ADAPTIVE POKER BOT BASED ON PLAYER-MODELLING

Bjrn Hasager Vinther & Nicolai Guldbk Holst
bhas@itu.dk ngul@itu.dk

A thesis submitted for the degree of

Bachelor Softwaredevelopment

May 2014

Abstract

Poker is a game which have a lot of different aspects which makes it a very interesting game to implement artificial intelligence (AI) into. The game is unlike games like chess and backgammon as poker has the problem that the information is imperfect. This is also one of the reasons why poker is interesting topic in regards to artificial intelligence as it opens up for some problematics as how to handle the information we have in regards to what we do not have. Because of this hidden information there are many decision to decide from, as we have to take into account all the information that we have access to and the possible information which is hidden from us. Another thing that makes poker interesting to have a look at is that we now have opponents who has the opportunity to trick us into thinking that he is in another state than what he actually is. In this bachelor thesis we will present a way to implement a neural network in a poker game to teach the bots how to play Texas Holdem Poker. The neural network will keep evolving along the opponents to ensure that the bot has the best possible chances of winning or minimize the loses. We will test the bots against real life human players in many suitable situations and from these results we can determine how effective our neural network is at teaching our bots how to play the game.

Preface

This is a bachelor thesis consisting of 15 ETCS points. The thesis has been written exclusively by Nicolai Guldbk Holst and Bjrn Hasager Vinther, both studying *Software development* at the IT University of Copenhagen. It has been written in the spring semester 2015. Our supervisor was Kasper Sty.

Introduction

Poker is the most popular card game in the world [9]. It involves intelligence, psychology and luck. Poker has a lot of hidden informations which requires the players to make decisions based on qualified guesses.

In a game of poker analysing your opponents is a major part. Being able to predict your opponents hands, future actions and reactions can give you a huge advantage. Top players are able to read their opponents and adapt their strategy in order to gain an advantage. Most rounds are won before the showdown simply by one player outplaying the others.

Another big element of poker is statistics. Since you don't know the cards that will be dealt throughout the game players must calculate the likelihood of their hand ending up being the winning hand. The more likely you are to have the winning hand the more aggressive you can bet to maximize your profit.

For humans it is often easy to recognise various patterns whereas it can be very challenging for a computer. On the other hand computers can perform thousands of computations in a matter of seconds. Even for most humans players it is really hard to read your opponents and it is even harder for a computer.

The goal of this thesis is to see if it is possible to train a computer to play poker and adapt to its opponents strategy. In order to achieve this goal we first need to answer the following questions:

1. How can we predict the probability of ending up with the winning hand?

2. Can the computer learn to play poker by observing humans playing poker, and if so how can we implement it?

3. How can the computer adapt to the opponents strategies?

Contents

1 Determine the strength of a poker hand

Our first step towards developing an adaptive poker is to finding a way to determine the strength of any given hand in any game state. In this chapter we will answer the question:

How can we predict the probability of ending up with the winning hand?

Since you don't know the outcome of the community cards during a round of poker you have to estimate your chance of winning based on the possible outcomes. The problem is that there are more than 250 millions different outcomes of community cards alone. Additionally the poker rules are quite complex when it comes to determining the wining hand so it's almost impossible to make a formula to calculate the exact probability. So how do you find the probability without having to check the outcome of 250 millions hands? The solution is to find an estimate instead of an exact probability.

1.1 Design

When solving this problem we have two options.

One way is to create a simplified formula to estimate the strength of the hand. The Chen formula is an example of this. This method is really simple to calculate but also less accurate.

Another way is to use the Monte Carlo method to simulate a lot of games and get an estimate of the probability. The more simulations you perform the more accurate the estimate you get.

For a human player the simplified formula would work best but in our case the Monte Carlo method is best suited. This is because a computer has no problem performing a lot of advanced computations. This method also gives the trade-off between accuracy and computations which allows us to decide how accurate we need the estimate. Other poker computers also uses this method.

We have developed our own subsystem to perform the simulations and return a probability of winning. We will refer to this subsystem as the calculator. It takes three arguments: the whole cards of the player, the community cards (optional) and the number of opponents. The calculator then performs the simulations and returns an object containing the distribution of outcomes.

Our requirements for the calculator are:

1. It shall be able to return the probability of winning for any poker state with up to ten players.
2. It must have a maximum error of one percent (deviation from caniwin[8]).
3. It shall calculate the probability in less than five seconds.

To find the right amount of simulations we tested the calculator with different numbers of simulations. From our tests we found that 50.000 simulations is the number of simulations best suited for the calculator.

1.1.1 Monte Carlo method

The Monte Carlo method can be used to calculate a distribution of results for a domain. This is done by creating a large amount of simulations with random inputs within a range of allowed inputs and note down the results of each simulation. These simulations are also called Monte Carlo simulations. The distribution of results can be used to find the likelihood of possible outcomes. The more simulations that are performed the more accurate the probabilities will get.

1.2 Test

To test if our subsystem calculates the correct probabilities we find the probability of a number of different pre-flop scenarios and compare the result to the results of caniwin. Every test have been preformed with 50.000 simulations against one opponent. The result can be seen in table 1.

We also test the accuracy of our Monte Carlo using different numbers of simulations. Figure 1, 2 and 3 shows the distribution of results from 50 tests. Each test is performed with a pair of jacks in pre-flop with one opponent. Caniwin found the probability to be 77,1 %.

In table 2 you can see the total result. The range is the difference between the lowest and highest result and the max error is the maximum deviation from caniwin.

hand	our result (%)	caniwin (%)	error (%)
Ac Ad	85,2	84,9	0,3
8c 8d	67,9	68,7	0,8
Qc Kc	62,8	62,4	0,4
Ah 8s	58,8	60,5	0,7
Js Qd	57,2	56,9	0,3
Th Jh	56,7	56,2	0,5
3d 3s	53,0	52,8	0,2
2d 2h	49,5	49,4	0,1
9d 3s	37,8	37,4	0,4
2d 7d	35,5	35,4	0,1
2d 7h	31,9	31,7	0,2

Table 1: Test results from different pre-flop hands

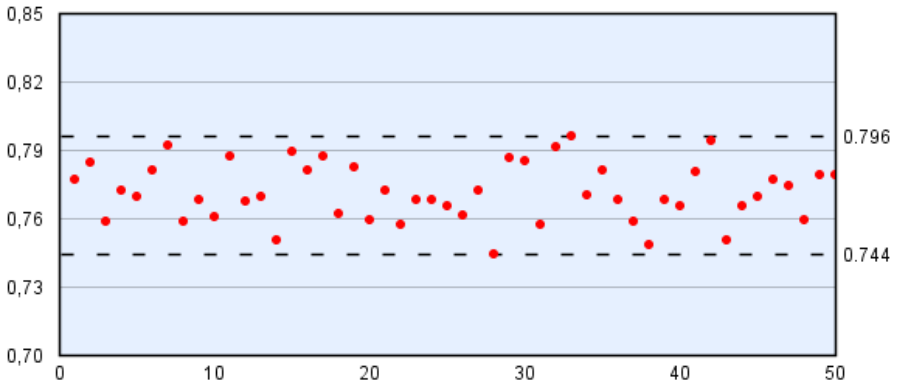

Figure 1: Result of the calculator with 1000 simulations

simulations	range (%)	max error (%)	time (seconds)
1000	5,2	2,7	0,03
10.000	2,2	1,5	0,22
50.000	0,9	0,7	0,82

Table 2: Test results from running Monte Carlo with different numbers of simulations.

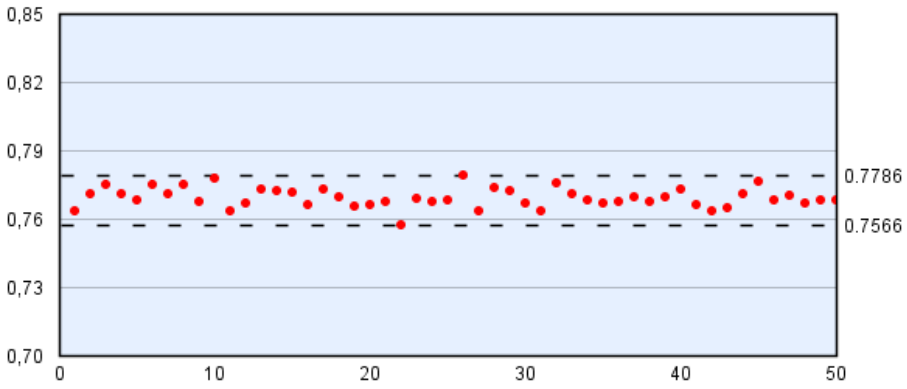

Figure 2: Result of the calculator with 10.000 simulations

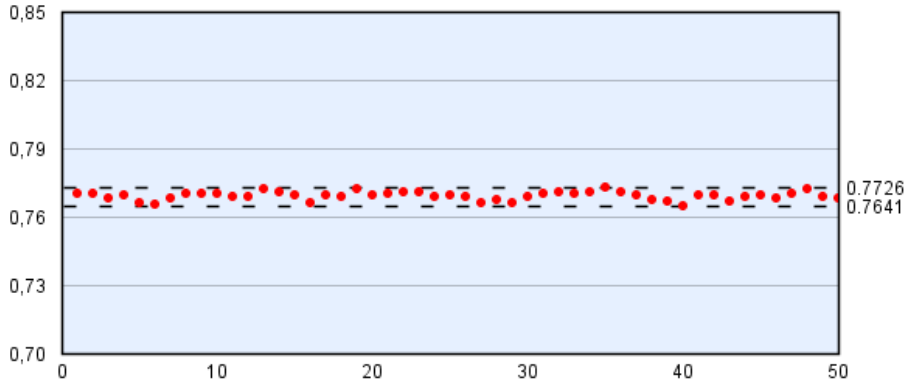

Figure 3: Result of the calculator with 50.000 simulations

We settled for 50.000 as the number of simulations for our calculator. This satisfies our requirements.

1.3 Discussion

1.4 Conclusion

2 Learning default play style

2.1 Introduction

2.2 Neural Network

2.3 Design

2.4 Test

2.5 Discussion

2.6 Conclusion

3 Adapt to opponents

3.1 Introduction

In attempt to make an artificial intelligence that is able to play along human players, we will have to adapt to opponents by modeling each player we are playing against. How can we use neural networks to model a player? And how can we use this to make our artificial intelligence a better poker player?

3.2 What is player-modeling?

If we want to make the best possible moves we will have to player model each player, to make the system adapt to their gameplay. Player modeling is basically when we model a player characteristics and the way they behave throughout the game to gain knowledge about the individual player. In poker there are many aspects of the game which could alter the players behaviour, such as position relatively to the dealer, the pot size, active players etc. Therefore it will be crucial to take atleast some of these things

into perspective when trying to model a player. One of the reasons why it is necessary to player model every player is because we need to have the opportunity to exploit a weak player. If one keeps making weak moves that we can predict it would be a wise consideration to exploit it. In a game of poker there will always be good and bad ways of handling a given situation, and this means there will also be an optimal way of handling each players moves. The pro players has to be good at adapting to an opponent. Even if the opponent changes their gameplay throughout the game, one must be able to adapt to this, and that is why we need to constantly player model each player.

3.3 How can we model a player dynamically?

To player model each player the system has to look at the games previous history while dynamic learning as the game proceeds. This can be implemented using a neural network and will help us reach our goal of predicting the opponents cards, next move or as a minimum what strength their dealt hand has. By using neural network we can model a player throughout the game. First the neural network of course take many inputs but one of them would be about a specific player and their history, the choices they made, their chips, cards etc. From that the neural network will continue to receive inputs about the player so that player modeling can be dynamic. Another big advantages of a neural network is that we are able to give a lot of inputs. These inputs will then be weighted by the system in order to come closer to our targeted output. This will help us cut out all of the noise that occur and leave us with data that is relevant.

3.3.1 Neural Network

To player model each player the system has to look at the games previous history while dynamic learning as the game proceeds. We decided to implement a neural network to achieve our goal of predicting the opponents cards, next move or as a minimum what strength their dealt hand had. This would be done by having the network look at many games played by that specific player. One of the big advantages of a neural network is that we are able to give a lot of inputs. These inputs will then be weighted by the system in order to come closer to our targeted output. This will help us cut out all of the noise that occur and leave us with data that is relevant.

Neural networks is models which are somewhat inspired by biological neural networks. Neural networks is considered to be one of the best methods to classify input-patterns. Neural network is for example used to recognize and reading handwriting and speech recognition. Humans are very good at recognizing visual patterns, but if we were to write a program that could do just that, it becomes alot more difficult. We have simple intuitions when we are trying to recognize different shapes. But to explain to a program how to recognize for example a Y would be something like a straight line with two lines pointing out from it at the top to each side at a 35 degree. When we try to make rules like this to let the program recognize letters we can quickly become lost in what we expect it to look like and the special cases. Neural networks has a different approach to the program which is much more suitable than describing each letter in some mathematical algorithm formula. We give the neural network a very large amount of handwritten letters, which

shall be used in the training of the neural network.

The neural network will then use the examples to automatically determine some rules for reading the handwritten letters. Of course this example doenst have a lot of different types of the letter A so if we want to let the neural network become better at reading the handwritten letter A we would have to give it an example with many more examples of the letter. It would improve the accuracy, therefore it is better to provide the neural network with a thorough example.

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs. A neural network consists of neurons which can send signals to each other. Each neuron processes its input signals and determines if the signal needs to be sent further. A neural network is not being fed with rules but instead examples that it can make rules from. With that a neural network is able to learn new skills, which is something that the traditionel computer cant do.

There are several types of neural networks and are being distinguished

by their type (feedforward or feedback), their structure, and the learning algorithm that they use. Feedforward neural networks will only allow the neurons to connect between two different layers. The feedback type of neural network will have a connection between neurons which are of the same layer but also between the different layers.

3.4 Test

3.5 Discussion

3.6 Conclusion

4 Discussion

5 Conclusion

References

[1] Texas Holdem Heads-Up Preflop Odds *URL: https://caniwin.com/texasholdem/preflop/heads-up.php*

[2] Marlos C. Machado, Eduardo P. C. Fantini and Luiz Chaimowicz *Player Modeling: What is it? How to do it?*, SBC - Proceedings of SBGames 2011.

[3] Aaron Davidson, Darse Billings, Jonathan Schaeffer and Duane Szafron, *Improved Opponent Modeling in Poker*, Department of Computing Science, University of Alberta.

[4] Garrett Nicolai and Robert Hilderman, *Algorithms for evolving no-limit Texas Hold'em poker playing agents*, Department of Computer Science, University of Regina, Regina, and Dalhousie University, Halifax, Canada.

[5] Jochen Frhlich, *Neural Net Components in an Object Oriented Class Structure*, ebook pp. 11-28

[6] Hold'em Review, *http://www.holdemreview.com/texas-holdem-heads-up-rules/*

[7] Lily Hay Newman, *Using AI to Study Poker Is Really About Solving Some of the Worlds Biggest Problems*, Slat blog

[8] probability of winning any pre-flop, *https://caniwin.com/texasholdem/preflop/heads-up.php*

[9] 10 most popular card games, *http://topyaps.com/top-10-popular-card-games*