct them if necessary (see Mohri et al., 2012)).

Later in this work, a monitored classifier will be implemented become. Classification means having an input vector on a set of classes is pictured. For the already introduced game LHE this means the game state is mapped to one of the three possible actions {F old, call, raise). The evaluation function F (x), which maps the state to the actions, should be learned. Success is measured by a cost function. Because this is a supervised learning process is, the rating function needs already classified data - it will be thus defines the following points:

- S is the set of all states. For every state s ∈ S.
- A (st) is the evaluation function, which for the state st is a probability distribution about the possible actions. That means A: st → ypred with ypred as the approximate probability distribution for st
- ytrue is the correct probability distribution for st
- C (w) as a cost function with w as the weights of the neural network.

Heinrich and Silver (2016) train the classifier with the following cost function:

$$\mathcal{L}(\Theta^{\Pi}) = \mathbb{E}_{(s,a)\sim M}[-\log \Pi(s,a|\Theta^{\Pi})]$$

with ΘΠ as the weights of the neural network Π, M as the memory for (s, a) tuples played in LHE, where s is the game state and a is a probability distribution is about all possible actions. For a, the "1 out of n" coding is used which means that exactly one element of the vector corresponds to a 1 while all other elements 0 correspond. Transfer to the defined points:

- ypred corresponds to Π (s, a) which corresponds to A (st)
- ytrue are collected in E (s, a) ~M
- C (w) corresponds to L (Θ)

The cost function corresponds to the cross-entropy function and measures how "close" the prediction ypred corresponds to the desired distribution ytrue (see De Boer et al., 2005):

$$H(y_{true}, y_{pred}) = -\sum_{s} y_{true}(s) \log y_{pred}(s)$$

With the Backpropagation Algorithm 3.3, the network can now be based on the cost function be trained.

4.2 Learning through amplification

Learning through reinforcement is also a branch of artificial intelligence and serves to let the AI learn without relying on expert knowledge or tagged data to reach back. As these data are missing, the AI must be in the process of learning received a feedback on the success so far. Because of this success, i Called Reward in this case, the previously selected actions can be evaluated become. Initially, without any knowledge of the reward that an action might have the AI will perform random actions. After then one reward has been confirmed, the AI can evaluate the previous actions. Ertel (2009) defines the following points on p.316:

- The world is the environment. This can be different, for example the Play chess or our "real" world (earth, solar system, universe). ever After what happens in which world, the state space changes under circumstances dramatically. The game of chess, for example, consists of all game pieces, where they are and so on, defining the whole State space while in a real-world scenario, there are more states can enter.
- S is the set of all possible states.
- A the set of all possible actions.
- δ is the transition function.
- A strategy π: S → A maps states to actions.

An agent interacts with his environment by giving the state at time t st ∈ S of the world. In a world with very large state space, the state must may be abstracted because the set of factors that define the state is too large or the condition is not fully captured for the agent. The Agent then executes an action at ∈ A under the condition st. This action has Impact on the world and the agent and leads to the state st + 1 - this one Step is determined by the transition function st + 1 = δ (st , at). The agent gets through this transition to the next state a feedback rt and stands directly related to the chosen action to a particular state (st, at). The feedback is positive if the selected action is under the condition of a particular state was expedient or negative if this

is not the case. If rt is positive then rt> 0 the chosen behavior is amplified, rt <0 becomes that Behavior is negatively reinforced. If the feedback rt = 0, then the agent has no Get direct feedback from the world.

Suppose an agent should learn chess. By the end of the game will be a whole Order of moves selected. The agent will randomly become any figures on the field and depending on how the reward is Lost} he will rate his moves - but it is not clear which train led to the defeat or the victory, because until the last move the feedback r = 0 was. This problem is called credit assignment problem known (see Ertel, 2009, p. This problem can be counteracted by using the discounted reward:

$$\mathcal{V}^{\pi}(s_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

With 0 ≤ γ <1 as a constant, the further in the future feedback the more it weakens, the farther they are in the future. Vπ is there the Reward on the condition that that agent pursues strategy π, this reward should be maximized. However, this becomes a problem if the agent is not a model of World and its actions that it will take in the future can rate from the outset. In the work of Heinrich and Silver (2016), the Game LHE investigated and in this scenario, the agent has no such model of World available. In this case, a different approach can be used and will be described in the following subchapter.

4.2.1 Q-Learning

Q-learning is a popular approach to RL problems. Because the agent does not know in which state its action at in state st leads, becomes an evaluation function introduced Q (s, a) - the selection of action at in state st is formally called (cf. Ertel, 2009, p. 324):

π ∗ (s) = arg maxa Q(s, a)

Are defined. The function Q (s, a) returns the value of the expected reward for at under st back. Since the Q-values without prior knowledge certainly do not correspond to the true Q-values, the Q values must be adjusted. Q-Learning is an "online" learning algorithm, that is, with each new experience, the Q values become as follows rectified:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$$

Heinrich and Silver (2016) use the article by Riedmiller (2005) in their article "Fitted Q Iteration" (called FQI in the following) algorithm. The The underlying idea here is to forgo learning online as it is slow converges. Riedmiller (2005) has shown that it is more efficient, first a lot

gaining experience without changing the Q values, these experiences become stored in transitional stuples (st, at , st + 1). This form of transitional tuple can be referred to as Markov Decision Process and will be referred to as MDP. An MDP includes (S, A, T, r, p0) where S is the set of states, A set of actions, T the action model, which with T: S × A × S → [0, 1] so that T (s, a, s0) = p (s0 | s, a) the probability is of state s and execution of action a in the state s0 to get. r: S × A × S → R is includedthe reward function and assigns the transition from s to s0 the reward too. p0 is the starting distribution and indicates the state of each probabilityto start in this state (see Sutton and Barto, 1998). With the MDP's from theExperience gained by the FQI without updating the values can now SL procedures are used to learn from these lessons learned,This is called Experience Replay. This approach has been described by Mnih et al. (2015) and is called "Deep Q Network". In this work will bethe "Deep Q Network" algorithm (called DQN in the following) used. The DQN has been developed specifically for deep learning and also in this work the algorithm is applied to a neural network. The authors have one implemented series of improvements that make learning more stable and faster:

- Like the FQI used on the DQN Experience Replay.
- A target network will be used (later in the chapter) illustrated).
- Clipping Rewards, which means the maximum and minimum reward be limited to a fixed value.

The target network is its own neural network, initialized with the same Weights as the DQN network itself ΘDQN → artarget. Will not be a target network used, applies:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a[r_{t+1} + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))]$$

The authors Mnih et al. (2015), however, have shown that the DQN network more stable learns when predicting the expected reward rates of the state st + 1 the target network is made. And the learning rule is changed as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + a[r_{t+1} + \gamma \max_{a_{t+1}} Q_{target}(s_{t+1}, a_{t+1}) - Q(s_t, a_t))]$$

The target network is delayed with the scheme ΘDQN → Θtarget updated. How much later the target network needs to be updated is not well defined and must be chosen differently depending on the claim. Heinrich and Silver (2016) update the target network every 300 iterations, with oneIteration of an update of the DQN network.

5 Neural Fictitious Self-Play

5.1 Basics

Fictitious play, introduced by Brown (1951), is a popular model of game theory to learn in games. Fictitious play means that the Players of a game play iteratively against each other. At each iteration the Players try to find the best response (introduced in 2.4.2) to the strategy of the opponent. Brown has shown that the blending strategies of players in certain games (e.g. Two-player zero-sum game) converge to a Nash equilibrium. Despite his Popularity found Ficititous Play only sparse use in large applications, which is due to its dependence on the normal form, because real-world scenarios can usually be described only by the Extensive form. Furthermore Fictitious play is based on the idea of going through all s ∈ S to achieve convergence. The Authors Leslie and Collins (2006) followed with generalized weakened ficititous play an algorithm that only approximates the best response and thus we do not have to fully traverse the game tree. This is especially essential in very large state spaces, which can not be traversed with dynamic recursive programming. That makes it especially suitable for machine learning. Heinrich et al. (2015) introduces Full-Width Extensive-Form Fictitious Play (XFP) which allows us to update behavioral strategies of the agents. An important point here is the realization probability of stems (1996) assuming that it is a game with perfect recall (see 2.4.2), the probability of realization of each state is defined as follows:

The probability of realization acts as a link between Extensive and normal form and, if one follows the argumentation of (ibid. 1996) and Kuhn (2016), to the normal form mixing strategy: σ = λ1π1 + λ2π2 the Extensive Form Equivalent as:

$$\sigma(s,a) \propto \lambda_1 x_{\pi_1}(s)\pi_1(s,a) + \lambda_2 x_{\pi_2}(s)\pi_2(s,a) \quad \forall s,a \qquad (4)$$

Equation 4 describes a way of experiencing the agents to collect without recursively going over all game states. (left to right) Heinrich and Silver, 2016, p. 3). Heinrich et al. (2015) introduces Fictitious Self-Play (FSP) Algorithm that works with Experience Replay and Machine Learning. 4 describes a form in which FSP can gain experience. More precisely, the FSP agents generate data in the game against themselves, saving the experience as MDP (st, at, rt + 1, st + 1) in a memory MRL provided for reinforcement learning is. The data in MRL includes the behavior of the opponent, since st + 1 only can be achieved if the opponent has also acted. The experiences, of one's own behavior (st, at) is stored in MSL, suitable for a supervised Learning approach. An approximate solution of the MDPs in MRL thus provides an approximated best response while the data in MSL represents the agent's average strategy represents (see Heinrich et al., 2015).

5.2 How the NFSP works

NFSP combines FSP with neural networks. The following is the operation explained in detail, but first some points are defined first (cf.Heinrich and Silver, 2016, p.3):

- The best response network (BR network) approximates from the data in MRL's best response, through off-policiy Reinforcement Learning (DQN).
- The average response network (AR network) approximates the average Strategy from MSL through supervised classification.
- βi (π-i) is the best response strategy of the player i on the condition that Player -i the strategy π-i tracks and is described by the BR network
- πi is the player's average strategy and is played by the AR network described.

In the algorithm, see Figure 6, each player is separated by a NFSP Controlled by agents. Every NFSP agent interacts with his teammates and learns doing the game against each other. The experiences that the agents will gather stored in two memories (MRL, MSL). The agent trains the BR network (DQN 4.2.1), which learns Q-values based on the MRL, which in turn is the best response represent. Another neuronal network, the AR network, is trained, which is the average of the previously achieved best responses from MSL approximated by classification: π = Π.

As the agents play, they choose their actions from a mix of the two Strategies βi and πi from (see Heinrich and Silver, 2016):

Initialize game Γ and execute an agent via RUNAGENT for each player in the game
function RUNAGENT(Γ)
  Initialize replay memories $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir)
  Initialize average-policy network $\Pi(s, a \mid \theta^{\Pi})$ with random parameters θ^{Π}
  Initialize action-value network $Q(s, a \mid \theta^{Q})$ with random parameters θ^{Q}
  Initialize target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
  Initialize anticipatory parameter η
  for each episode **do**
    Set policy $\sigma \leftarrow \begin{cases} \epsilon\text{-greedy}\,(Q), & \text{with probability } \eta \\ \Pi, & \text{with probability } 1-\eta \end{cases}$
    Observe initial information state s_1 and reward r_1
    for $t = 1, T$ **do**
      Sample action a_t from policy σ
      Execute action a_t in game and observe reward r_{t+1} and next information state s_{t+1}
      Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in reinforcement learning memory $\mathcal{M}_{RL}$
      if agent follows best response policy $\sigma = \epsilon$-greedy (Q) **then**
        Store behaviour tuple (s_t, a_t) in supervised learning memory $\mathcal{M}_{SL}$
      end if
      Update θ^{Π} with stochastic gradient descent on loss
        $\mathcal{L}(\theta^{\Pi}) = \mathbb{E}_{(s,a)\sim\mathcal{M}_{SL}}\left[-\log \Pi(s, a \mid \theta^{\Pi})\right]$
      Update θ^{Q} with stochastic gradient descent on loss
        $$\mathcal{L}\left(\theta^{Q}\right) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{M}_{RL}}\left[\left(r + \max_{a'} Q(s', a' \mid \theta^{Q'}) - Q(s, a \mid \theta^{Q})\right)^2\right]$$
      Periodically update target network parameters $\theta^{Q'} \leftarrow \theta^{Q}$
    end for
  end for
end function

Abbildung 6: NFSP: Algorithmus (vgl. Heinrich and Silver, 2016, S. 4)

So before any game, an agent chooses one of the two strategies and stays with this until the game is scheduled. Basically every player could do the Strategy π, then each player calculates the best response βi(π-i) against the adversary strategy π-i - but no player would be his own best response learn behavior. Therefore, the strategy is changed so that the average best response can be learned and the opponents nevertheless in the proportion σ ≡ (1 - η) π + ηβ play against the average strategy of this player. η is here the Anticipatory Paramater (see Heinrich and Silver, 2016) or (see Shamma and Arslan, 2005).

6 Methodology

The article by Heinrich and Silver (2016) is a recent publication and it there are no references that you could refer to. The procedure is deriving directly from the article, making it difficult to validate how exactly the specifications have been implemented. The project was divided into three modules:

- LHE game environment: Since the game should work detached from the agents, it is developed decoupled by the agents. The interface to the Interaction with the game the well-known OpenAI Gym by Brockman et al. (2016).
- The NFSP Agent: The agent contains all necessary technologies like the neural ones Nets and a "game" feature in which he based on a st on an at maps and returns them.
- The main function: Here all players and the game environment are initialized become. Here basic parameters are defined such as e.g. who the dealer is etc. In addition, in this function, each round of the game is started and the Number of iterations set.

All program code is written in Python and it becomes common technologies used as TensorFlow, Keras, Python and numpy. It is recommended common to use best practice such as Git and Lean Development. Thereby redundant Work is avoided and the code quality is kept constant.

If all modules have been implemented analogously, the entire process should be recapitulated, the data obtained are analyzed and the results are validated. The data should be compared against the specifications of Heinrich and Silver (2016).

7 Practical implementation

Note: The development of the game environment "Leduc Hold'em" is already covered in 2. Since the implementation of LHE was "straight forward", it will not be discussed in detail like the special features of the program. If the reader would like an insight on how LHE was implemented in detail, reference is made to the Annex.

7.1 Summary

In this chapter it is shown how the practical implementation of the article Heinrich and Silver (2016) is initialized, trained and neural networks are used. Keras is used to implement this because its syntax is easy and it offers an easy introduction to the topic of machine learning with neural networks. In some sections, the object "np" is referenced so that is numpy and is imported at the beginning of each file with "import numpy as np". Some parameters are not referenced, as they are by the authors articles are defined and can be found in the project in the

"Config.ini" file. The interested readers will find the complete code in the appendix. In addition, on the Interaction between the agents received, like this with the game environment LHE interact and how the course of a game is done. The implementation keeps close to the pseudocode of the authors, but has to be modified, since in the Pseudocode is not received on the generation of game data.

7.2 NFSP Agent

7.2.1 Initialisierung

The NFSP agent should have 3 neural networks:

- The best-response network
- A target network to learn that is initialized to the same weights as the best-response network itself.
- The average-response network

Heinrich and Silver (2016) tested the NFSP agents with only one hidden layer, which contains 64 hidden neurons. The input layer should be a vector with 30 elements The condition of the game contains 30 elements, the describe both the betting history and the cards. The state should be on three actions which in turn represents three output neurons (see 2). The activation function for the output layer is the identity of f (x) = x (in the code "linear"), since this is a zero-sum game and the Reward can be both negative and positive. On Clipping Reward 4.2.1 will be omitted, since no information can be found in article12. The activation function "relu" has already been introduced in 3 and is used for the Hidden Layer used

```
def _build_best_response_model(self):

        input_ = Input(shape=self.s_dim, name='input')

        hidden = Dense(self.n_hidden, activation='relu')(input_)

        out = Dense(3, activation='linear')(hidden)

        model = Model(inputs=input_, outputs=out, name="br-model")

        model.compile(loss='mse', \
                optimizer=self.sgd_br, \
                metrics=['accuracy', 'mse'])

        return model
```

Listing 2: Initialization function of the Best-Response network with Keras Chollet et al. (2015). Dense stands for "fully connected". "Mse" stands for "Mean Squared Error"

and corresponds to the specified cost function for the Best-Response network (cf. Heinrich and Silver, 2016).

The average-response network (see 3) is modeled on the best-response network, however, sets a different activation function over the output layer. The output should lay a probability distribution over the possible actions, whereupon "Softmax" 3 was selected.

```
def _build_avg_response_model(self):

        input_ = Input(shape=self.s_dim, name='input')

        hidden = Dense(self.n_hidden, activation='relu')(input_)

        out = Dense(3, activation='linear')(hidden)

        model = Model(inputs=input_, outputs=out, name="ar-model")

        model.compile(loss='categorical_crossentropy', \
                optimizer=self.sgd_ar, \
                metrics=['accuracy', 'ce'])

        return model
```

Listing 3: Initialization function of the average-response network with Keras Chollet et al. (2015). "Categorical crossentropy" is described in Figure 4 and corresponds to Specify cost function for the average-response network.

The nets are built in the "_init_" function and the weights are set. Keras allows a very legible and simple implementation of this process.

```
# build average strategy model

self.avg_strategy_model = self._build_avg_response_model()

# build dqn aka best response model

self.best_response_model = self._build_best_response_model()
self.target_br_model = self._build_best_response_model()
self.target_br_model.set_weights(self.best_response_model.get_weights())
```

Listing 4: Initializing the networks and setting the weights of the target network.

Once the networks have been initialized, they can be used with the "predict" function, depict states on actions.

7.2.2 Learning the Agent

The best-response network learns the Q values via the data in the MRL. These dates are stored as MDP and in the form (State, Action, Reward, Statenext, T erminated) to be found. Terminated states whether this state is a final state and state after Statenext nothing comes, because the game is over. It will be a "batch learning" Method (see Mnih et al., 2015), as this leads to more stable learning and learning faster convergence. The idea behind it is that from the data collected a randomized amount (batch) is taken out. The negative gradient becomes then calculated over all elements in the batch and the mesh due to this gradient updated (see 5).

```
def update_best_response_network(self):
        """
        Trains the dqn aka best response network through
        replay experiences.
        """

        if self._rl_memory.size() > self.minibatch_size:

                s_batch, a_batch, r_batch, s2_batch, t_batch = \
                        self._rl_memory.sample_batch(self.minibatch_size)

                target = self.target_br_model.predict(s_batch)

                for k in range(self.minibatch_size):
                        if t_batch[k]:
                                target[k][0][np.argmax(a_batch[k])] = \
                                        r_batch[k]
                        else:
                                Q_next = \
                                  np.max(self.target_br_model.predict \
                                  (np.reshape(s2_batch[k], (1, 1, 30))))
                                target[k][0][np.argmax(a_batch[k])] = \
                                  r_batch[k] + self.gamma * Q_next

                self.best_response_model.fit(s_batch, target,
                        batch_size=self.minibatch_size,
                        epochs=2,
                        verbose=0,
                        callbacks=[self.tensorboard_br])

                self.iteration += 1

                self.update_br_target_network()

                self.epsilon = self.epsilon ** 1/self.iteration
```

Listing 5: Updating the Q-Values of the DQN

Thus, the best-response network follows the DQN learning rule as described in 4.2.1. Since for the exploration of the states -greedy is used, with the probability selects a random action or with 1 - selects the action which is the highest reward promises, at the end of each update, must be smaller because of the Agent should increasingly choose actions that are promising. The "fit" feature implemented the backpropagation algorithm based on the given cost function ("Loss" see 2). The "update_target_network" function adjusts the weights every 300 steps of the target network, the Best Response Network The average-response network should from the data of the MSL the previous BestResponse Derive behavior (see 6).

```
def update_avg_response_network(self):
        """
        Trains average response network with mapping action to state
        """
        if self._sl_memory.size() > self.minibatch_size:
                s_batch, a_batch = \
                  self._sl_memory.sample_batch(self.minibatch_size)

                self.avg_strategy_model.fit(s_batch, a_batch, \
                batch_size=128, \
                epochs=2, \
                verbose=0, \
                callbacks=[self.tensorboard_sl])
```

Listing 6: Average-Response Network: Classify the previous BestResponse actions Network.

7.2.3 Playing the Agent

Each agent can interact with the game via the "play" feature. He chooses for the next game (albeit already in the main function, since he is for a total Play with his strategy remains) with the probability η the β strategy or with 1 - η the π strategy, each represented by the best response or the average response Network. In the game function, which strategy is selected was traded and after this. Where policy == a, then the AverageResponse Strategy to be played (see 7).

```
def play(self, policy, index, s2=None):
        if s2 is None:
                s, a, r, s2, t = self.env.get_state(index)
                self.remember_for_rl(s, a, r, s2, t)
                if t:
                        return t
        else:  # because it's the initial state
                t = False
                if not t:
                        if policy == 'a':
                                a = self.avg_strategy_model \
                                  .predict(np.reshape(s2, (1, 1, 30)))
                                self.env.step(a, index)
                        else:
                                a = self.act_best_response \
                                  (np.reshape(s2, (1, 1, 30)))
                                self.env.step(a, index)
                                self.remember_best_response(s2, a)
                self.update_strategy()
                return t
```

Since e-greedy finds use, the "predict" function of the best-response network becomes outsourced, which checks whether it is a randomized action or the most promising chooses. In addition, only the best-response behavior is stored in MSL, if the best-response strategy is also played (see Heinrich and Silver, 2016) and see Figure 6.

7.3 Main

In the Main function, the agents and the game environment are initialized. The Game Environment manages the states, rewards and rules of the game. It however, it has been found that it is difficult to play the game within the environment to regulate. This has been outsourced to the Main function (see 8).

```
def train(env, player1, player2):
  eta = float(Config.get('Agent', 'Eta'))
  players = [player1, player2]
  dealer = random.randint(0, 1)

  for i in range(int(Config.get('Common', 'Episodes'))):
    if dealer == 0:
    dealer = 1
    else:
    dealer = 0
    # Set dealer, reset env and pass dealer to it
    env.reset(dealer)

    lhand = 1 if dealer == 0 else 0
    policy = np.array(['', ''])
    # Set policies sigma
    if random.random() > eta:
      policy[dealer] = 'a'
    else:
      policy[dealer] = 'b'
    if random.random() > eta:
      policy[lhand] = 'a'
    else:
      policy[lhand] = 'b'

    # Observe initial state for dealer
    d_s = env.get_state(dealer)[3]

    terminated = False
    first_round = True
    d_t = False
    l_t = False

    while not terminated:
       actual_round = env.round_index
     if first_round and not d_t:
       d_t = players[dealer].play(policy[dealer], dealer, d_s)
       first_round = False
     elif not first_round and not d_t:
       d_t = players[dealer].play(policy[dealer], dealer)
     if not l_t:
       l_t = players[lhand].play(policy[lhand], lhand)
     if actual_round == env.round_index and not d_t:
       d_t = players[dealer].play(policy[dealer], dealer)
     if d_t and l_t:
       terminated = True
```

The "while" loop in line 34 launches a whole game, this is just scheduled, if one of the players folded in the meantime or played both betting rounds to an end and the winner is confirmed.

8 Conclusion

On the basis of the requirement an executable program was developed, that the necessary Implements technologies, provides a gaming environment and "Neural Fictitious Self-play "uses. The pretended game "Leduc Hold'em" was the specification implemented in accordance with a "two-player" scenario has been. It is always possible for any player to change his current state in the game request and store this in the form of a "Markov Decission Process". The stores for the respective experiences the requirement were implemented accordingly and could be used for the function approximation of the agents. The agents themselves implement the required three neural networks, with the best response with a target network and the average-response network the strategies of the players represent π and β. The goal was to create a Nash balance of player strategies. The NashGG should be measured by the exploitability of a strategy profile. The limited Computing power that was available in this work, it is owed that the results of the article could not be fully reproduced. The
original idea was to fully go through the game tree of the game environment putting one player's strategy on π and β's on the other.

The average expected gain of the β strategy represents the "exploitability" If this approach turned out to be too computationally intensive, the "exploitability" approximated and per iteration to a small amount of randomized states imaged, which has led to a distorted and distorted image has delivered (see 7). However, it can be seen the same curve as in the Default (see 8). The curve varies greatly depending on the activation function used in the best-response network. The choice of the activation function in the Best-response network has led to idiosyncratic behavior, has been responsible for the output For example, using "relu" layers, the agent has always folded, as it is quite fast for every action had the value 0 predicted. Because the action with the highest expected Reward should be chosen, however every action promised zero profit, it came to the behavior just described. The best results scored the agent with one linear activation function, which also assigns negative values to the actions. Of Further, it turned out that Q values are the target for the average-response network led to a poor

learning success. There were different possibilities in Considered to represent the Q values as a probability distribution. However For example, the best results were obtained as the Q values with a "1 out of k" coding used as a target. Reinforcement Learning is known to be computationally intensive and because of this work multiple agents, each with multiple networks included, requires more power as that of a standard Macbook Pro (mid-2012) with an i7 and no dedicated Graphic card. However, it could be shown that the program is a similar one Behavior shows as in the specification and thus the goal of the work could be sufficient be fulfilled.

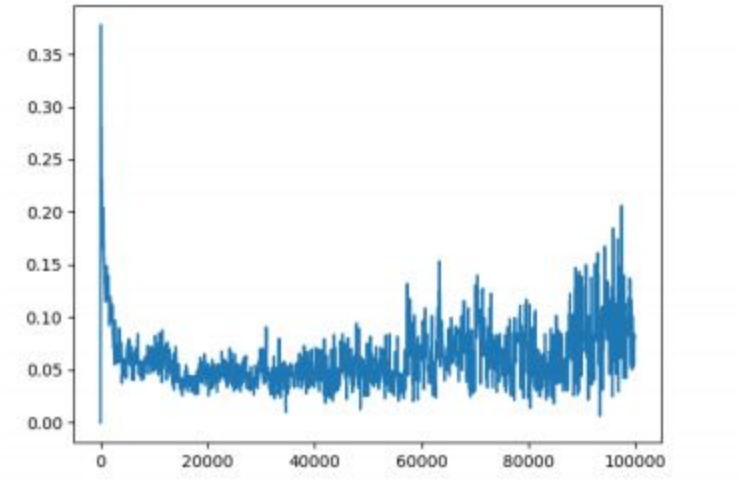
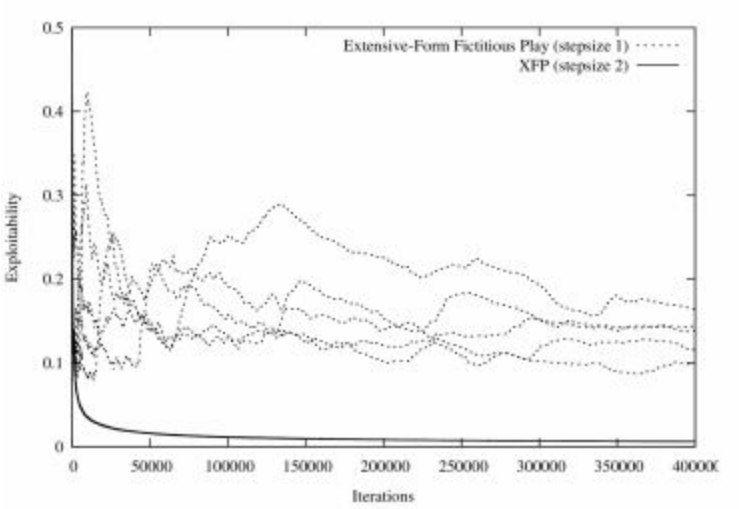


Abbildung 7: Ergebnis: Kurve bis 100k Iterationen



9 Outlook

Apart from the limiting factor of computing power, it can be said that too Games with incomplete information are no longer a problem for machine learning. With enough computing power you can use the program developed in this work Easily on games with larger state space such as Texas Hold'em porting. There are providers such as Amazon or Google's computing power in the cloud so that even more complex worlds can be explored. chipmaker like NVIDIA working under high pressure on chipsets optimized for machine learning become. If the hardware factor is set aside then it will become simple Game environments soon no longer pose hurdles to machine learning. Naturally Real-world scenarios are much larger, but with technology advancing It is conceivable that machines in the future will also be in the real world of complicated problems can accept the agent's limited information to be part of.