Computer Poker with Neural Networks

Tjebbe Laurens Vlieg
5647185

Bachelor thesis
Credits: 18 EC

Bachelor Opleiding Kunstmatige Intelligentie

University of Amsterdam
Faculty of Science
Science Park 904
1098 XH Amsterdam

Supervisors
Matthijs Snel, MSc.
dr. Maarten W. van Someren

Informatics Institute
Faculty of Science
University of Amsterdam
Science Park 904
1098 XH Amsterdam

June 26, 2012

Abstract

In this paper the suitability of artificial neural networks as main mechanism for poker agents will be examined. This is done by generating various neural networks out of a database of poker matches from the Annual Computer Poker Competition. It is shown that, although neural networks might be suitable for creating poker agents, a reasonable level of poker will not be achieved with only the most elemental features. It is suggested that to achieve a better poker agent, some form of opponent modelling is needed. Furthermore, this paper illustrates the various steps that must be taken in developing a poker agent.

Keywords: Computer poker, Artificial neural networks, Machine learning.

Contents

1 Introduction

Games have always been an important field in artificial intelligence research. Among the first computer programs were game playing agents: in 1941 the game 'Nim' was implemented and in 1951 the first checkers and chess programs were created. Probably the most famous game playing agent was IBM's 'Deep Blue', which was the first computer program to defeat former chess world champion Garry Kasparov.

Game research is important because it produces many algorithms and techniques, such as search algorithms, which can be used in other fields as well. On the other hand, many other problems can in effect be seen as games, in which all agents try to maximise some score. Stock market is an example of this.

Whereas chess, checkers and many other full-information games have been mastered over the past decades, progress in computer poker has always been a bit behind. Even these days, the best poker bots are not able to win against the top human players. The main reason why playing poker at a high level is hard for computers players is the fact that poker is a game of imperfect information. Lots of relevant game play information is unknown to the players, such as all opponents cards and all cards that have not (yet) been revealed.

Another aspect of poker which makes it a challenging field of AI research, is the enormous number of possible states a poker game tree can have. In a heads-up game (i.e. a game with only two players) there are 2.781.381.002.400[1] possible card combinations. Considering the game-tree, if we were to calculate all possible opponent actions for all possible card combination they might have, the search tree would grow exhaustively large.

The amount of complications of computer poker is simultaneously the reason why it is such an interesting area of research. There is a lot to learn about dealing with imperfect information by studying computer poker.

Although there has been much progress in the field of computer poker, there has not yet been much published about poker bots which have a neural network as their main engine. The main reason for this omission is the fact that neural networks, once trained, are very much like black boxes. After they are created, they cannot be tweaked or changed. Furthermore, all knowledge in neural networks is implicit, so there never is an explicit reason for any taken action. These shortcomings of neural networks do not make them unsuitable for playing poker: it is in fact quite possible that neural networks are actually capable of playing a good poker game. In this paper, the possibility of using a neural network as the main component of a poker bot is being examined.

[1] $\binom{52}{2} + \binom{50}{2} + \binom{48}{5}$: our possible card combinations, possible opponents card combinations, possible board cards.

The outline of this paper will be as follows: section 2 gives an overview of current and previous work on computer poker. In section 3 the game of poker will briefly be described. All the preliminaries that have to be taken are described in section 4. The neural network training is dealt with in section 5. In section 6 the implementation of the poker bot is specified. Section 7 discusses the test results. Finally, section 8 draws conclusions out of these results.

2 Literature

Due to the popularity of the poker game, as well as the growing interest in computer science over the past decades, there has already been much research on computer poker. Much research has been done by the Computer Poker Research Group[2] of the University of Alberta, but there are other researches who have achieved a lot in the field of computer poker as well. In this section, a brief overview of the diverse approaches of creating poker bots shall be given. Especially the work of Jonathan Rubin and Ian Watson in their review paper of computer poker has been very useful.[7]

In the forthcoming paragraphs, various approaches to computer poker shall be illustrated. Firstly, a knowledge-based approach shall be examined. Secondly, Monte-Carlo simulation agents will be described. After this, there will be dealt with the ϵ-Nash equilibrium approach and finally exploitive agents will be examined.

Knowledge-based agents can roughly be divided in two groups: 'rule-based expert systems' and 'formula-based methods'. Rule-based expert systems consist of explicit if-then rules, for example 'if preflop and two hole cards have same rank, then call 5% of the times or raise 95% of the times'. Formula-based methods on the other hand generate probability for their upcoming action with a formula instead of explicit rules. Such functions normally have numbers like hand strength and pot-odds as input parameters. An example of a rule-based expert system is Loki[6], whereas its successor `Poki`[1] is an example of a formula-based agent. Shortcomings of these knowledge-based agents are their need of an expert to provide them with their knowledge and the fact that they tend to produce a rather static, and thus predictable, strategy, which will be less and less suitable as the poker game emerges.

The second approach to be discussed consists of Monte-Carlo simulation based agents. While knowledge-based systems typically lead to a static agent, dynamic search algorithms such as minimax provide superior results. The main drawback of ordinary search algorithms is their lack of dealing with imperfect information: in poker, it is never certain what de game tree will look like. Monte-Carlo simulation overcomes this shortcoming by

[2] http://poker.cs.ualberta.ca/

running a number of trials, averaging the outcomes of all actions and choosing the most optimal among them. The second version of Loki, Loki-2, used some sort of Monte-Carlo simulation[3]. Despite Monte-Carlo simulation's expected superiority over formula-based systems, a simulation-based implementation of `Poki` actually performed worse than its formula-based counterpart. Simulation-based `Poki` played a far too tight preflop and too aggressive on the last streets (turn, river).[2]

The approach which nowadays is most examined, is to search for ϵ-Nash equilibrium solutions. A Nash equilibrium, which is a concept of game-theory, is a game-state in which a change of strategy is unprofitable to all players. Poker agents that use this kind of strategy perform many abstractions in order to reduce the size of the game tree, otherwise the calculation of the ϵ-Nash equilibrium would be unfeasible. An example of a poker bot using this approach is `Sparbot`, developed by the University of Alberta. `Sparbot` is provided with a separate pre- and postflop strategy. Furthermore, different postflop strategies have been developed for `Sparbot`. These postflop strategies have eventually been combined in UoA's Hyperborean, which has been perhaps the most successful poker bot over the last years. Despite the success of this near-optimal strategy, a point of criticism one can make is that it tries to reduce the damage in a way. ϵ-Nash equilibrium search is a very defensive strategy: it will never try to exploit weak spots of its opponents.[7]

Strategies that do try to exploit weaknesses, at the cost of being exploitive itself, are exploitive counter-strategies. An example of an exploitive counter-strategy is imperfect information game tree search, using an adaptation of the `expectimax` algorithm. The `expectimax` algorithm is a variant of the `minimax` algorithm, which is able to deal with chance events. `Expectimax` is not yet able to handle imperfect information, in this case the unknown opponent cards, but this can be overcome by adding a proper opponent model. The overall accuracy of the algorithm is then dependent of its opponent model. Examples of agents that use this strategy are `Vexbot` and `BRPlayer` (both developed again by the University of Alberta). `Vexbot` and `BRPlayer` have both proven to play a good exploitive strategy as they were able to exploit weak spots in `Sparbot`'s near-equilibrium strategy.[7]

3 Poker Game

There are numerous variants of poker, of which Texas Hold 'em is by far the most popular.[8] In most poker games, the goal is to achieve a better poker hand than your opponent.[4] Another feature that is shared among all

[3] In fact, Loki-2 uses 'selective-sampling simulation': instead of purely randomised sampling, a selective bias is used in choosing a random sample.[7]

[4] There exist however variants of poker in which the worst poker hand wins.

Royal flush	A♠	K♠	Q♠	J♠	10♠
Straight flush	J♡	10♡	9♡	8♡	7♡
Four of a kind	4♠	4♡	4♢	4♣	J♡
Full house	7♠	7♡	7♢	Q♣	Q♡
Flush	Q♣	10♣	7♣	5♣	4♣
Straight	9♢	8♡	7♠	6♣	5♣
Three of a kind	9♣	9♠	9♢	K♡	2♠
Two pair	A♡	A♣	J♡	J♢	5♠
Pair	6♣	6♢	J♠	10♣	7♢
High card	K♡	J♢	10♠	7♡	4♠

Table 1: Poker hand ranks

variants of poker, is the ability of players to place bets. Subsequent players then have the ability to 'call', to 'raise' or to 'fold'. All bets are collected in the 'pot', which will be granted to the last remaining player or the player who has the best hand at the end of the game. If two or more players have equal hands (i.e. play the same poker hand, this does not necessarily mean them to have the same private cards), the pot will be split equally among them.

3.1 Texas Hold 'em

In the specific game of Texas Hold 'em, there are at most four betting rounds, which are called 'streets'. These streets are respectively the 'preflop' stage, 'flop' stage, 'turn' stage and 'river' stage. In between these streets, 'community cards' are revealed. The first three cards, which are added to the board after the preflop betting round, are called the 'flop'. The card after the flop is called the 'turn' and the final card is called the 'river'.

In Texas Hold 'em, all players receive two private cards. These cards are dealt face-down, so they cannot be seen by the other players. As mentioned before, the player with the best poker hand wins the pot. In Texas Hold 'em, a player can use any five cards of his two private cards and five community cards to form a poker hand.The least poker hand rank is 'high card', the best is the 'royal flush'. In table 1 an overview of all poker hands is given.

3.2 Limit Heads Up

Another way in which the various poker games differ from each other is the betting structure that is used. Most popular among (human) players is 'no-limit' poker, but there are alternatives such as 'pot-limit' or 'limit' poker. In limit poker, the betting and raising amounts are fixed. On the preflop and flop, one can only raise the amount of the big blind. On the turn and the river, the raising is fixed on twice the big blind. In limit poker,

there normally is a maximum of four raises (one bet, three raises) every street. The poker bots described in this paper will all be tested in a 1/2 limit structure (i.e. with small blinds of $0,50 and big blinds of $1,00).

Most computer poker research is focused on limit poker, as it is less computationally complex than no-limit poker: on each street there are just three actions to choose from (i.e. check/call, bet/raise, fold), whereas in no-limit poker games, in case of of a bet/raise, the amount of raise has to be chosen as well.

To reduce the complexity of the poker game even further, the form of 'heads up' poker, in which there are only two players, is chosen. In heads-up poker, a poker agent only has to deal with one specific opponent, so there are much less factors to take into account.

3.3 Strategy

Now that the basic rules of poker have been illustrated, this paragraph will look at what it takes to play a good poker game. According to Darse Billings et al, there are several features that a poker agent should be aware of, either explicitly or implicitly, to compete with world class poker players. These features are 'hand strength', 'hand potential', 'bluffing', 'unpredictability' and 'opponent modeling'.[1]

Hand strength is de current strength of ones poker hand. It is the level of probability that this specific poker hand will win over a random other hand. Hand strength only takes the current game state into account, so any forthcoming cards have no influence on its value.

Whereas hand strength only takes the current strength into account, **hand potential** assesses the potential that a hand will improve with forthcoming cards. For example, a hand with four cards of the same suit might not have a good hand strength, but has a great potential of improvement, for if a fifth card of the same suit is added, it will be a flush. On the other side, if one has a high pair, one will have a good hand strength, but if the there are many outs for the opponent, this hand will have a great possibility of devaluation. The probability of improvement is called the 'positive potential', the probability that a hand strength will decrease is the 'negative potential'.

Bluffing means pretending to be holding a good hand, whereas one in fact possesses a rather weak hand (or especially a hand with a low hand strength, but a good positive potential), in order to win hands with reasonably weak hands. The main purpose is to force opponents fold their winning hands and to win pots which otherwise would have been lost. The risk of bluffing is that an opponent could have a really good hand and therefore does not fold. In that case, folding on a later street must be considered.

Another form of bluffing is 'slow playing'. In slow playing, a player acts as if he is having a reasonable weak hand, but in fact has a very good hand.

In this way, a player tries to make its opponent believe it has a winning hand, whereas in fact he most certainly will lose. Slow playing is a way of trying to increase the amount of chips to be won.

Unpredictability makes an agent's playing strategy difficult to track. It is important to not always perform the same action with the same cards, otherwise opponent agents can easily predict what range of cards one must be holding and adapt their playing styles. A good poker player changes its playing strategy over time in order to be unpredictable.

The final important feature for a poker agent is **opponent modeling**. An opponent model gives the probabilities of the actions that an opponent may take. Simple opponent models can be used for all opponents, whereas better models adapt to specific opponents.

3.4 Complications of poker

The main difficulty of playing poker at a high level is the fact that poker contains much unknown information, such as the opponent's private cards and the cards that will come up in the course of the game. During the hand, one does not have direct information about those cards, so all information about them is given by actions.[5] Dealing with this imperfect information requires knowledge about probabilities.

4 Preliminaries

In order to create a neural network that is able to play poker, lots of real poker playing data is required. This data has to meet a number of criteria. Firstly, it must be complete. That is, all hands from the target player must be visible, even the ones that were mucked before showdown. Secondly, the target player must have a reasonable good poker strategy, because the neural network to be trained will mimic its playing strategy. Finally, the dataset should be of sufficient size. In poker consists of many different circumstances, so lots of training examples are necessary to learn them.

Every year, the 'Annual Computer Poker Competition' (ACPC) is held, in which poker bots from all over the world play against each other. Last year, 2011, twenty-one poker bots participated in the Heads-up Limit Hold 'em competition. All those bots played twenty-five matches against all other bots. To minimise the variance, another twenty-five matches were played, but this time the same cards were given to the other bot. So both bots had to play the same match with the cards previously owned by their opponent bot. The result of these matches were logged and published on the competition-

[5] In real life, there are other ways to gain information about ones opponents private cards. For example gestures or emotions, but those signs cannot be used in computer poker.

website. The total package consists of 18.491 match logs. Each match consists of 3.000 hands.

Although the provided database contains all relevant information about the matches, its format is not yet in the desired form to use for the neural network training. First of all, the information is stored as one string per hand, whereas not the entire hands are of interest to us, but the specific actions are. Second, all lines contain some irrelevant information, for example the number of the hand. Third, we are only interested in the actions performed by our target bot, so the actions by the opponent bot must be filtered.

In the preprocessing step, the raw data from the ACPC site is transformed into useful data that can be fed to the neural network training algorithm. The various features of the network are extracted and calculated. These features are:

Flop Whether the action is performed during the flop. (boolean)

Turn Whether the action is performed during the turn. (boolean)

River Whether the action is performed during the river. (boolean)

Last action Whether opponent has raised or not. (boolean)

Pot odds Ratio between amount to call and pot to win. (between 0 and 1)

Effective hand strength The strength of current hand, with forthcoming cards taken into account. (between 0 and 1)

Effective hand strength is a combination of the ordinary hand strength and the positive potential. It is obtained by formula 1.[1] Note that only the positive potential is taken into account, so negative potential will have no influence on the effective hand strength. This lack of negative potential actually results in desired behavior, for we want our opponents to pay for their draws, and eventually force them to fold their hand.

$$EHS = HS + (1 - HS) \cdot PPot \tag{1}$$

By far the most expensive algorithm in the preprocessing step is the calculation of the effective hand strength. This algorithm iterates through all possible combinations of opponent cards and possible cards to come to calculate our strength with regard to those situations. For every card combination, the hand rank function has to be called. On the flop, the hand rank has to be called 17.344 times.[6] On the turn, hand rank is called 16.262

[6] $1 + \binom{47}{2} + 47 + \binom{47}{3}$; our current hand rank (1x); all possible current opponent hand ranks ($\binom{47}{2}$x); all our possible future hand ranks (47x); all possible opponent future hand ranks ($\binom{47}{3}$x).

times,[7] and on the river 'only' 991 times.[8] An efficient implementation of the hand rank function is thus of great importance in both the preprocessing and the final poker bot.

The hand ranking function used in the preprocessing is 'Cactus Kev's Poker Hand Evaluator', which uses a very elegant and efficient approach in calculating the hand rank. Cactus Kev's algorithm represents cards as integers, in which the various bit-regions represent various features of the card. These regions are used to compute unique keys with very fast bitwise operations, to find hand ranks in precomputed tables.[9]

In order to preprocess the data as efficiently as possible, the various algorithms were implemented in C. Preprocessing one match file took about 5 minutes, which was still quite long considering the huge amount of files, but as not all files were to be used, it was good enough.

5 Neural network

All training of the neural network is performed in MATLAB's 'Neural Network Toolbox'. This specific environment is chosen because it supplies many different network configuration and train algorithms. Another advantage is the easy modification of the provided neural networks.

Various choices had to be made regarding the neural network. To begin with, one of the many neural network types provided by MATLAB had to be selected. Of all these types, `feedforwardnet`, `fitnet` and `patternnet` seemed most appropriate at first glance. `feedforwardnet` is the straightforward neural network implementation, of which `fitnet` and `patternnet` are specific instances. `fitnet` is mainly used for learning input-output relation functions: the target vectors can consist of any real number. `patternnet` on the other hand, is mainly a classification network: its target vectors should consist of all zeros, except for the class the input vector belongs to.

Regarding these networks, the `patternnet` network seemed most appropriate for the poker learning problem, as our targets (check/call, bet/raise, fold) can actually be seen as classes. A consideration that has to be made is that, even though our target vectors are class vectors (i.e. consist of zeros except for the target class), we do not want the neural network to produce class vectors. We actually want the network to produce a chance distribution over the three classes, otherwise the poker bot would have a definite strategy and would be very easy to exploit. Fortunately, we do not have to worry about this, because the target vectors themselves are distributed over all classes with some, previously unknown, probability distribution (i.e. similar input vectors do not always have the same class, there is unpredictability

[7] Similarly: $1 + \binom{46}{2} + 46 + \binom{46}{3}$.

[8] $1 + \binom{45}{2}$

[9] www.suffecool.net/poker/evaluator.html

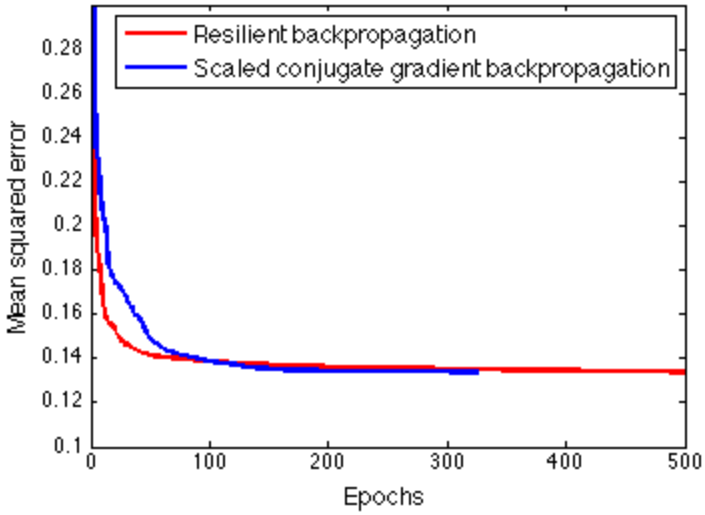

Figure 1: `trainrp` versus `trainscg`

in the input data), and we may therefore assume that the neural network converges over time to this probability distribution.

MATLAB offers lots of different training functions, which all have specific properties. The default training function is the 'Levenberg-Marquardt' algorithm, which is a quite efficient algorithm, but has the disadvantage of rapidly slowing down at processing large datasets, as is the case with our dataset. The algorithms `trainrp` (resilient backpropagation) and `trainscg` (scaled conjugate gradient backpropagation) were recommended in combination with `patternnet`.[4, pp. 2-18] Both functions were tested on a sample dataset of 500.000 instances to compare their performance. As can be seen in figure 1, both training functions achieved more or less the same result, although `trainscg` was slightly better and required less time. For those reasons, `trainscg` was selected to be used in training.

Another important feature of a neural network is the number of nodes in the hidden layer. Too few nodes can make a network inappropriate for a certain learning problem, whereas too many nodes may cause overfitting. Various amounts of nodes were tested on a sample dataset, the same that was used in choosing the training function. Networks with 6, 8, 9, 10, 11 and 12 nodes were tested. Table 2 gives an overview of the test results with both the mean squared error of the test set on the last epoch and the total number of epochs that were needed to create the neural net. All network configurations gave similar results, but the network with eleven hidden nodes needed significantly less time than most other configuration, so this amount of nodes was selected for the final neural networks.

To further improve the neural network, some more modifications were

	mse	Epochs
6	0,1355	268
8	0,1320	404
9	0,1318	407
10	0,1339	325
11	0,1329	255
12	0,1313	486

Table 2: Number of nodes

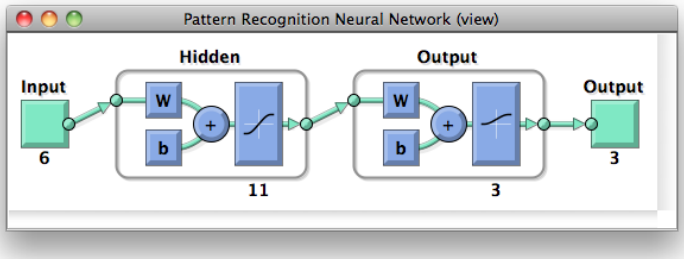

Figure 2: Neural network overview

made. Firstly, all pre- en post-processing functions were discarded in order to be able to use just the raw weight matrices and not having to bother with the internal mechanism MATLAB uses. Secondly, the output transfer function was set to `logsig` as all desired output values should be in between 0 and 1. Finally, some efficiency parameters were set, to speed up the neural network training a little. Figure 2 demonstrates the final network configuration.

Two different neural networks were trained. The first network was trained using a data set consisting of the first eight matches Hyperborean-iro played against all other opponents. The target was to learn the network a steady overall strategy, as it was fed examples of lots of different bots. The second network was trained with all matches Hyperborean-iro played against one specific opponent bot. The Sartre bot,[10] which achieved a second place in the 'Total Bankroll Competition'; was chosen as opponent bot. The intention was to learn this second network a more opponent specific strategy.

[10]Created by Jonathan Rubin and Ian Watson of the University of Auckland

6 Poker Bot

As a test environment, the Poker Academy Pro program was chosen. Poker Academy Pro has several advantages over other test environments. Firstly, its integration of the 'Meerkat API' facilitates a flexible way to build or import poker bots into the program. The Meerkat API is a Java library which provides tools for building poker bots, and that furthermore is required to add these poker bots into Poker Academy Pro. Secondly, Poker Academy Pro supplies lots of ways to statistically analyse poker games. And finally, Poker Academy Pro features several poker bots developed by the University of Alberta, the leading institution in the field of computer poker. These poker bots are probably the best poker bots available to play against.[11]

In the system folder of Poker Academy Pro, a simple poker bot example file was supplied which was used as a starting point for the neural net based poker bot implementations. This file, called `SimpleBot.java`, features a simple pre- and postflop strategy method. The `SimpleBot` source file has been transformed into new poker bot source files, encapsulating the neural networks. The first created poker bot is called `ANNBotFinal`, as it was supposed to be the final neural network. `ANNBotFinal` uses `SimpleBot`'s preflop strategy and has the neural network that was trained on all opponents as its postflop strategy. The second poker bot that was created is `ANNBotOne`. This bot bot is supplied with the neural network that was trained on all matches between Hyperborean-iro and Sartre. Like `ANNBotFinal`, `ANNBotOne` uses `SimpleBot`'s preflop strategy. The final poker bot, with an identical postflop strategy as `ANNBotFinal`, but with a more sophisticated near-equilibrium preflop strategy,[12] was `ANNBotFinalPF`.

In all poker bots, The postflop method was discarded and replaced with the newly trained neural networks. The networks are internally represented as two-dimensional arrays (or one-dimensional in case of the bias vectors). A simple matrix multiplication function was written to calculate the results of the input data. The resulting three values (check/call, bet/raise, fold) are normalised (so they sum up to one) and used to generate a randomised action.

7 Results

To test the neural networks, experiments were ran in which all created poker bots played against all bots supplied by Poker Academy Pro. To reduce the influence of variance and fuzz, all matches consisted of at least 14.000 hands.

[11] Even better would be to test a poker bot against real human players, but that is not really an option, as playing online poker is illegal, and a bad poker bot could lose lots of money.

[12] This preflop game approaches the Nash equilibrium, extracted from http://www.archduke.org/simplex/art

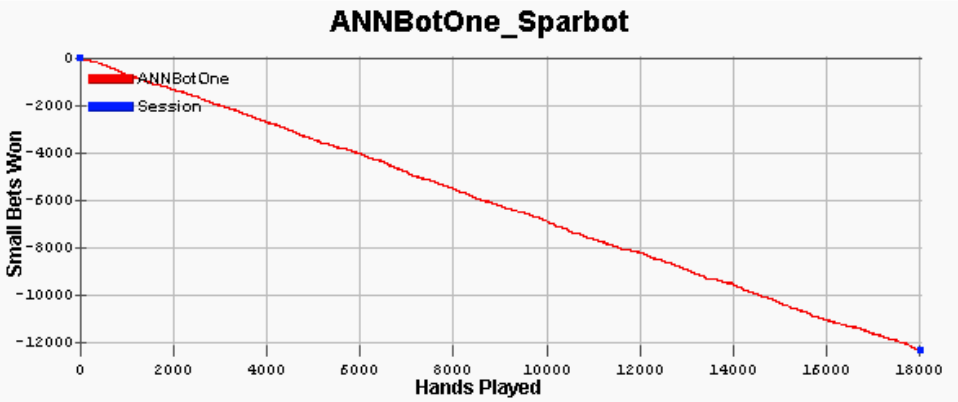

Figure 3: small bets won versus hands played

Unfortunately, Poker Academy Pro does not provide a random seed for the cards, so it was impossible to run every match with identical cards. For the same reason, a second match with swapped cards for both players was impossible. Nevertheless, Poker Academy Pro is a good test environment as has been pointed out in section 6. It even has been used as main environment in the 'World Poker Robot Championship'.[5] Furthermore, all small bets won versus hands played graphs show a very straight line as can be seen in figure 3, so it is believed that variance had little to no influence on the test results.

7.1 Opponents

The opponent bots provided by Poker Academy Pro, against which our bots are tested, are `Jagbot`, `Pokibot`, `Simbot`, `Sparbot` and `Vexbot`. All of them will briefly be described below.

Jagbot is a rather simple poker bot, which uses a rule-based strategy and no opponent modeling. It uses a hand evaluation function to generate a probability triple to choose an action. `Jagbot` is considered to be our weakest opponent.

Pokibot, developed after years of research by the University of Alberta, is perhaps the most well-known poker bot of all time. It is a very sophisticated bot, which adapts its strategy to its opponent over time by tracking their playing styles.

Simbot is a direct derivative of `Pokibot`, which replaces `Poki`'s formula-based betting strategy with many simulations to hand completion to determine the most profitable action.

Sparbot is a very tight, defensive player. It has a near-optimal strategy and is therefore hard to exploit. `Sparbot` uses a good balance of randomised mixed strategies for every hand, but will mainly focus on defence and not

	Jagbot	Pokibot	Simbot	Sparbot	Vexbot
sb/h	-0.9277	-1.4427	-0.3954	-1.3954	-2.5832
Preflop played	88%	87%	85%	92%	88%
Showdowns won	51%	53%	60%	53%	26%
Preflop aggression	0.9	1.0	0.9	0.9	1.0
Postflop aggression	0.3	0.5	0.5	0.4	0.5

Table 3: Results for `ANNBotFinal`

	Jagbot	Pokibot	Simbot	Sparbot	Vexbot
sb/h	-0.6209	-0.7860	-0.3477	-0.7460	-1.2655
Preflop played	35%	35%	27%	36%	37%
Showdowns won	55%	59%	67%	56%	36%
Preflop aggression	0.3	0.4	0.3	0.4	0.3
Postflop aggression	0.3	0.3	0.3	0.3	0.3

Table 4: Results for `ANNBotFinalPF`

try to exploit its opponent's weaknesses.

Vexbot is a poker bot that relies almost entirely on its opponent model. It is a heads-up only bot. It will have to play enough hands to have a good player profile of its opponent, and therefore will get tougher to beat the longer one plays against it.

7.2 Test parameters

The test results are summarised in table 3, 4 and 5. The values that have been depicted in these tables are 'sb/h', 'preflop played', 'showdowns won', 'preflop aggression' and 'postflop aggression'. These values will briefly be explained below:

'sb/h' stands for small bets/hand. A small bet is the same amount as the small blind, so in this case one small bet equals $0.50. 1.0 sb/h means that the agent wins an average of 1 small bet (thus $0.50) every hand.

The second value, 'preflop played', represents the percentage of hands the agent does not fold preflop. The higher this percentage, the looser the

	Jagbot	Pokibot	Simbot	Sparbot	Vexbot
sb/h	-0.5544	-0.7487	-0.4558	-0.6910	-1.1710
Preflop played	36%	36%	29%	38%	38%
Showdowns won	58%	61%	64%	58%	40%
Preflop aggression	0.3	0.4	0.3	0.4	0.3
Postflop aggression	0.2	0.2	0.3	0.2	0.2

Table 5: Results for `ANNBotOne`

playing style, so a player with a high 'preflop played' percentage will play relatively weaker hands.

'Showdowns won' is the percentage of hands that were won during showdown. So only the hands that were played up to the very last street and that resulted in a showdown are included in this percentage.

'Preflop aggression' is a measure of how aggressive the preflop strategy was. Its range is between 0 and 4, with 0 being the tightest possible and 4 the most aggressive.

'Postflop aggression' is similar to preflop aggression, but instead of depicting the preflop aggression level, it depicts the aggression postflop. The same range as preflop aggression is used.

7.3 Discussion

As can be seen in the results, all the trained bots are weaker than their opponent bots. The best bot loses 0.3477 small bets every hand (`ANNBotFinalPF` versus `Simbot`), whereas the worst bot loses 2.5832 small bets a hand (`ANNBotFinal` versus `Vexbot`). As a comparison: a hypothetical bot that folds every hand would lose 1.5 small bets every hand, so `ANNBotFinal` performs worse against `Vexbot` than an all folding bot!

Interesting to see is that the neural network bots apparently have a very exploitive playing style: `Vexbot`, who's strategy is based almost entirely on its opponents, achieves by far the best results against our bots.

Another thing we can conclude by the results, is that our bots which were provided with the `SimpleBot` preflop strategy, play far too tight. Both bots had aggression scores between 0.2 and 0.4 (with a maximum of 4), preflop and postflop. A reason for the postflop tightness might be that the neural networks were trained on `Hyperborean-iro`, which is a rather defensive agent.

It is further interesting to compare the two different preflop strategies in `ANNBotFinal` and `ANNBotFinalPF`. The overall performance of `ANNBotFinalPF` is much better than `ANNBotFinal`, as it as a better sb/h ratio against all of its opponents. The difference between both agents is clearly visible in their aggression scores: `ANNBotFinal` is much more aggressive, especially preflop, but remains more aggressive postflop, although it has the same postflop strategy as `ANNBotFinalPF`. This latter difference can be explained by the fact that `ANNBotFinal` has a broader range of hole cards postflop (it plays more specific hands than `ANNBotFinalPF` does.

The main reason for the bad results might be the lack of an opponent model among the input features of the neural network. Whereas `Hyperborean-iro` does have such input features, our trained bots do not. Therefore, it is impossible for the `ANNBots` to differentiate between various types of opponents and thus the network has in effect learned some strategy that is generalised over all sorts of opponents. The fact that `ANNBotOne`,

which was trained on only one specific opponent, achieved the best results (except for the match against the exploitive `Vexbot`), supports this assumption.

8 Conclusion

Considering the results illustrated in section 7, one could conclude that artificial neural networks are unsuitable as a main engine for poker agents. However, such a conclusion would be wrong to make. The author believes that it is actually possible to build a poker agent that uses artificial neural networks. This believe is enforced by achievements of Marv Andersen, who has built several well performing poker agents largely relying on neural networks. Unfortunately, little of his approach is known, so it is impossible to compare our approach with the one he used creating them.

To improve the results of poker bots with neural networks, it is suggested to add an opponent modelling unit to the inputs of the network. With such an opponent model, it is believed that the network is able to learn a more adequate input-output relation and thus be more able to generate a winning poker strategy.

The conclusion of this thesis is thus that neural networks might be able to play a proper poker game, but only if the right input features, such as effective hand strength, pot-odds and an opponent model, are supplied. These features are absolutely necessary for a reasonable level of poker, as has been illustrated by the shortcomings of the poker bots in this paper.

Glossary

Bet The first raise in a betting round. See raise.

Big Blind A forced bet, put before other actions take place. In heads-up poker, the big blind is to be paid by the player who is not the dealer.

Blind A forced bet, has to be paid in advance.

Bluff To pretend having better (or worse) cards than one in fact has.

Board All community cards.

Call To bet the same amount as the previous better/raiser. If the previous bet was zero, it is called 'to check'.

Check The action of betting zero.

Community Cards The cards on the board shared by all players. They consist of the flop, turn and river.

Draw Hands with a high potential of improvement, examples are 'flush draw' or 'straight draw'.

Flop The first three cards on the board, followed by the second betting round.

Fold To muck ones two private cards and lose ones chance on winning the pot.

Hand a) Both private cards of a player. b) One complete game of poker, from dealing the private cards up to the handing over of the pot.

Hole Card A private card.

Muck To discard ones two private cards.

Offsuit Two hole cards with different suits.

Outs The number of cards that can significantly improve ones poker hand.

Poket Pair Two hole cards of the same rank.

Postflop Betting rounds after the flop.

Pot Odds The ratio between the amount to call and the amount to win.

Preflop The betting round before the flop.

Raise Increasing the amount to bet, i.e. betting more than the last bet. If the previous betting amount is zero, it is called 'to bet'.

River The last community card that is added to the board, followed by the last betting round.

Showdown The stadium of the game in which all players reveil their hole cards and the winner is determined.

Slow-play To mimic the behaviour of playing with bad cards, whereas ones hole cards are actually very good.

Small Blind A forced bet, paid in advance of the start of the round. In case of a heads-up game, the small blind is paid by the dealer.

Suited Two hole cards with the same suit.

Turn The fourth (and second last) community card that is added to the board, followed by a betting round.

References

[1] Darse Billings et al. “The Challenge of Poker”. In: *Artificial Intelligence* 134 (2001), p. 2002.

[2] Aaron Davidson. “Opponent Modeling in Poker: Learning and Acting in a Hostile Environment”. In: ().

[3] Aaron Davidson. “Using Artificial Neural Networks to Model Opponents in Texas Hold’em”. In: (1999).

[4] H.B. Demuth, M.H. Beale, and Inc MathWorks. *MATLAB: Neural Network Toolbox*. MathWorks, Incorporated, 1994.

[5] Larry Greenemeier. *Poker-Playing Robots Battle For $100,000 Pot*. 2005. URL: `http://www.informationweek.com/news/165701734`.

[6] Denis Richard Papp, Duane Szafron, and Oliver Schulte. *Dealing with Imperfect Information in Poker*. 1998.

[7] Jonathan Rubin and Ian Watson. “Computer poker: A review”. In: *Artificial Intelligence* 175.5-6 (Apr. 2011), pp. 958–987. ISSN: 0004-3702.

[8] *The History of Texas Hold’em*. URL: `http://http://www.pokerpages.com/about-texasholdem.htm`.