Applying Machine Learning Techniques to an Imperfect Information Game

by

Néill Sweeney B.Sc. M.Sc.

A thesis submitted to the School of Computing,
Dublin City University

in partial fulfilment

of the requirements for the award of

Doctor of Philosophy

January 2012

Supervisor: Dr. David Sinclair

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Doctor of Philosophy (Ph.D.) is entirely my own work, and that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____

(Candidate) ID No.: 96970847

Date:_____

Acknowledgements

I would like to thank my supervisor, Dr. David Sinclair, especially for his patience.

I would like to thank my parents, especially for their financial support.

Abstract

The game of poker presents a challenging game to Artificial Intelligence researchers because it is a complex asymmetric information game. In such games, a player can improve his performance by inferring the private information held by the other players from their prior actions. A novel connectionist structure was designed to play a version of poker (multi-player limit Hold'em). This allows simple reinforcement learning techniques to be used which previously not been considered for the game of multi-player hold'em. A related hidden Markov model was designed to be fitted to records of poker play without using any private information. Belief vectors generated by this model provide a more convenient and flexible representation of an opponent's action history than alternative approaches.

The structure was tested in two settings. Firstly self-play simulation was used to generate an approximation to a Nash equilibrium strategy. A related, but slower, rollout strategy that uses Monte-Carlo samples was used to evaluate the performance. Secondly the structure was used to model and hence exploit a population of opponents within a relatively small number of games. When and how to adapt quickly to new opponents are open questions in poker AI research. A opponent model with a small number of discrete types is used to identify the largest differences in strategy between members of the population. A commercial software package (Poker Academy) was used to provide a population of sophisticated opponents to test against. A series of experiments was conducted to compare adaptive and static systems. All systems showed positive results but surprisingly the adaptive systems did not show a significant improvement over similar static systems. The possible reasons for this result are discussed.

This work formed the basis of a series of entries to the computer poker competition hosted at the annual conferences of the Association for the Advancement of Artificial Intelligence (AAAI). Its best rankings were 3rd in the 2006 6-player limit hold'em competition and 2nd in the 2008 3-player limit hold'em competition.

Table of Contents

1. Introduction

1.1.Overview

This thesis describes the application of a connectionist approach to poker AI. Poker is set of popular card and betting games. Designing a program to play any variation of poker provides an interesting challenge for artificial intelligence (AI) researchers. This is because hidden information is central to all of the variations of poker. Players have private information in form of cards that their opponents don't see and these cards are central to the strategy of the game. The board games that have been a previous focus for AI research do not have any hidden information; all players can see the full state of the game at all times. This means that the techniques that were successful on board games cannot be directly extended to card games. Hidden information is important because, when games are used to model real world situations (in economic, military or political applications), it is often central to the strategy as well.

Poker AI is a relatively new focus for research and there have been a number of distinct approaches tried. One approach that has not been explored much is the connectionist one. This appears surprising as neural nets (and other connectionist structures) have been successfully applied in a wide variety of applications. This broad approach is appealing in its simplicity. A vector of inputs is designed representing all the information that is relevant to the task. These are then used by a smooth non-linear function with many free parameters (usually called weights) to generate the required output. The weights are then optimised by repeatedly taking small steps which are on average in the direction that optimises a performance measure (i.e. applying the stochastic gradient ascent algorithm). Given the broad range of applications where such an approach has had success, a natural question to ask is: can it be adapted to poker? The fact that the stochastic gradient ascent algorithm naturally averages out noise suggests that it is a worthwhile line to explore considering that poker is a game with high variance. Another advantage of the connectionist approach is that it is purely reactive and generates outputs much faster than approaches that use time-expensive routines. This means that much larger sample sizes can be generated; again useful because large sample sizes average out the variance in poker.

The first difficulty presents itself when considering which inputs to give the structure. As stated previously poker is game that involves cards and betting. Designing a set of features that describe the state of the betting during a game is not difficult. Designing a set of features that describe the cards visible to a player is more tedious because the rules of the game relating to the cards is relatively complex (in most variations) but does not generate any intellectual difficulty. What does present a difficulty is the private information held by each player. A skilled poker player should deduce, from an opponent's actions, the likelihood that he holds certain private information. On first consideration, it is not obvious how to represent the results of these deductions as a vector of inputs to a connectionist structure.

The solution proposed here is to construct a hidden Markov model (HMM) of a deal of the game of poker. HMMs are commonly used in variety of applications where the true state of a system is not fully observable. They have not been used before in work in poker. In our model, the private information (cards) held by each player is replaced by a hidden variable called the card-type. The card-type variable used is a categorical variable with a small number of distinct values. At any point in the game, the model can be used to calculate a probability for each card-type and each player. This short vector of probabilities is then used as an input to the structure representing an approximation to the deductions a skilled player should make.

This structure is then used to address two distinct problems in poker AI. In both cases, the variation of poker studied is 6-player limit hold'em. Limit hold'em can be played with any number of players between two and ten but six is a popular choice. Payers act in a strict order and have at most three options at when it is there turn to act. (This is in contrast to no-limit variations where they have a continuum of options). A more detailed description of the variations of poker that have been studied is given in section 3.2. Some aspects of poker as played by humans will not be considered at all. It will be assumed that the game is played so that only the information explicitly expressed in the rules is available to the players. There will be no discussion of the "physical tells" available in "live" poker – physical tells refer to the mannerisms of a player that indicate his state of mind and can be observed by his opponents if they are playing around the same table. Also there will no discussion of table selection; the first decision in any game is whether to play at all.

While two and three player hold'em has received a lot of attention in the poker AI community, most of the techniques used do not easily extend when more players are involved. In contrast, a technique can always cope with fewer players. The first problem considered is one of generating “solid” play in the absence of any prior data about the opponents to be faced. More formally, we seek an approximation to a Nash equilibrium. To do this, self-play training is used. The structure is used to repeatedly play games of poker against itself and reinforcement learning is used to update the weights so that decisions that result in higher rewards are made more likely. As far as we are aware, this is the first use of Reinforcement Learning in training an AI for a popular version of poker.

The reinforcement learning proves successful, in the sense that it can be demonstrated that better decisions are selected more often as the training run develops. But the resulting bot is a poor approximation to an equilibrium because it is easy to find a bot that could win at a rapid rate against it. This is not a surprise as theory indicates that self-play learning does not converge to an equilibrium. Still it is interesting to see that these theoretical concerns have a large effect in practice.

This section of the project generated a series of entries (under the name dcubot) to the annual computer poker competition hosted at the AAAI conference. The competition is extremely useful as it allows different approaches to building a poker bot to be compared using evidence that is not in the control of any single researcher. While this is advertised as a competition, game theory tells us that it is not possible to always place strategies in a strict ranking. Every possible set of opponents offers a different performance measure to compare strategies. Despite the fact that the combination of self-play training and gradient based reinforcement learning is known not to converge to an equilibrium, the resulting entries have not proven easy to dominate in the competition. The competition includes contests in 2-player limit hold’em, 3-player limit hold’em and 2-player no-limit hold’em. The one occasion in 2008 when a 6-player limit hold’em competition was included is of particular interest as this is the game that is used in this thesis. Dcubot finished 3rd out of 6 in the 6-player limit competition in 2008. In 2009, it came 10th out of 11 in the 2-player limit competition and 6th out of 7 in the 3-player competition. In 2010, it came 2nd out of 5 in the 3 player competition. The result in 2008 is particularly interesting because subsequent analysis

by another competitor (Schweizer et al. 2009) suggests, it might have finished very close to the winner if some of the lower ranked opponents had been replaced with stronger players. In subsequent events it has been dominated by at least one of the best specialist 2 or 3 player entries. Further details can be found in section 3.2.1 and on the competition website www.computerpokercompetition.org.

The second problem is considered is how to adapt quickly to new opponents. The first step is to frame the problem so that a solution is feasible. If it is known that the new opponents are drawn from a population and a sample of play by that population is available then statistical thinking can be applied. (Statistics infers the properties of a population from a sample.) A simple performance measure is then available; the mean performance of the bot against new opponents drawn from the sampled population.

The designed structure mentioned previously is then used in two ways. Firstly as an opponent model that is fitted by the maximum likelihood principle. In fact two models are fitted. One models the mean strategy of the whole population. The second uses a hidden variable to group players into a small number of types to capture some of the variation between members of the population. This is a very common idea is very common in marketing, where it is called customer profiling, but has not been used much in poker AI. The desired bot (that has optimal performance against the population) is then trained by reinforcement learning as in the self-play experiments but with the one of the opponent models playing each of the other seats in the game. For the purposes of this thesis, this optimising bot is called *the exploiter*. When the profiling model is used, a belief vector over the opponent types is used as an input, so that bots can adapt to their opponents as they reveal their strategy.

To demonstrate this process, the commercial software "Poker Academy" (PA) is used to provide the population of opponent bots. It is advertised as ideal training for human players wishing to improve their game, so was considered as a challenging population to test against. A sample of play from the PA bots is generated. The opponent models are fitted. Various exploiter bots are trained in simulated games with the models. Finally, the exploiters play against real PA bots instead of their models. Short matches are used throughout; the opponents are redrawn and all memories wiped after only a small number of deals. The exploiter "knows" the opponents are drawn from the

population but nothing more at the start of each match. The aim of these experiments is to see whether using the profiling opponent model and adaptive exploiter can generate a significant improvement in performance over the mean opponent model and static exploiter in short matches.

To summarise the thesis in single hypothesis: Reinforcement learning can be productively applied to poker with an appropriate connectionist structure.

1.2.Chapter Outlines

Chapter 2 describes the various techniques during the work.

Chapter 3 the rules of hold'em poker, the alternative techniques that have been used in poker AI and a review of the published work in poker AI.

Chapter 4 describes the basic structure including the hidden Markov model.

Chapter 5 describes the results of self-play training. The performance is measured by comparing it to the performance of a slower but improved bot that takes a sample before each decision. This shows that reinforcement learning does improve the decision making. A best response is trained against the result of the self-play training. This earns chips at a high rate indicating that the self-play does not generate a good approximation to the equilibrium.

Chapters 4 and 5 were presented in an abbreviated form at the sixth European Workshop for Multi-Agent systems.

Sweeney N. and Sinclair D. 2008. Learning to Play Multi-Player Limit Hold'em Poker by Stochastic Gradient Ascent, EUMAS 2008, Bath.

Chapter 6 considers the question of finding a best response to a population of opponents. The necessary adjustments to the structure in chapter 4 are described. The commercial software "Poker Academy" (BioTools Incorporated 2007) is used to provide the population of opponents. A model is fitted to a sample of play from the population. A series of bots are then trained by reinforcement learning to optimise its performance in simulated hands against the model. Inspecting the results showed that

the adaptive systems did not generate substantial improvement over their non-adaptive counterparts. The chapter finishes with a discussion of why this might be the case.

Chapter 7 starts with a summary of the work. The main findings and the original elements are listed. It finishes by discussing some of the future lines of research possible.

2. Background

This work describes a novel structure for a poker playing program. This chapter discusses the underlying theory to many of the techniques later in the work. The structure will be used as an opponent model, so the first section discusses statistical model fitting. The structure will also make decisions in a game, so the next section discusses decision and game theory. (Game theory is an extension of decision theory when there is more than one decision maker interacting in a situation.) The approach used is a combination of a carefully designed connectionist structure and some basic machine learning algorithms. The final section discusses this very popular approach for developing AI systems.

2.1.Statistical Models

The first concept to mention is that of a probability distribution. At its simplest a distribution consists of a set (the sample space) and a probability associated with each element. Statistics usually involves inferring the unknown properties of a distribution from observations of that distribution.

One way of doing this is to design a generative model; a model which specifies the probability of all possible observations. One advantage of a generative model is that it only needs to be combined with a random number generator to create a simulation.

2.1.1. Model Fitting

A model usually includes a number of free parameters. The process of selecting the model parameters given a set of observations is known as model fitting. Usually this is done by defining some error (loss) function and then finding the set of parameters that minimise this loss for the observed sample. Something to be careful here is that the model doesn't overfit the sample. The temptation is to use a complex model with a lot of free parameters which can fit the sample with little or no error. But often this doesn't approximate the true distribution well as becomes clear when it is used to make fresh predictions outside the sample used to fit it. A simpler (smoother) model may have worse performance on the sample but it may predict better on new cases. It

is said to generalise from the data. A simple way to test for overfitting is not to use a portion of the sample when fitting but use it to test the model afterwards.

2.1.2. Maximum Likelihood

For any setting of the free parameters, there is a likelihood of any set of observations under a generative model. The Maximum Likelihood (ML) set of free parameters are the ones that maximise the likelihood of the observations. (The negative of the likelihood becomes the loss function to be minimised.) Maximum likelihood fitting has a number of attractive properties. It is consistent; in the limit of infinite observations the maximum likelihood parameters will approach the true values if the form of the model is correct. In fact it is asymptotically efficient; again in the limit of infinite independent observations, it makes most use of the available data.

The maximum likelihood principle is quite flexible. Sometimes other principles can be converted to a ML format. For example, a least squares loss function can be used by including a normal distribution in the model. Regression or discriminative problems can also be included. These are sometimes called supervised learning. In these problems, the variables are divided into two subsets; input or independent and output or dependent. The aim is to predict the output variables from the input variables. This can be framed as a generative ML problem by considering the inputs as modelled exactly which means the likelihood is solely given by the conditional probability of the output variables given the input variables.

2.1.3. Information Theory

A couple of calculations from information theory will be useful when describing the results of experiments. The first is the entropy of a probability of a distribution which is a measure of how much information remains to be revealed about a variable. (There are other interpretations of the entropy but this is the one we will use later.)

$$\sum_i P(x_i)(-\log(P(x_i)))$$

The connection with likelihood is clear; the entropy is the negative of the expected value of the log-likelihood. Maximising the likelihood is the same as minimising the

entropy. Usually the logarithm is taken to base two and the unit of entropy is then bits (binary digits). The minimum possible entropy is zero which occurs when the distribution is concentrated on a single value i.e. the variable is known with certainty. The maximum occurs if all values are equally likely.

A related calculation is the Kullback-Liebler (KL) divergence from one distribution to another,

where P is the "true" distribution and KL divergence measure the difference of the Q distribution from P. Again it is an expected value, if the sample is drawn from the P distribution. If Q is the distribution generated by a model then maximising the likelihood of a sample drawn from P minimises the KL divergence from Q to P.

$$\sum_i P(x_i)(-\log(Q(x_i)) + \log(P(x_i)))$$

More detailed discussion of these quantities can be found in any textbook on information theory e.g. (Cover and Thomas 1991).

2.1.4. Hierarchical Bayesian models

There is a common pattern to many of the models discussed in this thesis. When a modeller/decision maker is missing some information he would prefer to have, he models his uncertainty as a distribution over the unknown quantities and adds that to the model. This completes a generative model so that every observation has a likelihood. This distribution is often called the hyper-parameter prior but it can be discrete. When it is discrete it usually described as a distribution over types. Care has to be taken to ensure the model remains computationally feasible so simple forms are preferred.

The advantage of doing this is that the process of making probabilistic deductions from observations becomes mechanical assuming the model is correct. This is because the Bayesian update formula can always be used;

$$P(X \mid O_i, O_{i-1}, \ldots) = \frac{P(O_i \mid X) \times P(X \mid O_{i-1}, \ldots)}{\sum_x P(O_i \mid X) \times P(X \mid O_{i-1}, \ldots)}$$

where O_i is the latest observation and X is a random vector representing all the variables in the model.

2.1.5. Missing Data

Missing data problems refer to a subset of observation patterns. In these problems, the data consists of a record of a fixed set of variables for a number of cases but not all variables are revealed for all cases. In practical applications, missing data is a common problem.

Missing data causes no theoretical problems to maximum likelihood fitting. The likelihood of any case can be found by summing over the possible values of the hidden variables. But this isn't practical when the likelihood would have to be summed over a lot of values. Thankfully, missing data is a well-studied problem and there is a suite of algorithms available to be adapted to the specific problem. (Little and Rubin 1987) Where a maximum likelihood fit is sought, most algorithms are based on the Expectation-Maximisation (EM) family. (McLachlan and Krishnan 1997)

They are all based on two steps. Firstly construct a conditional expectation complete data likelihood function. This is a function of the free parameters where the expected value of the complete data likelihood is calculated by summing over every possibility of missing data using the probability of each given the observed data and the current estimate of the parameters. This is called the E-step. The M-step consists of maximising the function generated in the E-step over the free parameters. This forms an iterative scheme which converges to the ML estimate. It is well known the M-step does not have to be a complete maximisation; any alteration that increases the function is sufficient (McLachlan and Krishnan 1997). This is known as generalised EM. If an iterative method that only improves the EM function by a small amount at each step is used then ideally the E-step should be quick and approximate as well. Drawing a sample from the posterior distribution given the current parameter estimates and the revealed data is ideal. This filling in of missing data is called imputation. If the imputed examples are drawn from the distribution implied by the current settings of the parameters this is a valid EM method and should converge to the ML parameter values. Imputing from another distribution doesn't have the same guarantees.

2.1.6. Hidden/Latent Variables

These are variables that are never observed. Hidden variables are added to a model because they can simplify it by reducing the number of free parameters and making it computationally feasible. A saturated model is one that can express every possible distribution over the possible values of a set of variables. If there are a large number of variables in a dataset it is not feasible to work with a saturated model. The number of free parameters will be an exponential function of the number of variables; meaning an unreasonably large amount of data will be required to fit the model and making it very slow to perform any calculations. Adding extra variables in such a way that the many observables are independent given the hidden variables makes a simpler model which is feasible to work with. These are called mediating variables and they are said to divorce the observables (Pearl 1988, Jensen 2001). Hidden variables are particularly useful in applications with sequences of data. Often a hidden Markov state is added to the model with observations being independent given the state; the hidden state is said to have the Markov property if two states are independent given the state at any time in between. Fitting a model with hidden variables causes no extra difficulties to maximum likelihood methods; hidden data techniques still apply.

Sometimes the type of hidden variable to use and its dependence on the other variables is suggested by a detailed theory of the application domain. Often though there may not be a detailed theory. Instead the model designer selects the positioning, type and dimension of the hidden variables. A good learning algorithm can then learn to make the best use of the hidden variables it is given i.e. to fit the strongest dependencies between the observables.

A simple type of hidden variable we use is a categorical variable which divorces many observables. It is natural to think of each setting of the variable as a type which determines the other variables except for some independent random variation. Another way of thinking about such variables is that each setting of the variable represents a cluster of cases in the database with similar observations. Learning these type (or cluster) variables can be prone to a particular type of poor local optima. These are where one type collapses onto a single case. This can usually be identified quite easily

afterwards. Correcting the problem is usually not too difficult though; sometimes simply drawing a different starting point will work.

2.1.7. Sampling

Using imputation to fit a model is one situation in which one might want to draw a sample from a specified distribution. There is a large body of research in this area but for this thesis only the most basic methods are used. Most sampling techniques are used in situations where the probability distribution can be calculated easily except for the normalisation constant (the factor that ensures the distribution sums to one as all probability distributions should). This often arises when working with a conditional distribution when the normalisation constant is the sum over all values of the condition.

If the variable to be sampled has a finite number of cases, a sample can be found by enumeration. To find the normalisation constant, the probability of each case is summed. Drawing a sample is then straightforward; first draw a value from a U[0,1] (a uniform distribution between zero and one) and then run through the values of the variable summing probabilities until it exceeds the uniform value drawn returning the last value. Of course this method is only feasible where the variable to be sampled has a relatively small number of values, otherwise it takes too long.

Rejection sampling is another simple method of drawing from a distribution. It starts by sampling from a proposal distribution. The proposal distribution (g(x)) should be easy to sample from. A value M must be found such that $\frac{f(x)}{g(x)} \leq M$ (where f(x) is the desired distribution).

A random number is also drawn from U[0,1]. If $u < \frac{f(x)}{Mg(x)}$ the current example is selected, otherwise the example is rejected and the process is repeated. This process will generate an i.i.d. sample from the distribution f(x).

For continuous variables finding the bound M can be difficult but this is not the case for discrete variables. An obvious difficulty is that sampling may take too long to

generate the examples. The probability of accepting an example is $\pi = \frac{f(x)}{Mg(x)}$. So the expected number of examples tried before accepting one is $\frac{1}{\pi}$. This is a value that any user of rejection sampling should be aware of and preferably be able offer an upper bound.

An alternative method of sampling is possible when rejection sampling is possible but a large sample is required from a distribution. Instead of accepting or rejecting each proposed value, a fixed number (n) of values are drawn from a proposal distribution and each is weighted by its acceptance probability (π). The final sample could be drawn from the proposed values by selecting with probability $\frac{\pi_i}{\sum_i \pi_i}$. More often though this weighted sampling is used when an estimate of the mean of the distribution is desired $\bar{x} = \frac{\sum_i x_i \pi_i}{\sum_i \pi_i}$. Note that this estimate is biased (its expected value is not the same as the true expected value) but it is consistent (it converges to the correct value for large number (n) of proposed values). The variance of a weighted estimate of the mean such as this depends on the weights and is given by the formula $Var(\bar{x}) = \frac{\sum_i \pi_i^2}{\left(\sum_i \pi_i\right)} \sigma^2$ (where σ^2 is the variance of the distribution to be sampled).

Where there is a large variation between the acceptance probabilities, the variance of the estimate of the mean is larger than it would be for the same sample size if the sample could be taken directly.

Markov Chain Monte Carlo (MCMC) methods are more sophisticated methods for generating a sample. These generate a sequence of sampled values which eventually converges to the desired distribution. Identifying when a sequence has converged is not straightforward but if the application allows long sequences, MCMC methods are attractive. MCMC methods were considered but not used in this thesis. A review of MCMC techniques can be found in (Andrieu, et al. 2003).

2.1.8. Central Moments

The population mean (or expected value) is given by the formula.

$$E(x) = \sum_i P(x_i).x_i$$

This is the first central moment.

Note that it is only defined for a particular distribution.

The variance is defined by the related formula.

$$\sigma^2(x) = E\left((x-\mu)^2\right)$$

This is the second central moment. The standard deviation is the square root of the variance which means it has the same units as the mean. If an independent and identically distributed sample is available, it is possible to estimate the true mean and variance of a distribution with

$$\bar{x} = \frac{\sum_i x_i}{n} \quad s^2 = \frac{\sum_i (x_i)^2 - n\left(\sum_i x_i\right)^2}{n-1}$$

which are the best unbiased estimates. The central limit team says that in the limit of a large sample size; the mean of an independently and identically drawn (i.i.d.) sample is a normal distribution with the same mean as the original distribution and a standard deviation given by

$$\frac{\sigma}{\sqrt{n}}.$$

We will set-up our experiments so that the top level of the experiment consists of i.i.d. replications and report the mean and standard deviation.

It is possible to define more central moments but the central limit theorem says that in the limit for large sample sizes, the early moments dominate when summing i.i.d. variables.

A sample generated by one distribution can be used to estimate the moments of another distribution. This is called importance sampling and will be described in Section 2.1.7.

2.1.9. Ratios for assessing a risky strategy

An important ratio in assessing the performance of a poker strategy is the z-score (statistics) or Sharpe ratio (finance): $\frac{\mu}{\sigma}$. For any large fixed number of repetitions, increasing the z-score will increase the probability of having made a profit (assuming the strategy is profitable). Alternatively, the higher the z-score the quicker a profitable strategy will show that profit for any fixed significance level. We will estimate the true ratio with the ratio of estimates. (The ratio of estimates isn't the best estimate of a ratio but in the results we look at later this isn't a major effect.)

Another ratio to consider is $\frac{\mu}{\sigma^2}$. This quantity is important in Gambler's ruin problems. In these problems, a gambler repeatedly makes a series of identical and independent gambles until his fortune reaches either an upper or lower limit. If both limits are far enough away from his current fortune, the probability of finishing on the upper rather than the lower limit is an increasing function of this ratio (Kozek 1995).

Hence a gambler who wants to win as quickly as possible should aim to maximise his mean return; a gambler who wants to show that his strategy is profitable as quickly as possible should optimise the Sharpe ratio; a gambler who wants to reach a certain target with as little risk as possible should optimise the gambler's ratio.

In practice, a professional gambler should subtract his expenses before calculating either ratio, just as a financial investor should subtract the cost of funds from his investment return before calculating the Sharpe ratio.

2.2.Decision Theory

Decision theory recommends starting each problem by specifying the relative desirability of outcomes in the form of a utility function. Von Neumann and Morgenstein (Von Neumann and Morgenstern 1944) showed that a rational decision maker faced with decisions with uncertain outcomes and obeying some natural conditions (axioms) should act as if maximising the expected value of some utility function. In risky problems, the exact form of the utility may be important. Two functions which rank all possible outcomes the same can have different expected values unless one is an affine function of the other.

Completeness.	All outcomes can be ranked.
Transitivity.	Cycles of preferences are not possible.
Independence.	Preferences are maintained if they are mixed (probability distribution) with the same probability and alternative.
Continuity.	If an outcome is ranked between two alternatives, there is a lottery over the two outside alternatives which is considered equally preferable to the middle one

Table 2-1: Utility Theory axioms

Then a rational decision-maker should list all possible strategies, calculate the expected utility for each and select the choice with the highest utility. The interesting problems occur when this is not feasible. There are two reasons why this might be; firstly it might not be feasible to enumerate all strategies, secondly some aspects of the problem are not known for certain i.e. some statistical inference is required.

2.2.1. Markov Decision Problems (MDP's) and Partially Observable Markov Decision Problems (POMDP'S)

Markov decision problems (MDP's) are formalism for sequential decision problems. They consist of a set of states (one of which is the initial state); a set of actions that are possible in each state; a transition distribution over the next state given the current state and an action; a reward function specifying for each combination of current state,

next state and action. If there is a terminal absorbing state which cannot transition to any other state and which has zero reward, the decision problem is known as episodic. When you reach the terminal state, the task is over and your overall reward is the sum of the individual rewards on each transition from the initial state. If there is no absorbing state, the overall reward is usually expressed as a decaying sum of future individual rewards

$$R = \sum_i \lambda^i r_i \ .$$

A strategy for a MDP is a function giving a distribution of actions given a state. A hard strategy selects a single action in each state. The Markov property means that only the current state needs to be considered as all transitions are independent of previous states given the current state. This formulism can be made cover a wide variety of decision problems.

The simple enumerate all strategies approach usually isn't practical for two reasons; there is an exponential explosion in the number of strategies as the number of states or actions per state increase and if episodes tend to be long (or in continuing tasks) it can take a lot of computation to evaluate a single strategy as there will be a lot of transitions to work through. Dynamic programming streamlines the computation to reach the same solution as enumeration quicker. This assigns a value to each state so that an actor only has to calculate forward one step to select the best action.

A related formulism is a partially observable Markov decision process (POMDP). Added to the structure is a distribution of observations for each state transition. (One of the observations must be which actions are legal). The decision maker cannot tell exactly which state he is in but can use the observations. A strategy is a distribution over actions given the sequence of observations since the start of the task.

Any POMDP can be converted to MDP by replacing the (hidden) state with a belief state; the current distribution over hidden states given the observations. The down side is that even if the original problem had a small number of discrete states, the belief state is a continuous vector of the same dimension as the number of original states

2.2.2. Value of information

An important concept in decision theory is the value of information. This calculates the value of any observation. This is found by comparing the mean reward from selecting the best decision for each case that could be observed and the best reward possible if the decision had to be made before the observation. This requires a lot of computation in all but the simplest examples but the concept is still useful. An observation cannot have negative value under this definition but certain criteria must be met for it to have positive value; the information must be available before some decisions are made and the best strategy must be different for some cases. (Smith 1988)

2.3.Game Theory

A game can be specified in normal (or strategic) form; this consists of a list of players (at least two), a set of legal pure strategies for each player and a utility for each player for each combination of legal strategies. All of these must be common knowledge. The normal form is usually not a very efficient representation of a game but any other representation can be converted to this format.

An alternative description of a game is the extensive form. It is particularly useful for capturing the order in which decisions are taken. This consists of a tree of nodes. Some are chance nodes representing random events. The distribution over subsequent nodes is common knowledge. There is a single initial node and a number of terminal nodes. For each terminal node there is a utility for each player. The other nodes are partitioned into information sets. Each set represents a distinct decision for a player and there is an associated list of legal choices. For each legal choice, all the nodes in an information set transition to subsequent nodes.

If the information sets are all singletons, the game is classed as perfect information. Otherwise the information sets represent the uncertainty that any player faces when making a decision.

A pure strategy in an extensive game consists of a legal move at every information set of a player. A mixed strategy is a distribution over pure strategies. (Assume strategy means mixed strategy unless otherwise stated.)

If the strategies of all other players (the opponents) are fixed, a strategy for player A is a best response (to the opponents' strategies) if no other strategy generates a higher utility. A pure strategy that is not a best response to any combination of opponent strategies is classed as dominated. A process of repeatedly eliminating dominated strategies for each player in turn until no more can be eliminated is known iterated dominance elimination. A best-response may be brittle; it will do well against the opponents it was designed for but poorly against other opponents.

2.3.1. Equilibrium

A Nash equilibrium is a strategy profile (one strategy for each player) such that no player can improve his utility by changing his strategy. Nash proved that at least one exists for every game. An equilibrium can only include strategies that remain after iterated dominance elimination. In a zero-sum 2-player game, the equilibrium strategies form a convex set where all combinations of strategies have the same value. A game is zero-sum if the sum of all utilities is a constant at every terminal node. In the 2-player zero-sum case this means that the player's aims are in complete opposition. In all other cases there may be more than one equilibrium, each with different values so that the players care which one is played. A strategy profile is called an ε-equilibrium if no player can increase his utility by more than ε by changing his strategy. The smallest ε such that a strategy profile is an ε-equilibrium can be used as a measure of how close a profile is to equilibrium. It can be difficult to calculate though as it involves finding the best response for each player to the rest of the profile.

2.3.2. Evaluating strategies

An easily overlooked aspect of games is that it isn't possible to score strategies on a single dimension. This point is made in a discussion paper (Halck and Dahl 1999). Every distribution of opponents forms a separate decision problem and hence a separate dimension on which a strategy can be scored. Hence, it doesn't make sense to say one strategy is better than another in general. There are two simple evaluation criteria that can be used. The first is to select an opponent population and then to evaluate strategies by estimating their mean when used against that population. This

will be used in chapter 6. The second is to evaluate each strategy by its worst performance against any opponent strategy. In a two-player zero-sum game, the strategy with the best worst-case (usually shortened to maximin) performance is the equilibrium strategy. The difficulty with this evaluation criterion is that it involves finding a best response to the strategy that is being assessed which is often a difficult problem itself.

2.3.3. Perfect Information

Perfect information simplifies the analysis of games considerably. There is no hidden information so there is no need infer that hidden information. In a 2-player zero-sum game, iterated dominance elimination will leave a single pure strategy (Zermelo's Algorithm). Most classic board games are examples of 2-player zero-sum perfect information games e.g. chess, chequers (draughts), backgammon. While Zermelo's algorithm can be used to prove an optimal strategy exists, it is not practical in these games because the game tree is so huge in most classic board games. Instead most successful algorithms, approximate the true game tree with a tree starting at the current node and working as many moves forward as is feasible; if it cannot reach a the terminal nodes on any branch, an evaluation function is used to give an approximate value to the last nodes reached. In chess and chequers, the first successes used quite simple functions but a deep search. In gammon, a deep search is not possible because future decisions depend on the roll of the dice. Tesauro (Tesauro 1995) has considerable success using shallow search (4-6 moves) but a complex evaluation function computed on a neural net and learnt by self-play.

Go is a game that has resisted this approach which suggests something else is important in this game.

2.3.4. Imperfect Information

Imperfect information usually comes into games into two ways. In simultaneous move games, players have to act without knowing the last action of the other players. This will be the case if there is no rule enforcing a strict order of play. (There will be an order of play in the extensive form but this will be arbitrary. Any order can be used with different information sets.) Asymmetric (or private) information is relevant

information that is only available to some of the players. This also changes a game to one of imperfect information. Most card games involve asymmetric information as some cards are only visible to some players but not simultaneous moves. Asymmetric information games are also used extensively in economics to model real-world situations.

The definition of a game implies that while the strategy a player intends to play is unknown, both his capabilities (which strategies he could play) and his desires (as utilities on terminal nodes) must be common knowledge. When modelling economic situations this isn't often realistic. A common way round this is to model the variation between players as a distribution over types. At the start of the game, the type of each player is drawn and each player is informed of their own type (private information). As long as the distribution over types is common knowledge this corresponds to a game. The resulting game is known as a Bayesian game and the process is known as Harsanyi completion (Binmore 1992). Imperfect information games formed in this manner are common in economic literature. A classic example that is much cited in economic textbooks is Spence's model of employment and education.(Spence 1973). This explains how completing an arduous course of education might be useful to a job applicant even if the education itself did not improve his performance on the job.

2.3.5. Repeated Games

Repeated games involve repeatedly playing an identical game (called the *stage game*) to form a larger game (called the *meta-game*). Poker is usually played as a repeated game. Each deal isn't identical because the starting player changes but each circuit of deals is. So a circuit corresponds to the formal definition of a stage game.

Associated with any game and player in that game, is a 2-player zero-sum game. This is formed by assuming all the other players are operating as a team whose utility is the negative of the solo player. The solo player's strategy is called a *security strategy* and the value of the game is called the *security value*.

The Folk Theorem states that any set of strategies can form part of the Nash Equilibrium in a repeated game, if all players receive more than their security value (in the limit of little discounting). The set of returns such that every player has a return

above their security value is known as the *rational set*. Inside this set Nash equilibrium offers no guidance as to which strategies will be played under the conditions of the Folk Theorem. Bargaining theory though does have some guidance; it suggests the players with the highest tolerance for risk will grab the biggest share. An implicit assumption in most work in multi-player poker is that the feasible set is small relative to other errors made by the programs. This would mean considerations of how to bargain over the feasible sets can be safely ignored. (Heads-up poker is 2-player and zero-sum so the feasible set is a single point anyway.)

If poker is considered as a Bayesian game with types drawn before any deal, then poker is no longer a strictly repeated game. Instead the meta-game resembles the individual deals; during each deal a player tries to infer the other players' cards while over the course of a match a player tries to infer the other players' types. In this situation, it will (in general) be necessary to change strategy between stages in an equilibrium strategy.

2.3.6. Exploitation

Any game can be converted to a symmetric game by having each player draw for which player in the original game he will play (Halck and Dahl 1999). Equilibrium strategies are an attractive concept in game theory. Repeatedly playing an equilibrium in the stage game forms an equilibrium in a repeated game. So a natural question to ask is; what are the justifications for not playing a static symmetric equilibrium in a repeated game that has been symmetricized? Exploitation refers to using a non-equilibrium strategy to generate a better return. In this setup the player trying to achieve this higher return is called the exploiter and the other players in the game are called the opponents.

Many of the reasons for not using a symmetric equilibrium are because the game only appears to be a symmetric repeated game. Firstly the game may only appear to be symmetric. At the start of most poker matches, the players draw for position around the table. This only appears to symmetrise the game. The position determines the order in which players act but this is only one of the properties of a player; the others such as

the utility function of a player (or a players' computational ability) cannot be randomised.

Equally a game may only appear to be a repeated game. If Harsanyi completion is used on a repeated game (all stages would be identical if the characteristics of the players were known) then the resulting Bayesian game ceases to be a repeated game. The play of each stage may provide evidence about the players' characteristics which could provide useful information to the players on future stages.

The simplest case where an exploitative strategy is advisable is where the opponents are known before the start of the match. In which case, a best-response should be sought instead of playing an equilibrium strategy.

If the opponents are limited to static strategies and a very long sequence of stage games is to be played then the game reduces to a decision problem (only one actor has any choice) suitable for reinforcement learning. This is because while the exact nature of the decision problem to be solved is not known at the start, the exploiter can generate (a large number of) experiences to learn from. As such it is reasonable to look for good asymptotic performance as can be achieved in other reinforcement learning problems. The eventual result of the learning will be a (static) best response to the opponents.

Another scenario where a static equilibrium strategy is unattractive is if the opponents are drawn from a known population distribution, but not identified at the start of the repeated game. This population best response may be an adaptive strategy even if the opponents are static. This is because the exploiter may be able to improve his performance by identifying which opponents he faces from their play in the earlier deals and adjusting his play in the later stages accordingly. In practice it is unlikely that the opponents' strategies will be known precisely but if a sample is available a model may be fitted. This is similar to the multi-armed bandit problems where the reinforcement learning problem faced is drawn from a known distribution. Gitten's indexes have been calculated for various known distributions (Gittins and Whittle 1989).

Finally if players change strategies between stages, they can coordinate their actions in a correlated equilibrium. A correlated equilibrium can have better rewards for all players. This is especially the case in non-zero sum games where there may be a large incentive for cooperation.

Exploitation receives attention in both popular (Sklansky 2005) and academic work (Billings, et al. 2003) on poker. There a number of good reasons for this. It resembles a Bayesian game. Player's utilities differ; their appetite for risk or their enjoyment of aspects of the game for example. Their computational abilities can differ also e.g. how easily they calculate the relevant probabilities. Another reason is that actual poker resembles a statistical decision problem. Typical players will play against a large number of opponents. As such it would be difficult for them to play a completely distinct game against every opponent. Examples of their play should be suitable for statistical analysis which will generalise their play across opponents. This does not mean they play the same against all opponents; just it should be possible to generalise from their play against other opponents.

When considering exploitation, it is tempting to look for "clairvoyant" solutions. But it is not possible to know an opponent's strategy (and hence the best response) without that knowledge coming from somewhere. If a long sequence of games is played against a static strategy then it is possible to learn the best response eventually. Even in a complex game, a static opponent must reveal his strategy to an exploiter using a strategy with infinite exploration. This will be called an asymptotic best response in this thesis. If the opponent is drawn from a known distribution of similar opponents then some of information can come from the properties of the distribution which are known before the start of the repeated game. The only missing information is which members of the opponent population the exploiter is currently facing. Hence rapid adaptation is feasible as there may only be a small amount of useful information missing. This will be called a population best response in this thesis.

2.3.7. Self-play training

The simplest way to solve a 2-player zero-sum game is to express it in normal form (as a matrix) and use linear programming (LP) to solve the resulting equations and

constraints. General sum 2-player games can be solved by linear complimentary programs. 2-player perfect information games can be particularly easy to solve because there is no need to mix strategies.

Using (LP) to solve the game isn't always feasible because the size of the matrix enumerating all strategies for both players can be enormous for many games. Writing the game in sequence form can reduce the size of the description.

Most games of interest are too large to be solved exactly and this is the case with the game of multi-player hold'em studied here. The self-play approach described next is used here. Alternatives used in poker AI will be described in the next chapter.

This is a large class of algorithms that consist of repeatedly simulating a game and then learning from that simulation. A particularly attractive feature of self-play training is that it ignores that the problem is a game and uses reinforcement learning techniques. Convergence to a mixed equilibrium is tricky issue. This is called "Crawford's problem" after an early paper to identify that convergence is not assured (Crawford 1974). This problem cannot be eliminated by choosing a sufficiently low learning rate as demonstrated in another paper (Singh, Kearns and Mansour 2000). This paper looked at the simplest possible games, those with two players and two actions. They proved that simple gradient ascent in returns with a small (infinitesimal) growth rate will cause the mean returns to the players to converge to those of a Nash equilibrium strategy but that the strategies themselves may not necessarily converge.

Self-play should converge onto the set of rationalisable strategies i.e. those that remain after iterated dominance elimination. The rationalisable set is the set of strategies that will be played in a mixed equilibrium. The reason why self-play is expected to converge onto this set is that if self-play has stopped playing strategies that are outside the rationalisable set there is no reason to start again as no strategy outside the set can do better than any strategy inside.

There is a plethora of refinements of and alternatives to self-play gradient ascent which have better convergence properties. For example, the WoLF (Win or Learn Fast) algorithms store two strategies for each player each of which learn at different rates

(Bowling and Veloso 2002). Another example is the lagging anchor algorithm (Dahl 2002). Dahl has a nice survey of the various approaches in (Dahl 2005).

The most famous example of self-play is TD-gammon (Tesauro 1995). Convergence is not an issue here because this is a game of perfect information so there is a single pure equilibrium strategy. TD-Gammon worked by using an evaluation function generated by a neural net. The inputs to the net were simple functions of the board position. It then selected the action with the highest estimated value. Later versions searched a small number of moves ahead. The evaluation function was trained by TD learning on games generated by self-play.

The success of TD-gammon generated a lot of interest. It rapidly overtook all other gammon bots and was competitive with the best human players despite a deceptively simple approach. This did not automatically lead to a string of similar successes which has led to some speculation as to what properties of backgammon made it suitable for the self-play approach.

2.4.Machine Learning

As the name suggests, machine learning aims to get a computer to improve its performance on a task from experience. It deals with the same problems as statistics and decision theory. In an online learning task the algorithm's performance is evaluated as it learns. In an offline task, it is given certain amount of learning time before it is evaluated. Most of machine learning algorithms are iterative. Whether the process converges, and what solution it converges to, are the two key questions with iterative algorithms. Good iterative algorithms are desirable because the quality of the solution tends to improve steadily if convergence is achieved.

The work in this thesis will employ a connectionist approach.

2.4.1. Connectionism

Connectionism is a broad tradition based on using neural nets but that can be used with any non-linear functional form (the structure) with a number of free parameters. In this tradition, the same steps are taken. Firstly specify a performance measure which is

easy to estimate. Next design a vector of input features. As mentioned in (Michie 1982), humans often find it easier to describe which basic features they use than explain how they combine them. Then select a smooth non-linear computable function which takes the vector of features and outputs a solution to the problem. Repeatedly estimate the gradient of the performance measure with respect to the function parameters and increment the free parameters a small step proportional to the gradient;

$$w_{i+1} = w_i + \lambda \frac{\partial \mathrm{T}_i}{\partial w}(w_i)$$

where w_i is the value of a parameter in the structure and T_i is the value of the performance measure on the i^{th} example

Stopping and the learning rate (λ) are not always automated but this hasn't stopped many successful applications. The algorithm is usually stopped when either the time available for learning has run out or there is no discernible improvement in performance over recent steps. The stability of the method isn't guaranteed; slower learning rates are more stable. (LeCun et al. 1998) provides advice on these implementation details.

For many model fitting tasks stochastic gradient ascent is better than batch gradient ascent. In batch gradient ascent, the weights aren't changed until the total derivative is calculated over the whole sample. In stochastic gradient ascent, an unbiased estimate of the gradient is used. The simplest unbiased estimate of a mean is a single element of that mean, so the derivative calculated on a single case is used. This will add variance to the learning as the change in weights will depend on which case is presented next. This can be controlled by choosing a small learning rate. A small learning rate is often necessary anyway to prevent instability. With stochastic gradient ascent a substantial amount of learning can occur before the algorithm has got through a large sample even once.

Connectionist techniques are sometimes described as knowledge-free but that isn't a helpful description. It is not that the designer does not need knowledge of or insight into the task, but that that knowledge and insight is confined to designing the relevant features, the choice of structure and the learning algorithm. After that he patiently

waits for stochastic gradient ascent to hopefully converge to a good set of parameter values. While the same structure can be used for many tasks, the features are usually task-specific. For this approach to work, the desired output must be a smooth function of the features. Care must be taken so that the desired relationship is a function and that it is as smooth as predictably possible. While a connectionist structure can learn a non-linear function, there is only so much complexity any fixed structure can learn; this complexity should not be wasted if a simple redesign of the features will smooth the desired function. (Swingler 1996) emphasises the importance of how the inputs are encoded.

2.4.2. Reinforcement Learning

Reinforcement learning imagines a situation in which an agent is given some observations and a set of legal actions and tries to act so as optimise some utility. Model fitting can be classed as a form of reinforcement learning because the distribution generated by the model can be classed as an action and the error function can be classed as a reward. So RL is a larger class of problems.

RL is usually expressed as an agent learning to improve its performance from its own experience on the task. But an agent can learn from another agent's experience as long as that agent had a soft policy (every legal action is selected with a non-zero probability) and the probability of taking each action is known. A simple reweighting will convert the sample of one agents experience to that of another. If the two strategies are very different though, this reweighting will increase the variance by effectively making the sample smaller.

The simplest illustrative example of RL is a class of problems called the multi-armed bandit problems. In these problems the agent is repeatedly presented with the same set of actions, after each action he receives a reward from the distribution for that action. The reward is independent given the action and observed after the action. Gittens and Whittle worked out how to optimally solve this problem if the distribution of returns for each action and their priors are drawn from common exponential family distributions (Gittins and Whittle 1989). This makes the problem a POMDP.

The multi-armed bandit problem illustrates the exploration-exploitation trade-off. Each action has two effects; it generates an immediate reward and it allows the actor to be better informed about the reward distribution for that action. In multi-armed bandit problem it is easy to maintain an estimate for the mean reward for each action. The greedy action is the action that has the highest estimated reward. Always selecting the greedy action is not an optimal strategy while it may get the best reward (exploit) the current state of information, it will not generate much new information (explore) because it only selects one action. Most of the convergence results for reinforcement learning depend on the GLIE (greedy in the limit but with infinite exploration) condition; each action is taken infinitely often but in the probability of selecting the greedy action tends to 1 (Singh, et al. 2000).

The simplest strategy is the ε-greedy strategy; keep a record of the mean return for each action, select the action with highest mean reward with probability (1- ε) and select an action using the uniform distribution with probability ε. GLIE can be selected be achieved by ensuring that $\varepsilon \to 0$.

Another simple policy is soft-max (or Boltzmann) selection;

$$P(A = j) = \frac{\exp\left(u_j / \tau\right)}{\sum_j \exp\left(u_j / \tau\right)}$$

where u_j is the current estimate of the value of action j and τ is a temperature parameter that controls how "soft" the selection is.

2.4.3. Reinforcement Learning Techniques for Decision Problems

Most RL techniques are designed to work where aspects of the environment are uncertain or unknown. To compare various techniques, an experimental format has developed. A generative model was used to simulate the actions of the environment and the different RL techniques could operate in the simulated environment. Their performance could then be compared in a standardised setting. This format also proved

a competitive method for solving difficult decision problems. RL algorithms are usually iterative; they offer a decision in any situation and because the simulation can be controlled it is usually possible to ensure that they always improve. Even if the technique is not particularly efficient as an online RL technique (it may take a long sample to converge onto a good strategy and online it would be losing reward all the time), it may be useful in this offline setting if the simulation is rapid enough. This is convenient because many of the positive theoretical guarantees for RL are in the long run limit and based on the GLIE condition. The form of RL used in this thesis is a policy-gradient method.

Policy Gradient Methods

The simplest policy-gradient algorithm is REINFORCE (Williams 1992). This is limited to episodic tasks. A soft policy function with free parameters is defined. The derivative of the reward with respect by using the formula;

$$\frac{\partial}{\partial \lambda} E(r) = E\left(\sum_d (r_t - b_d) \frac{\partial \pi_{id}}{\partial \lambda} \frac{1}{\pi_{id}} \right)$$

(where r_t is the reward on episode t; λ is a parameter of the policy function; π_{id} is the probability of selecting action i as the d^{th} decision of an episode; b_d is the baseline chosen for the d^{th} decision). This is then setup for stochastic gradient ascent because the quantity is an unbiased but noisy estimate of the derivative. Note that equation holds for any choice of baseline b as long as it is independent of the action chosen.

There are two limitations to this method. Firstly it works only for episodic tasks. It has to wait until the end of the episode before doing any learning. Secondly the updates can show high variance especially if a poor baseline is selected. Surprisingly, the expected reward from a node is not the best (least variance) baseline (Dayan 1991).

2.5.Outline

This chapter concludes with a brief summary of where the techniques outlined previously will be used later. This thesis aims to use a connectionist approach to poker. This will involve using a simple reinforcement learning algorithm (REINFORCE) to

improve the decision making of the structure. Hidden variables will be used to represent how an observer (who can't see anyone's private cards) would predict the play of a deal of poker. Self-play training will be used to develop a poker bot without requiring a sample of opponent play.

Exploitation will be attempted by looking for a population best response using a sample from that population. The sample will be used by fitting an opponent model using maximum likelihood learning. In poker, usually only hands of players involved in a showdown are ever revealed. This means that missing data methods are required. Here, the unrevealed hands will be imputed using rejection sampling.

3. Poker AI

There was only a smattering of interest in poker AI until the Computer Poker Research Group at the University of Alberta started publishing (Billings et al. 1998). In mature areas of research, there is often a dominant idea that has proved itself and work concentrates on refinements of the dominant idea or expanding the set of problems it can be applied to. This is not the case in poker. In fact, a wide variety of artificial intelligence techniques have tried on the various challenges presented by poker usually in a variety of combinations.

Before discussing the development of poker AI, poker games in general and the rules of limit hold'em in particular (as the game used in this thesis) are outlined.

3.1. Poker

Poker is a family of similar card games. (These are sometimes placed in the larger class of games called vying games.) They are multi-player games which use a pack of cards and some counters (or chips) which may be in the possession of the players known as their stacks or available to the winner of the game (deal or hand) known as the pot. The play in each deal consists of a number of betting rounds. At the start of each betting round, cards are revealed either to a single player or to all players. Each player acts in turn and may take three types of actions. Folding which means conceding any interest in the pot. Calling means matching the largest amount put into the pot by any other player up to this point in the deal. If a player does not have to put any chips into the pot, calling is known as checking. Folding when a player could check is frowned upon but not always banned. Raising involves putting more chips into the pot than the maximum so far. A betting round ends when each player has put the same amount into the pot or folded and each player has acted once. The first betting round always starts with some forced bets where some (or all) of the players must put chips into the pot before looking at their cards. If only one player hasn’t folded, the pot is awarded to that player otherwise the play continues to the next stage. If at the end of the final betting round there is still more than one player involved, then their cards are revealed and ranked according to the rules with the best hand awarded

the pot. (In some versions, two rankings are used and the best hand on each ranking is awarded the pot.)

Within this framework, there is a multitude of variations. Some common features should be noted. In general, there is little or no card play. In “draw” poker, a player chooses which cards to swap between rounds but this is not considered strategically difficult. In most other variations, the three betting options outlined above are the only choices made. This means that poker should be simpler than other card games. The result of any single game is strongly influenced by luck. This is evidenced by a simple observation. A simple strategy can be at least as likely to win chips in a single deal as any expert player; that strategy is to always call. This does not mean that game is all luck as the expert player is more likely to win many chips and lose few. One-off games of poker are rare; usually a sequence of deals is played.

3.1.1. Rules of Limit hold'em

The game studied in this thesis is multi-player limit hold’em. Limit hold’em can be played by between 2 and 10 players. Players act in a strict rotation. One player is identified as the dealer. Before any players look at their cards there are two forced bets called the small and big blind. The player to act after the dealer puts one unit into the pot; this is called the small blind. The next player puts two units into the pot and this called the big blind.

Each player then receives two cards drawn without replacement from a standard 52 card deck. A round of betting then starts with the next player after the big blind. All raises must be by two units. This is called the pre-flop round. The blinds are known as live bets because those players will still be allowed to bet once even if all the other players call. After the betting round has finished, three cards are drawn from the deck and revealed to all the players; these are known as the flop. A betting round follows starting with the first player to act after the dealer. A single card is then revealed to all players; this is known as the turn. Another betting round ensues again starting with the first player to act after the dealer but raises must now be by four units. The fifth and final card is revealed to all players; this is called the river. The five cards revealed to all the players are known as the board. The final betting round ensues again starting

with the first player to act after the dealer and raises are again of four units. If more than two players remain at the end of the river stage then a showdown occurs. The player who can make the best five card poker hand from the two private cards he was dealt at the start of the game and all five board cards is awarded the pot. The ranking of the hands is given in Table 1.

	Description of category	Ranking within category
Straight Flush	All 5 cards of the same suit and of consecutive ranks	Top card in the sequence
Quads	4 cards of the same rank and 1 other (kicker)	By cards of the same rank and then by rank of the kicker
Full House	3 cards of one rank (trips) and two cards of another rank (pair)	By the rank of the trips first and then be the rank of the pair
Flush	All 5 cards of the same suit	By the highest card among the five that differs
Straight	5 cards of consecutive ranks	Top card in the sequence
Two pair	2 cards of the same rank with another two cards of the same rank and another (kicker)	By the highest pair and then by the second pair and then by the kicker
Pair	2 cards of the same rank and three unmatched cards	By the rank of the pair and the by the highest kicker that differs
No pair	5 unmatched cards	By the highest card that differs

Table 1: Ranking of hands in hold'em poker.

Note that, because at least three of the cards in any player's best 5 card hand must be from the shared cards on the board, the strength of a holding in any category depends on which categories the board cards make possible. Also note that, unlike the other high categories, the straight or flush categories are all or nothing. This adds strategic complexity as a player may be one card short of these categories and must bet before knowing whether he will have a good or poor hand. This is called *drawing* or *being on a draw*.

Variations of hold'em have been a particular focus for research since the publication of a paper entitled "The Challenge of Poker "(Billings, et al. 2002). While there were

publications before that, this paper sparked much of the current interest in poker. Poker has many of the features that make games attractive as an area of research. The rules are quite simple but they generate a strategically complex game. While on first impressions it might not seem to be that different from other gambling games played in a casino, there are important differences. This is not to say that the human urge to gamble is not a major source of its popularity. Firstly the rewards of any player depend on the actions of other players who have choices. In most other casino gambling games, money is won or lost from the house that does not have any choices during the game; either it has no decisions or it must follow a published strategy. More significantly, a player's private cards are central to the strategy of the game. In fact, if everyone revealed their cards the game would trivially become an exercise in calculating probabilities. Other card games retain some strategic complexity even if they were played with all cards face-up. Playing bridge with all cards face-up is known as a double-dummy game. This is not trivial as mentioned in this paper (Ginsberg 1999) amongst others.

3.2.Games studied

The first games studied are the U[0,1] games which are usually 2-player zero sum. In these games, each player receives a number drawn from a uniform distribution. There is usually one round of betting and the player with the highest drawn number wins a showdown. Usually the betting round is quite constrained so that only a small number of betting strategies are possible. These games are attractive because the equilibrium solutions are simple to express and to find. Solutions are expressed by associated each distinct betting strategy for a player with an interval. The main difficulty with finding the solution is to identify the relative positions of the cut-offs between strategy intervals. Once the relative order of cut-offs is fixed, the mean reward is a quadratic function of the various cut-offs. Finding the equilibrium cut-offs involves solving a system of linear equations in the cut-offs. If the resulting cut-offs don't respect the order and hence don't describe a legitimate strategy (playing some sequences a negative probability) then the order was wrong and must be changed. A U[0,1] poker game appeared in the founding work on Game Theory (Von Neumann and Morgenstern 1944). They have also been discussed in papers by Tom Fergusson (Ferguson, Ferguson and Gawargy 2007) and the book "The Mathematics of Poker"

(Chen and Ankenman 2006). These simple games demonstrate many of the counter-intuitive results that characterise equilibrium play in poker games in the simplest possible form. For example, equilibrium play usually involves raising with the weakest holdings possible when a check is possible. This is an example of bluffing.

The next class of "toy problem" games used to generate insight are those that use cards but are simpler than any of the versions of poker actually played. Three such games appear in the literature, in order of complexity. Kuhn poker is played with a three card deck. Each player gets one card and a single betting round is played with one bet allowed. Leduc Hold'em is played with a six card deck, consisting of 2 suits and three ranks. Each of two players is dealt a single card. There is a betting round with a single chip ante and a maximum of two, 2 chip raises allowed. A shared board card is revealed and there is second betting round with a bets of 4 chips allowed. If there is a showdown, a player that pairs the board card wins otherwise the highest card wins.

Rhode Island hold'em is played with a full 52 card deck. Again the game starts with each player receiving a single card. There follows three betting rounds with 3 raises allowed. Between each round, a shared board card is revealed. At a showdown the winner is the player with the best three-card poker hand. (Three card poker is a rarely played variant.) The bets in the first round are half those in the later rounds but twice the antes. The point of the last two games in particular is that they maintain many of the features of hold'em but can solutions can be expressed in a more compact format. While work on these toy-games can generate insights into solving the popular variants, this does not always occur. With toy-games, it is not necessary to make the kind of compromises that a more complex game forces. But learning which compromises are safest for a particular class of problems is often one of the most interesting and useful questions to answer.

Most of the work on poker centres on the game of Texas Hold'em in its many variants. In all variants, the revelation of the cards follows the same pattern as does the rules for deciding who wins the showdown. Limit variants are simpler as the number options for any decision is at most three; fold, call and raise a fixed amount. 2-player (heads-up) is the simplest. Some variations of the rules cap the number of raises allowed in each betting round. In the AAAI competitions, this cap is set at four. With a cap of

four, the number of betting sequences on a standard betting round is 17; 8 of these end with a fold (Billings, et al. 2003). A simple calculation results in a total of 13,122 betting sequences. Abou Risk (2009) estimates the number of betting sequences is increases by a factor of 1,500 for 3-player hold'em. Usually more than 3-players are involved in matches between humans, 6-player and 10-player matches are popular but any number between 4 and 10 is common. Increasing the number of players causes an exponential explosion in the number of betting sequences. This has a dramatic effect on the range of techniques that are feasible. Any technique that involves enumerating the betting sequences becomes impractical.

No-limit hold'em has also been studied. In this game, a player may raise any amount as long as the total amount he puts into the pot during a deal is less than some limit known as his stack. Here two variants have received most study. Push or fold games limit a player's option to betting his whole stack (going all-in) or folding. It is assumed that this is nearly optimal when player's stacks are a small multiple of the forced bets. This mimics the situation near the end of a tournament and hence is of interest to people who participate in tournaments. The solution for the rules used by a poker site is given in (Miltersen and Sørensen 2007). Here they also show that the loss from restricting your strategy to push or fold instead of considering all legal decisions is small as assumed. Similar work for the end of a tournament with 3 players is given in (Ganzfried and Sandholm 2008).

The game of no-limit hold'em is much more complex when the player's stacks are more than a small multiple of the forced bets. The version used in the AAAI competitions is called Doyle's game where the stacks are reset at the start of each hand. In the 2010 competition, the stacks were 200 big blinds. This causes an explosion in the number of available betting sequences.

3.2.1. Public Arenas for testing

There have been two main arenas for the public testing of bots. The first was the IRC poker channel. This was free service that allowed humans and bots to play each other for play money. A difficulty with using play money games against humans for research is that human players tend to be unmotivated when playing. This was not as much of a

problem on the IRC channel. Partly, this was because there was a hierarchy of games; to play in the top games you had to maintain a bankroll above a certain level. Mainly though this was because it attracted some serious players who used it as an arena to test strategies online before there was any real money alternatives. Unfortunately for poker researchers, the IRC poker channel ceased in 2001 and no other internet poker room has replaced it. The University of Alberta computer poker research group ran a server for a period but it was not as popular as IRC. While internet poker has exploded in popularity, this is of little use to poker AI researchers. Commercial internet poker rooms ban bots. Whilst the presence and performance of bots on commercial sites is source of constant speculation, reliable data is difficult to acquire data is unlikely to be published because bots are banned. Also because there is so much real money to be won by competent human players online, it would be hard to persuade them to play for long against a research bot for little or no financial reward.

The other major arena for research bots is the annual computer poker competition hosted at the AAAI conferences since 2006. This attracts the best academic poker AI researchers and independent programmers that are not attached to any academic institution. Competitions are scored in two ways. Firstly there is the total bankroll scoring. All entered contestants are paired with each other in a round-robin format. The bot with the best average is declared the winner. The second scoring is iterated run-off. This starts with the same round-robin as the total bankroll competition but continues in a series of stages. At each stage, the competitor with the lowest mean reward against all opponents is eliminated until there is only one bot left or it impossible to generate significant differences. The iterated run-off scoring is meant to identify which bot approximates equilibrium best. In a symmetric pure adversarial game, an equilibrium bot should never score less than zero so it could not come last at any stage of the runoff. The total bankroll scoring is meant to encourage work on exploitative and particularly adaptive bots. At the moment this scoring rule presents a very difficult statistical decision problem. A population best response would win the competition but this is a very difficult population to model. The population is the entries to next year's competition. While the public logs of last year's competition provide plenty of information about last year's entries, predicting future entries is a very difficult task. There is a small sample of very complex sample points; the teams of bot designers.

The competition results tend to show two patterns. Either all entries make a reasonable fist of approximating the equilibrium and the results are highly correlated with that of the iterated run-off, or there is an entry that is far from equilibrium and the contest is decided by who can win most against this weakest bot. Further information can be found at the competition website **http://www.computerpokercompetition.org/** .

3.2.2. Poker Bots

A bot is a program that can play one of the full versions of poker. The literature on Poker AI has been dominated by the Computer Poker Research Group from the University of Alberta. Their first series of bots are called Pokibot (early versions were called Lokibot).

Loki/Poki

The Loki/Poki bots are series of expert-system style player with population best-response as the aim. Pre-flop decisions are based on a simple expert-designed strategy. A simple offline simulation is used to score each of the starting hands on a single dimension. The first time the bot has to make a decision pre-flop, it uses some simple position features and the score of the holehand to determine the strategy. Post-flop the strategy is more sophisticated. An explicit estimate of the holecards of all players given the public information is maintained. Lokibot would then calculate a pair of values using its hand and the stored distribution of opponent hands. The first is called the *Hand Strength*; this is the probability it has the better 5-card hand (if no more cards were revealed) than all of its opponents. The second is the *Hand Potential*: the probability that the next board card to be revealed will give it a winning hand if it does not already have one. The calculations are feasible for hold'em for a single opponent because there are at most 4 unseen cards; the two cards in the opponents hand and at most two board cards to be revealed. A reasonable shortcut is used to approximate these values if there is more than 1 opponent; the probability of winning overall is assumed to be the product of winning against each opponent. The strategy for a betting round is an expert defined function of these values calculated at the first decision. In the original Loki (Papp 1998), decision making was deterministic but later versions included some randomisation. To update the holehand distribution, it is assumed that opponents used a similar decision-making process to Loki but with some free

parameters. When there was little data for an opponent, an expert designed variation on Loki was used as a default model. In early versions, a simple model is used. This model predict the probability of an action by dividing the betting tree into 12 contexts and keeping a count of an opponent's actions in each. It doesn't use a player's holecards even if they are revealed. A formula is then used to adjust the parameters in the default model so that they would produce the same probability of action. A more sophisticated model is used in Davidson's M.Sc. thesis (2002). The next uses a neural net with 17 inputs and 3 hidden nodes to predict the action. These include 13 simple inputs that describe the current betting and the last action by the player. One input is whether Poki is still active in the hand. (Pokibot developed a reputation for itself in the IRC poker room and opponents were adapting to it specifically.) It also includes the 2 heuristics described above. The final two inputs are the whether the default model predict a call or a raise. These last four inputs depend on knowing a players' holecards. When the holecards weren't revealed, a sample hand was drawn from the distribution maintained. Rollout improvements are also added to each version. These add sophisticated strategies that aren't explicitly in the base expert system. For example in situations where the base system was recommending betting now and at the next decision point, a simulation could show that calling now and raising later can generate better returns. This is known as a check-raise.

The last paper giving a description of the Pokibot systems is (Billings, et al. 2002). A more detailed description with an emphasis on the opponent modelling aspects can be found in Davidson's M.Sc. thesis (2002).

3.2.3. Two & Three Player Hold'em

This is one area of poker research where there is a common theme; game abstraction. Game abstraction is a simple idea. A game that is too large to solve is approximated by a smaller game. The smaller game is solved to find an equilibrium. The strategy in the smaller (abstract) game then has to be translated back into the full game. It forms the backbone of many of the most competitive poker bots entered in competition which will be described later. So far this simple idea does not seem to have spread beyond poker to other applications.

Recent work shows that this process can generate some counter-intuitive behaviour; it is possible that using a larger abstract game may result in a worse approximation to equilibrium. (Waugh et al. 2009) This has not prevented game abstraction on larger and larger abstract games being used to create some of the most successful bots.

To apply these methods to poker, most work relies on the fact that while the number of betting sequences is large, it is not infeasible to enumerate them. The number of distinct information sets is still an unfeasibly large number because of the number of distinct combinations of cards visible to a player. This is counteracted by clustering hands and using the clusters instead of the hands to generate a strategy. This is called *card abstraction* in the terminology. The resulting abstracted game still has a huge number of information states but they can be solved by sophisticated algorithms. Research varies on whether there is a little approximation of the betting sequences or none, how the cards are clustered and which algorithm is used to solve the resulting abstracted game.

The first bots to embody this approach is PsiOpti (Billings, et al. 2003). At each stage, cards were grouped into six clusters or buckets. Imperfect recall clusters are used with transition probabilities being calculated. (Imperfect recall means that buckets at later stages are not a strict partition of the buckets at previous stages.) Five of the six buckets on each stage were based on the Expected Hand Strength statistics

$$\mathrm{E}(HS) = \mathrm{E}_b(\mathrm{E}_o(W))$$

where E_b is the expected value over board cards, E_o is the expected value over opponent cards (all pairs equally likely) and W indicates whether the player wins, loses or draws the showdown. The last was grouped those hands that had a high Hand Potential statistic as used in Pokibot. Later versions dispensed with Hand Potential statistic but added an Expected Hand Strength Squared statistic

$$\mathrm{E}(HS)^2 = \mathrm{E}_b\left((\mathrm{E}_o(W))^2\right).$$

The thinking is that this value will distinguish between draws and mediocre hands that have the same Expected Hand Strength statistic. A certain amount of expert opinion is then used to create the buckets from these statistics.

The game solution technique used was linear programming with CPLEX numerical software. It was only considered feasible to solve three stages of the game together. The first three stages could be solved. For any of the seven pre-flop betting sequences that continue, there is a distribution over buckets. Each sequence can then be thought of as a separate game with a different initial distribution over buckets. A further reduction in game size was achieved by limiting betting to three bets per round in the abstraction instead of the four in the full game they were trying to solve. It was not expected that this would generate a large difference as sequences of four bets on a stage are rarely seen when competent players play heads-up. After these approximations/abstractions, it was possible to solve the game.

A similar approach was taken by researchers at Carnegie Mellon University. Here though they automated the generation of the card buckets using the Game-Shrink algorithm. The exact algorithm can reduce the size of the game without changing the equilibrium. This was demonstrated on the game of Rhode Island Hold'em. GameShrink could reduce this game sufficiently that it can be solved exactly but that isn't the case for heads-up limit hold'em so some approximations have to be made.

A variety of card clustering techniques are used to reduce the size of the game. A difference with the Alberta work is that there is an emphasis on using automated clustering algorithms instead of using an expert to help decide on the cluster boundaries or the values they are created from. The most sophisticated version of this process involves clustering river hands on their probability of winning against a random hand. On the turn hands are then clustered based on the distribution over river clusters. Earlier versions use the same metrics as the Alberta work. Again the early versions couldn't solve all four stages of the abstracted game together. The difference here is that while the abstracted game curtailed to the first rounds was solved and the solution stored, play on the turn and river was produced by solving the associated LP at decision time. The abstractions for the late rounds were calculated offline but the solution was computed online, using the early round model to generate the starting

distribution over clusters by enumerating over possible opponent hands. (Gilpin and Sandholm 2006)

Since then both teams have concentrated on using more sophisticated equilibrium finding algorithms which obviates the need for the compromises of grafting the solution of an early rounds game to one for the later rounds. This involved using the excessive gap technique for the team from CMU (Hoda, et al. 2010). The team from Alberta has used a different technique to the same end. This is called Counter-Factual Regret minimisation (Zinkevich, et al. 2008). CFRM is guaranteed to converge in the case of a perfect recall 2-player zero-sum game. Perfect recall in this case forces the buckets at each stage to be refinements of the buckets at the previous stages. A Monte-Carlo version of this algorithm has been developed (Lanctot, et al. 2009). This gives quite a bit of flexibility over how much of the tree is enumerated and how much is sampled. The advantage of sampling is the same as that for stochastic gradient ascent; improvements start quicker because each cycle is shorter and this compensates for the variance in each cycle.

3-player limit hold'em has been attacked in the same fashion. This is a larger game so more compromises are required to reduce the abstracted game to a size that is suitable for the equilibrium finding algorithms. Risk (2009)solves for a 3-player abstracted game using fewer buckets than the 2-player versions. To this solution, he then adds Heads-Up Experts. When three near equilibrium strategies are playing each other approximately 65% of deals are heads-up before the flop. Less than 0.2% of those don't involve the six most common sequences. For these sequences a strategy with more card buckets is grafted onto the base solution. In this way, the more abstract (less buckets) solution is required when all three players are involved is only used when all players take the flop (Risk and Szafron 2010).

No limit poker is more challenging for this approach. In most situations, a player has the option of raising by a large range of amounts as well as calling or folding. As such it will not be possible to enumerate all betting sequences. The abstraction approach is still used but in the abstracted game, the betting is limited to a small number of multiples of the pot. This is in line with the advice of poker literature especially for beginning players. This reduces the number of betting sequences considerably. An

added complication is that the actions of opponents that aren't limiting their betting in this way have to be translated to actions in the abstract game so that the abstracting player can interpret the actions. This can mean a bot that took a lot of effort to create can be exposed by a simple opponent that exploits the weaknesses in the way it translates bets. This exploitation usually involves betting just either side of the cut-off in raises used by the abstraction. This is explained in detail in a M.Sc. thesis by Schnizlein (2009). In that thesis he also describes a number of tricks to minimise the effect of these weaknesses.

3.2.4. Exploitation in Heads-up Hold'em

The Vexbot (Billings, et al. 2004)exploitative bot for heads-up hold'em creates a detailed model of the opponents play and then uses that to calculate the best response. A later version was called BRPlayer (Schauenberg 2006). Again the relatively small number of betting sequences in heads-up limit poker is key. While playing an opponent, a count of the opponent's reaction (fold, call or raise) is kept for every betting sequence. When there is a showdown, the strength of the opponents hand is recorded. This is done by calculating its strength against a random hand and assigning it to a category as a result. With this information it is possible to calculate the expected value of any betting option and cards by enumerating over all betting sequences. On the river there are no board cards to come. On the turn it is possible to enumerate over the one board card to come. On the flop all turn cards are enumerated but a sample of six out of the 46 possible river cards is used. For the pre-flop, it is not possible to enumerate over the board cards to come so the system reverts to the card abstraction used in the equilibrium approximation bots. Expected values are calculated for the abstract game with card buckets and transition matrices instead of the full games with private cards and board cards.

A difficulty with this approach is that it does not naturally generalise between betting sequences. Some generalisation is added by using intuitive similarity measure to group sequences. If there is few examples of a particular sequence, data from the most similar sequences are used. Some default data is also added to the model to help this system to play the first few deals before the opponent modelling kicks in. This is of the order of a few hundred deals. These types of bots are very useful in that they can find

weaknesses in the equilibrium approximations. Remember in a two-player zero-sum equilibrium no player should be able to win chips from an equilibrium bot. Johanson (2007) describes another method for calculated the exploitability of bots, he calls Frequentist Best Response. This calculates an offline best response strategy i.e. in a way that could not be used during a match. It can though produce a better best response and hence give a more accurate estimate of the exploitability.

Best response bots based on a model can be 'brittle': they can do very well against the bot they are trained against but can do poorly against other bots. This issue has been address in Johanson's M.Sc. thesis (2007). There a restricted Nash response is defined. In this a game is defined where one player (the exploiter) can choose any strategy while the opponent must play a fixed strategy with probability p and any strategy the rest of the time. An equilibrium is then sought in this restricted game. The fixed part of the opponent's strategy is usually a model of the opponent. The exploiter's strategy will then be a trade-off between winning as much as possible against the fixed part of the strategy without exposing too many weaknesses that the equilibrium seeking part of the opponent strategy can find. What the team at Alberta have found is that with a small value of p, the exploiter gives up a little of its performance against the model but reduces its exploitability dramatically. A variation on the theme is the data biased response (Johanson and Bowling 2009). Instead of fixing a probability p for the whole strategy, different restriction probabilities are used for different betting sequences. A variety of ways of doing this are discussed in the paper but the general idea is that for betting sequences where there are many observations, use those observations and where there is few assume the opponent is playing to maximise his expected return.

A disadvantage of the BRPlayer/Vexbot style bots is that it can be a long way into a match before they have enough data to make good use of their detailed models. An approach that can show adaption to the current opponents much quicker is one that selects between a set of prepared strategies. This can be framed as an 'experts problem' and any algorithm for such a problem can be applied. The question then becomes how to select the experts. Johanson has experimented with using Frequentist Best Responses and Restricted Nash Responses to other bots in the Alberta stable (Johanson 2007).

3.2.5. Toy Game Exploitation

If the game chosen is simple enough, more sophisticated opponent modelling techniques become feasible. Kuhn poker is used in a number of studies (Hoehn 2006) (Bard 2008). This is a particularly simple game as the undominated strategies can be modelled by five parameters. Because of the simplicity a simple model for the change in strategy between stages is feasible. Bard uses a particle filter. Baye's Bluff (Southey et al. 2005) discusses a simple Bayesian approach to opponent modelling assuming the opponents play a static strategy. It is demonstrated on Leduc hold'em. Leduc hold'em is a small enough game that an explicit distribution of the probability of selecting each action at each information set is feasible. While work on toy-games is of interest in itself, it is often difficult to extend the work to larger problems.

3.2.6. No Limit Exploitation

A different line of research has developed recently. The game studied is multi-player no-limit hold'em played in commercial internet rooms. A database of play in these rooms is used to fit a specific opponent model to the observations. The first paper in this line discusses how to build an opponent model only (Ponsen et al. 2008).This specific model consists of four parts. Each part is trained by using decision-tree construction software. Two parts form a general model of the population. One part predicts the action of players given the public information (board cards, betting situation and history); the other part predicts the strength of hand at a showdown given the public information. The strength of hand is defined by using buckets similar to those used by the heads-up abstraction bots. The rule on showdowns is that players reveal hands in rotation and that a player can concede his interest in the pot without revealing his hand. This doesn't cause a difficulty for their model because they assume this only occurs when a player cannot beat the previously shown hand. This is a natural assumption. The specific part of the model then constructs separate models for each distinct opponent. Decision trees are a constructive learning method; this means that the tree becomes more complex as more data becomes available. Intermediate variables allow these trees to deal intelligently with the betting history.

In this scenario, this means that the general models can be quite detailed while the specific model may grow slowly as evidence of the differences in play of a specific player from the general population becomes available. Useful as an opponent model is, it needs an exploiter to take actions using the model. Monte-Carlo tree search is used to generate the actions. When it is the exploiters turn to act, it can use the opponent model to simulate the rest of the hand (the action part to generate decisions and the hand strength part to predict who will win a showdown if it gets that far). A Monte-Carlo tree search algorithm can use these simulations to select an action (Van den Broeck, Driessens and Ramon 2009). Some testing has also been used with this style in limit play against bots as well (Ponsen, Gerritsen and Chaslot 2010)

Van der Kleij (2010) constructs a bot using similar techniques. The game studied is heads-up no-limit hold'em but the approach is sufficiently flexible to adapt to other variants. The difference here is that instead of a general and a specific model, he describes using a hidden Markov model to group opponents into types and learns models for each type. He calls this k-models clustering but in essence it is a hidden variable model like the ones that will be used section 4.3.3.

3.2.7. Others

An interesting phenomenon in recent AAAI competitions is the presence of imitation bots. These were trained to imitate bots that were successful in previous competitions. An example of this is the SARTRE bot (Rubin and Watson 2010) that used case based reasoning to imitate the play of the top players from previous years. While there is nothing published about the entries MANZANA and PULPO, it is reported in internet chat rooms (http://pokerai.org/) that they were created by fitting a neural net to previous successful entries. In both cases the worst-case performance wasn't much worse than the later version of the equilibrium bots they were trained from. In 2009 MANZANA toughest opponents were Alberta's Hyperborean-Eqm and GGValuta to which it's score in both matches was -0.007 sb/h (small bets per hand). Hyperborean's own worst performance was -0.012 sb/h against GGValuta. This would suggest it is nearer an equilibrium than the bot it imitates. This is not infeasible. If the neural net is smoothing between hard categories in the original, that smoothing may improve the performance. For SARTRE, its worst case performance is -0.051 sb/h against

Hyperborean: so its performance is worse, but not by much despite using a quite different structure.

Another consistent source of bots for the AAAI competition is the Technical University in Darmstadt. They have included designing a bot for the competition as a component in one of their courses. They have made all their bots available (Knowledge Engineering Group, TU Darmstadt 2011). A description of the bot that finished second in the 2008 AAAI competition is given in (Schweizer et al. 2009). This bot combined many ideas including a low-dimension opponent model and rollout simulation to improve a basic strategy. This rollout simulation was specifically tuned to the competition rules.

A recent and extensive review of work in poker is available (Rubin and Watson 2011).

3.2.8. Summary

It is useful to summarise where the main strands of poker-AI currently lie and what challenges each faces. The expert system approach behind Pokibot has been abandoned. It can be difficult to develop expert systems beyond a certain point as it becomes tedious for the expert to explain his thinking beyond a certain level of complexity. Later versions added the use of simulations and a neural-net based statistical opponent model. Techniques that rely on enumerating the betting sequences have had great success but they are naturally limited to games where that is feasible. Some approximation of the game tree was used in the early versions of Psi-Opti (Billings, et al. 2003) but this was a little ad-hoc and there is no obvious way of extending these techniques to games with larger game trees. While the Monte-Carlo techniques are interesting, they present their own difficulties. The seven second per decision time limit used in the AAAI competition does allow the generation of the kind of large samples required by Monte–Carlo methods. But using so long per decision makes any kind of fine-tuning experiments very difficult in a high variance game like poker. It also makes it difficult to assess their performance without considerable computational resources. The imitation bots (like SARTRE (Rubin and Watson 2010)) are an interesting new line of research but they are only a partial solution. They need a good player to imitate.

4. The Structure

4.1. Introduction

The first decision to make in designing a poker playing system is which version of poker to focus on. Two and three player limit hold'em have received a lot of attention but they are special cases and techniques that are successful there may not extend to other games. Instead we focus on limit hold'em with more than three players which is a version that was an initial focus of work by the Alberta Computer Poker Research Group. Focus in the poker AI community has shifted to other versions of poker but this is not because all the interesting questions related to multi-player hold'em have been solved. Rather it is because the team behind the most successful multi-player bot (Pokibot) felt they had taken it is far as they could.

Up to the 2008 AAAI competition, all the work was done assuming that the number of opponents the bot would face would vary. The number of players varies naturally as players join and leave a game. The distribution of player numbers in a sample of hands from the IRC database was used. The 2008 AAAI competition included a six-player limit hold'em competition and after that it was decided to concentrate on six-player. Note though that all of the techniques could work with a distribution of player numbers just as easily as a fixed number.

4.2.Overall Plan

The plan is to use two bots to find a population best-response. The first is an *opponent model* bot which mimics the play of opponents observed in the population. The second is an *exploiter* bot which learns to optimise its return against the opponent population. It is intended that both should use the same basic connectionist structure with the opponent model being fitted by maximum likelihood and the exploiter being trained by policy-gradient reinforcement learning on simulated hands generated with the opponent model.

To be suitable for this plan, the basic structure must possess certain characteristics. It should employ soft selection; every legal action should be chosen with non-zero probability. This is for two reasons; firstly, so that when fitting the model, no action

can have a zero-probability which would prevent the use of the logarithm function; secondly so that every branch has a chance of being explored by the reinforcement algorithm.

This structure should be parameterised by a finite (but possibly large) number of parameters. The output of the structure (probability of an action) should be a differentiable function of those parameters. Those derivatives should be in a closed form so that they can be calculated quickly.

It also should act quickly as the reinforcement learning algorithms envisaged uses simulated deals. Simulation only works if enough sample points can be taken to average out the noise. This means that time-expensive components should be avoided wherever possible. Time-expensive components would include any simulation within the strategy, any deep-tree search, or full enumeration of any variable with more than a few values.

So far the desirable properties outlined are not particularly limiting. For example most neural net architectures would satisfy these properties. The final desirable property is much more demanding. The strategy must be capable of solid play. This is the other important consideration when choosing input features. If the structure cannot even approach a Nash equilibrium, it cannot model decent players.

So the task in this chapter is to design a structure and select a learning algorithm that results in a bot that is capable of approximating an equilibrium within the constraints outlined above. Of course approximating equilibrium is an interesting question in its own right.

4.3.Outline of structure

The first element to be designed in a connectionist structure is the vector of input features. The relevant information that the bot uses to make its decisions is grouped into three categories.

- Bet Features: These describe the betting situation after the next action.
- Card Features: These describe the cards that can be observed by the bot and how they interact; both those in his private hand and the shared cards on the board.
- Hidden Card-type Beliefs for opponents: Instead of trying to convert the action history of opponents directly into features, a hidden variable model is used. The distribution over the hidden variable (called card-type) for each player given the board cards and the previous actions of that particular opponent. This is usually called a belief vector in work in partially observable problems.

4.3.1. Bet Features

The bet features consist of ten values that are simple to calculate. Examples include the size of the pot, the number of players still involved, the number of players to act after the player on this round or the next and whether this action will finish the betting in the stage. The direct cost of the action (i.e. how many chips have to be put into the pot) is a special case which will be explained in the details of structure section. Two sets of features are calculated; those if the bot calls and those if it raises. The state of the betting after a fold is of no interest to the player that folded as the rest of the deal cannot affect their result.

4.3.2. Card Features

Designing the card features is difficult because of the rules of the game relating to the cards are complex. The ranking of hands is a complex function of the ranks and suits of the cards involved. The order in which board cards were revealed is also important as in interaction with betting decisions this can reveal useful information about an opponent's hand.

A detailed set of card features was designed. This was guided by the details mentioned by expert poker players in their published tactical guides e.g. (Brunson, Baldwin and Goldberg 1979) . The detailed but concise description of hands in the quiz questions of "Middle Limit Hold'em" was particularly useful in this regard (Ciaffone and Brier 2002). There are over 300 distinct features. The purpose in using so many features is to give the rest of the structure a fair chance. A potentially successful structure can be hampered by insufficient detail in its inputs. This is illustrated by an attempt to use machine learning on the game of gin-rummy using features (Kotnik and Kalita 2003). No gin-rummy specific knowledge was used in the design of the input features and its standard of play was below that of beginners.

The features themselves are extremely simple; each requiring at most three lines of code. Most are binary, the rest consist of simple counts or the rank of a card. For any set of visible cards, there are at most ten nonzero features and often less. As a consequence, the features are very quick to calculate and take a small fraction of the overall time.

Using card features based on tactical guides adds an extra justification if the structure were to be used as a model of human play. It is reasonable to expect that the strategy of human players is a function of those features as the strategy guides give an indication of what features human players look for. Also many players will have read these guides and hence will be calculating these features before deciding on their actions.

The features can be divided into two categories, those that refer only to the public cards that can be seen by all the players and those that represent the player's own private cards and how they relate to the board cards. On the later stages (turn and river), the board features from the previous stages are retained but the private card features from previous stages are discarded at the next stages. Remembering in which order the board cards were revealed may be relevant in deducing an opponent's private cards but remembering which potential hands a player could have made is irrelevant. For example, knowing that a flush draw was possible on the flop is relevant even if the draw didn't come because it helps explain a player's actions and hence deduce his hand. But remembering that you had the flush draw is irrelevant in the same situation

because it can't affect the result of a showdown and it can't affect your opponents' play because they won't see you had a flush draw until after all the actions are taken.

Some example features are described here and the complete list included in the appendix. These are the features relevant to flushes and potential flushes when the turn consists of three cards of the same suit. The features in brackets are those retained from a previous stage (the flop). The indented features are those that are only relevant when the feature above is on. There are two ways three cards of the same suit can be on the board by the flop; either they all appeared on the flop or one appeared on the turn. Each has a distinct feature. A player can then have a flush, need another card of that suit to appear on the river to make a flush using four board cards and one from their hand or they may have no cards of that suit. No cards in the possible flush suit is indicated by the absence of any of these features, while the other cases are indicated by the Flush and 1-card Flush draw features. If either of the flush or flush draw features are on the number of possible higher flushes or flush draws possible is calculated; remember if two hands are in the same category they are separated by the highest card.

(Flop all same suit)
- **Turn card of same suit**
- Flush *-one card in hand of flush suit*
  - Higher ranks not on board in flush suit
  -

[(Flop all same suit)
- **Turn card of different suit**

or
(Flop 2 same suit)
- **Turn card matches two of same suit]**
- Flush
  - Higher ranks not on board in flush suit
- 1 holecard flush draw
  - Higher ranks not on board in flush suit

Table 4-1: Some example card features

4.3.3. Observer model

It is well-known that history (how a position is reached) is important in imperfect information games because it help deduce an opponent's state. We know from the theory of partially observable Markov decision processes (POMDP) that a belief state

vector can replace the history without affecting the quality of the decision making ; where the belief state vector is the probability of each hidden state at the time of taking a decision. In poker, the hidden part of the current state is the other players' holehands. It is not infeasible to enumerate all possible holehands in hold'em as there are at most $^{52}C_2$ (1,326) possible. But it is far from ideal, as it would slow down the simulation. Instead we approximate the distribution of hidden holehands by a distribution over a small number of card-types in an observer model. At any point in the game, the observer model has a distribution over card-types for each player based on the decisions made by that player so far and the visible board cards. Before each player makes his decision, the observer model makes the card-type distribution available for each opponent.

For all experiments in this thesis, we use 8 card-types at all stages. Early experimentation indicated that this was a reasonable compromise between speed and accuracy in the observer model. It was decided to maintain this value throughout the work in order to make it comparisons easier.

The point of the observer model is that the players don't have to use the betting history directly (which is awkward to encode into simple features as it is of variable length) or a full enumeration of the opponents' hand distribution (which slows the simulation); instead they use a short vector (the probabilities of each card-type) for each opponent. The observer model acts as a shared service to the players, that converts the public history into a convenient form.

With any model it is important that it approximates the most relevant aspects of the true situation. In poker, the result of any hand is completely determined by the sequence of betting decisions and the result of the showdown. So this is the information our observer model tries to fit. It does that by maximising the likelihood of the observed actions but not of the cards.

As outlined in Table 4-2, for each step during a deal, there is a corresponding step in the observer model. The difference is that the steps in the observer model do not depend on the hidden cards. Both make use of an action selection function which is described in the details section.

	Simulation	Observer
Start	Draw holecards for all players	Initial distribution of pre-flop card-types
At betting decisions	Use the action selection function with the acting player holehand features, the bet features and the card-type beliefs calculated by the observer model as inputs to calculate the probability of each action. Then draw an action.	Use the same action selection function except remove the holehand features from the inputs and include the card-type beliefs. Then update the card type beliefs based on the action selected in the simulation
When new board cards revealed	Calculate the holehand features for the new stage for each player	Calculate the new card-type distribution for each player using the transition matrices
Showdown	Compare hands and award the pot	Calculate final card types using the showdown estimator

Table 4-2: The components of the observer model

In the simulation model, only the action selection function needs to be learnt. The other elements in the table are completely specified by the rules or by the design of the features. Some freely available code is used to calculate which hand won at a showdown. The action selection function used by the simulation is learnt by reinforcement comparison (also known as the REINFORCE algorithm).

The observer model is learnt by maximum likelihood with a refinement to avoid predicting board cards which have no relevance to the outcome of the game. The observer model does not adjust the parameters it shares with the simulation model in the action selection function. Only the parameters that are directly connected to the

card-types are learnt in the observer model. For each stage of the game a completely different set of parameters is used. There is no attempt to generalise across stages.

4.4.Details of Structure

4.4.1. Action selection function

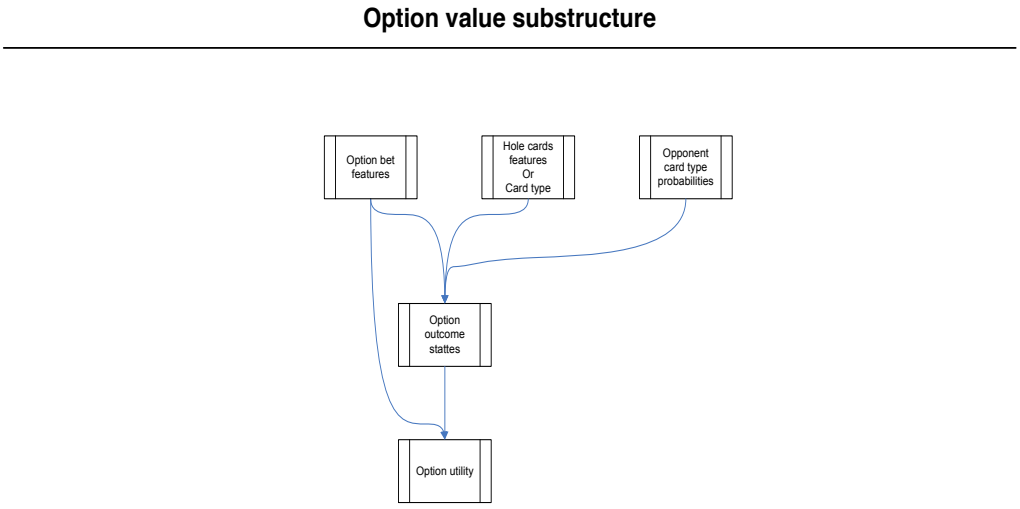


Figure 4-1: A subsection of the Action Selection Function

The design of the structure starts with the elements that relate to a single option (call or raise). This sub-structure can be thought of as a class-mixture regression model. All of the inputs are used to generate a probability of belonging to each class (called an outcome state) by multinomial logistic regression. For each class a predicted return to the active player is calculated by a linear regression using the bet features subtracting the direct cost so that options can be compared. This appears to be a sensible way of valuing a position.

This class-mixture regression model has strong similarities to the most popular form of neural net: the three layer perceptron with sigmoidal squashing function. The inputs correspond to the input layer. The single output is the estimated value of the position. The first difference is that the hidden nodes in a three-layer perceptron can be

considered as a set of independent binary classes while this structure uses a single multinomial variable. This is not usually how the hidden layer nodes are described but any continuous variable on a fixed range can be linearly mapped to a probability between 0 and 1. The second difference is that the bet features are used a second time in the structure when they are multiplied by the parameters associated with each state. In a three-layer perceptron, the inputs are divorced from the outputs by the hidden nodes.

While many structures (including the three layer perceptron) can in theory express any function if made large enough, in practice it can make a huge difference if the structure matches the task. The class-mixture regression was chosen as it seemed more suited to the problem of calculating an expected value from the inputs. As an example consider calculating the expected value of a call on the river that leads to a showdown: this is the product of the size of the pot and the probability of winning the showdown. The mixture-class regression can express this quite compactly, using only two classes whose relative probability is an estimate of the probability of winning (the input features include card features which express the strength of the player's hand and a belief vector that represents the probable strength of the opponents' hands) and whose value is the size of the pot (which is a bet feature). A three-layer perceptron would have to use more parameters to estimate this type of product. We would expect calculations near the end of the game to be dominated by this value as any extra betting is likely to be small relative to the size of the pot (in a limit game).

While this segment of the structure is motivated as a way to value a betting option it is never used for this task. Instead it forms a component of the larger Action Selection Function structure. A value is calculated for each of the legal actions. Call or raise uses the class-mixture regression. Fold is always given the value of zero. After a fold a player cannot lose any more chips than he has already put into the pot or win any chips. Finally to get the probabilities of each legal action a soft-max function is applied to each value.

It may seem odd to carefully design a structure that might be ideal for valuing actions and then not use it for that purpose. But there are an almost infinite number of ways of

putting together a connectionist structure. Some intuition is often required to suggest a system for more thorough testing.

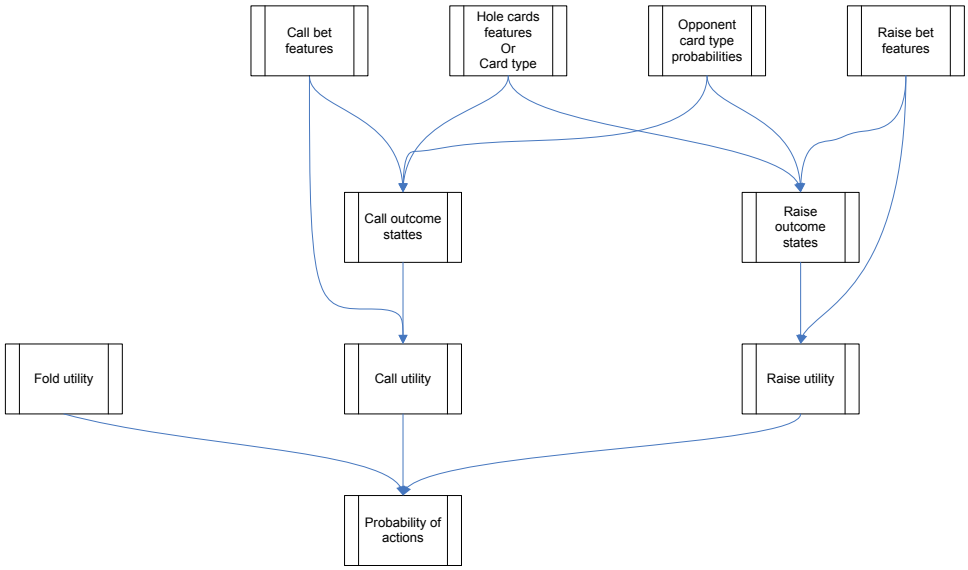


Figure 3-1: Action Selection Function diagram

The detail of the structure can best be described by a series of equations. At the top level, the input features lead into a layer that calculates the probability of each outcome states;

$$P(o_s|j) = \frac{\exp(m_{sj})}{\sum_j \exp(m_{sj})}$$

Equation 4-1

where the m_{sj} are a linear function of the bet features (b_{fj}), the card features (or the card types when used by the observer model) and the card-type distributions reported by the observer model for the opponents. The coefficients of the linear function are free parameters.

The value of each outcome state is represented by a linear function of the bet features.

$$\left(\sum_{f} w_{sf} b_{fj} \right)$$

Equation 4-2

Then the value of taking option j is calculated for each of the options where c_j is the direct cost of the action (the number of chips that must be put into the pot).

$$u_j = \sum_{s} P(o_s | j) \times \left(\sum_{f} w_{sf} b_{fj} \right) - w_c c_j$$

Equation 4-3

The w_{sf} and w_c are free parameters.

Finally the probability of an action is then calculated by a soft-max function on the values.

$$P(a = j) = \frac{\exp(u_j)}{\sum_{j} \exp(u_j)}$$

Equation 4-4

A limitation of this structure is that when there is more than one active opponent who have already made a decision, potentially useful information is hidden by summing the opponent card-type distributions. In self-play, this shouldn't be a major consideration because deals tend to become heads-up quickly when good players play limit hold'em. This point will be discussed further in the section on population best response where a solution will be proposed.

For all experiments in this thesis, we use 8 outstates at all stages. Just as with the card-types, early experimentation indicated that this was a reasonable compromise between

speed and performance. Selecting the size of components in a connectionist structure is an interesting research question on its own but it is not tackled here.

4.4.2. Transition matrices

The card-type belief vector at the end of a stage represents a convenient summary of the information contained in the decisions taken by each player. But intelligent players should not ignore the prior play of opponents at the next stage. The interaction between play at previous stages and the board cards revealed provide useful evidence as to the likely characteristics of the private cards held by opponents. When the board-cards are revealed, a transition matrix is calculated and this is used to calculate the card-type belief vector of each player at the next stage from the card-type belief vector at the previous stage.

Most tactical guides recommend using both the board cards and an opponent's previous actions to predict their current private cards. For example consider the situation where a players previous actions indicate it is likely he has a flush draw (needs one card to complete a flush). It would be expected that his future actions will be strongly dependent on whether the next card completes the flush draw.

The entries in a simple transition matrix would represent

$$P(C_i|C_{i-1}),$$

where C_i is the card type at the current stage and C_{i-1} is the card type at the next stage.

Learning such a transition matrix can be done by a version of the Expectation-Maximisation (EM) algorithm called the forward-backward (or Baum-Welch) algorithm. This is a common practice in spoken language processing and textbooks in that area give a concise description . Transition matrices have been used before in the work on approximating maximin equilibrium using abstraction but they are calculated by sampling (for the pre-flop stage) and enumeration for the later stages. The transitions matrices here are learnt as a function of the card features derived from the public board cards.

Including the public board cards, the elements of the transition matrix represent

$P(F_i \cap C_i | C_{i-1})$ where F_i is the board cards revealed up to this stage.

These entries are estimated as the exponential of a linear function of the board card features up to and including this stage.

Up to this point we have used maximum likelihood estimation in a straightforward manner because we are interested in predicting all of the actions. The board cards add a complication because the observer model is not directly interested in them. The observer is only interested in the board cards because modelling the interaction between board cards and actions could improve the models predictions of both actions and showdowns. This is achieved by including board cards in the overall likelihood but dividing out their direct effect. Dividing out unwanted elements of a total likelihood function like this is sometimes known as conditional maximum likelihood.

An overall likelihood of the actions so far can always be calculated up to any stage of the game. As each new piece of information is revealed, the likelihood is multiplied by the probability of that action given the previous information. To illustrate the method the calculations involved for a player that makes decisions in the first two stages of the game are shown.

Likelihood of all the actions for a player during the pre-flop stage,

$$\sum_{C_0} P(C_0) P(A_0 | C_0) = L_1$$

Likelihood of all the actions for a player during the flop stage followed by the flop

$$\sum_{C_0} P(C_0) P(A_0 | C_0) \times \left(\sum_{C_1} P(F_1 \cap C_1 | C_0) \right) = L_2$$

Likelihood of all the actions for a player during the flop stage followed by the flop and then followed by all the actions on the flop

$$\sum_{C_0} P(C_0)P(A_0|C_0)\times\left(\sum_{C_1} P(F_1 \cap C_1|C_0)P(C_1)P(A_1|C_1)\right) = L_3$$

L_2/L_1 is the likelihood of seeing a particular flop F_1 given the actions A_0 of a player pre-flop. So the likelihood of seeing the actions both pre-flop A_0 and on the flop A_1 is the overall likelihood after the flop divided by the likelihood of the flop cards F_1. i.e. $L_3/(L_2/L_1)$. The log of the likelihood of the actions is the target value to be optimised. $T = \ln(L_3) - \ln(L_2) + \ln(L_1)$

If the player also made actions on the turn and the river, the calculations are extended recursively. Every time a board card is revealed the likelihood of that card is divided out. The formulas are quite involved but can be rewritten as an adjustment to the backward factors in the standard forward-backward algorithm. Considering the equations above we can define the forward factors as usual as

$$fwd(C_0) = P(C_0)P(A_0|C_0)$$

Then we can define the backward factors as

$$bwd(C_0) = \frac{\partial T}{\partial fwd(C_0)} = \frac{\sum_{C_1} P(F_1 \cap C_1|C_0)P(C_1)P(A_1|C_1)}{L_3} - \frac{\sum_{C_1} P(F_1 \cap C_1|C_0)}{L_2} + \frac{1}{L_1}$$

Equation 4-5

From this the partial derivatives associated with each the initial distribution and the action probabilities can be calculated. (Every action probability appears in a forward equation somewhere.)

For the partial derivatives associated with the transition matrix itself we have

$$\frac{\partial T}{\partial P\left(F_1 \cap C_1 | C_0\right)} = fwd\left(C_0\right)\left(\frac{P\left(A_1 | C_1\right)}{L_3} - \frac{1}{L_2}\right)$$

Equation 4-6

The method of estimation is strictly gradient ascent but the calculations themselves are very similar to those for the forward backward algorithm except for the two formulas above.

4.4.3. Showdown estimator

The showdown estimator allows the observer model to estimate the probability of each player winning the showdown given the card-type belief vectors before the cards are revealed. These parameters don't directly affect any player's strategy because there are no decisions made between the last action and when the board cards are revealed. But the model should be encouraged to predict the showdown from the card-types as an intelligent player will be trying to predict the strength of his opponents' cards from their previous actions. Remember all information from the previous actions of players is channelled through card-type beliefs.

In this model the probability that player *i* wins at a showdown is

$$P\left(W = w | C(0) \ldots C(n)\right) = \frac{q_{C(w)}}{\sum_{j=1}^{n} q_{C(j)}}$$

Equation 4-7

where *W* is the winner of the showdown, *n* is the number of players in at the showdown, *C(j)* is the card type of player *j* on the river and the *q* parameters are strictly positive free parameters.

We can then define an overall likelihood (of all the actions and the result of the showdown) as

$$\sum_{C(0)\ldots C(n)} P\left(W = w \middle| C(0)\ldots C(n)\right) \times \prod_{j} P\left(A_0, \ldots, A_3 \cap C(j)\right)$$

Equation 4-8

This can be combined with the conditional likelihood calculations for the actions to get an overall likelihood. To learn the q values, gradient ascent is used treating $\ln(q_i)$ as the parameter so that there is no danger of a negative value which would give an invalid probability.

In other circumstances, this could be a very slow procedure as it runs through every combination of card-types for all the players in at the showdown. Luckily in poker in general showdowns with more than 3 players are rare with reasonable strategies. When starting self-play from random initial weights, strategies that never fold can be a phase that the learning system goes through. If the showdown parameters were updated for every showdown, it would cause a wasteful slowdown for the overall system. A work-around is possible by performing the calculation on a random sample of showdowns with many players and weighting the results to compensate. The length of time to enumerate all possibilities of c clusters for n players is c^n. If there are more than three players the program ignores the showdown with probability $\frac{c^3}{c^n}$ but increases the learning rate by a factor of $\frac{c^n}{c^3}$ for the showdown parameters to compensate. This has very little effect once past the initial phase of learning but prevents any excessive slowdown in the early stages of learning. Such practical additions can be important in ensuring an otherwise sound algorithm doesn't get excessively slowed down in the early phase of learning from self-play. Another such practical adjustment is to cap the number of decisions in any one hand at 150, after which the algorithm forces all players to call. This is because early in the learning it is possible for the bot to get trapped in an endless sequence of raises. (Remember there is no cap on the number of raises when only two players are left in hand in most common versions of the rules.) Again, once a reasonable strategy has been learnt long sequences of raises, between two players are extremely rare.

4.5.Pseudo-code

The following pseudo-code summarises the workings during self-play training. The main loop is run for each deal of the training run. Before starting training the parameters in the structure are initialised to small random values.

Main loop:

```
For each deal
    Initialise betting state to start of the hand
    Initialise card-type belief vector for each player to
    initial pre-flop distribution
    Draw cards for each player
    While deal not ended
        If start of new stage
            Calculate card features for all players
            Calculate transition matrix (results are
            represented by P(F_i ∩ C_i | C_{i-1}) in the formulas)
            Update card-type belief vector for the next
            stage using the transition matrix
        Call Action Selection Function to generate
        action probabilities
        Draw action
        Call Card-type Belief Update
        Update betting state
    Calculate net winnings for each player
    For each decision during deal
        For each parameter used with private cards
            Calculate derivative of Action Selection
            Function with respect to each parameter
            Update parameters using reinforcement
            formula
    Call HMM learning routine
```

The card-type belief update function is used to perform a Bayesian update after each action.

Card-type Belief Update

```
Inputs: prior card-type beliefs, action taken
For each card-type
    Call Action Selection Function
    Multiply the prior card-type belief by the
    probability of the action given the card-type
Normalise the beliefs (divide by the total so that they
sum to one)
Return: The updated card-type beliefs
```

The action selection function generates the probability of each legal action using the available public information and either the actor's private card features or a card-type.

Action Selection Function

Inputs: betting state, (vector) sum of opponents' card-type beliefs,
    Either
    (list) shared card features, (list) actor's private card features;
    Or
    A card-type
For both call and raise
    If action legal
        Calculate betting features after action
        Call Action Value Substructure function and store value
If fold is legal assign a value of zero
Calculate action probabilities by soft-max function applied to values for each legal action
Return: action probabilities *(Equation 4-4)*

The next subroutine generates a value for each legal action.

Action Value Substructure

Inputs: action betting features, (vector) sum of opponents' card-type beliefs,
    Either
    (list) shared card features, (list) actor's private card features;
    Or
    Actor's card-type
For each outstate
    Get a total weight (m_{sj} *in* Equation 4-1*)* by summing over all betting features and opponents' card-type beliefs
        Feature value by the associated weight for feature and outstate
    If used for HMM model update
        Add weight associated with actor's card-type and the outstate to total
    Else
        Sum over all card features
            Feature value by the associated weight for feature and outstate
Calculate a probability for each outstate ($P(o_s|j)$) proportional to exponential of each associated total *(* Equation 4-1 *)*
Calculate a value for each outstate by summing over all betting features *(Equation 4-2)*
    Betting feature multiplied by associated weight for betting feature and outstate
Get expected action value by summing over all outstates
    Probability of state by value of state
Add (or subtract) direct cost by associated weight
Return: Action value *(Equation 4-3)*

Implementing the Baum-Welch algorithm for this model is a fairly involved process. The top-level outline of the HMM learning subroutine is listed first and then the various sections are described.

HMM learning subroutine

```
Calculate the all-action probabilities (P(A|Ci) in formulas)
Calculate the forward factors
If deal ended in a showdown
    Skip some showdowns but weight the ones that are worked
    If (more than 3 in at showdown)
        Calculate probability of working the showdown
        If showdown is not worked
            Skip showdown section
        Else
            Adjust forward factors for showdown
Calculate the backward factors
Calculate the adjusted post probabilities
Can now start the learning the weights in the model
Update card-type weights
Update transition matrices and initial distribution
If showdown is worked
    Update showdown weights
```

Calculate the all-action probabilities (P(A|Ci) in formulas)

```
For each player and stage he played and card-type on that
stage
    Calculate the all-actions probability for that player
    and stage and card-type
        By multiplying the probability of each action
        taken during the stage
```

Calculate the forward factors

```
For each player
    For each stage played (starting at the pre-flop) and
    card-type on that stage
        If the stage is first
            Initialise forward factor to 1
        Else
            Initialise forward factor as
            Sum over card-types at previous stage
                Forward factor at previous stage by
                corresponding transition matrix entry
        Multiply the forward factor by the all-action
        probability
```

Adjust forward factors for showdown
For every combination of river card-types for the players in the showdown
    Calculate the joint probability of that combination by multiplying the probability of the observed result *(Equation 4-8)* by the original river forward factors for all the players.
    Store the sum over-all combinations as total likelihood of model
    Store a total for each player and card-type as the new (including showdown) forward factors

Calculate the backward factors
For each player
    For each stage played (starting at the last stage played) and card-type on that stage
    If the stage is last
        Initialise backward factor to 1/(total likelihood of model)
    For every stage prior to the last
        Backward factor is sum over all card-types at the next stage of
            Backward factor at the next stage by the all action probability at the stage after by the corresponding entry in the transition matrix between the stages
Adjust backward factor using Equation 4-5

Calculate the adjusted post probabilities
For each player and stage player
    Adjusted post probability is product of forward factor and backward factor
    A running average is used to scale learning rate so that card-types that are rarely used learn quicker and hence are more likely to come back into use.
    Store a running average for each stage and card-type

Update card-type weights
For each decision and card-type
    Update card-type weights to increase likelihood of taking observed decision with learning rate in proportion to adjusted post probability for that player and stage and card-type over the running average

Update transition matrices and initial distribution
For each player and transition between stages
    For each combination of card-types either side of transition
        Calculate likelihood derivative by Equation 4-6
        Update transition matrix elements by likelihood derivative above divided by product of running averages for both card-types
Update initial pre-flop card-type distribution
    As running average of adjusted post probabilities for each player

```
Update showdown weights
For every combination of river card-types for the players
in the showdown
      For each player
            Calculate the derivative of the joint
            probability with respect to changing the
            showdown weight for one of the players
      Store the sum for each card-type
Update each card-type with a learning rate proportional to
the one over the running average for that card-type
```

4.6.Summary

This basic design and variations will be used for all the subsequent experiments. The key elements of the design are;

- A short list (9) of inputs that describe a state of the betting
- An long list (363) of inputs that describe in detail the visible cards to a player at each stage of the game
- An action selection function that works as if it was valuing each legal action by a class-mixture regression.
- A card-type variable which replaces the private cards in a hidden Markov model of the game. The hidden Markov model also uses by an initial distribution over the card-types at the start of the game, transition matrices calculated form the card inputs and a formula for estimating which card-type win at a showdown. At all points in the game, a belief vector over the card-types is maintained for each player. These belief vectors are then used as extra inputs to the action selection function.

The performance of this structure in self-play will be tested in the next chapter. Its performance when used as an opponent model will be tested in the subsequent chapter.

5. Self-play

The structure was trained by self-play; a deal was played and at the end of the deal machine learning updates were performed on all the free parameters that were used during the deal. Reinforcement learning was used to improve the decision making and maximum likelihood learning was used to improve fit of the hidden Markov model. A difficulty immediately presents itself in that there is not a simple overall measure of performance. The mean return of a set of identical players against each other is zero; poker is a zero sum game. Because reinforcement comparison is a policy-gradient method, there are no value functions to evaluate. Only the decision making can be checked. The next section describes an error measure that is used to assess the performance of the structure and identify on which decisions it struggles.

5.1. Error Measure - Rollout Improvement

To measure the quality of the decisions made by a strategy, we compare the performance of a strategy to the improvement possible with a rollout of that strategy. Rollout is a method for creating an improved policy from any quick policy using simulation. The term rollout was initially used in the paper by Tesauro and Galperin on backgammon (1997). It is described in general in "Neuro-dynamic Programming" (Bertsekas and Tsitsiklis 1996). A true rollout error (which requires an infinite sample size) calculates the average improvement possible from changing a single decision but using the original policy from then on. A small rollout improvement indicates that there is no simple change (i.e. at a single information state) to the original policy that would generate significant improvements. So calculating this error identifies situations where the current strategy is making mistakes.

The idea is to compare the performance of our structure to an algorithm which is known to improve performance but uses more time than we would like.

5.1.1. Procedure

A decision is selected at random from a record of self-play and all the information available to the player before that decision is frozen i.e. own cards, visible board cards

and all decision so far. All unseen cards are drawn. The prior actions are used to weight the results for that card distribution by the probability of the opponents making the decisions seen with those cards. Then the rest of the hand is played out using the player once for each of the legal decisions possible. Using the same cards for both actions (raise and call), means this forms a paired sample.

This process is similar to the procedure used in the earlier versions of pokibot (Billings, et al. 1999), except that weighted sampling is used instead of enumerate and sample [see section 2.1.7]. The advantage of weighted sampling is that it takes the approximately the same amount time to run each sample while rejection sampling can take considerably longer on rare sequences. Having a sampling method that takes the same amount of time on each case makes interpretation of the results easier in that they show the advantage of taking the extra time to run the sample.

```
Repeat for decision sample size
     Play out a hand.
     Draw a decision at random
     Repeat for rollout sample size
          Draw other players hands and any unseen board
          cards
          Calculate probability of other players actions
          up to this decision
          For each legal option
               play out rest of hand
               store return achieved
     Calculate mean return (weighted by probability of
     opponent actions up to this decision)
     Identify action with best return
     Calculate error made by player if the sample means
     were the true means
```

Algorithm 5-1

The comparison between the best decision on the rollout sample and the decision made by the structure is not a fair one. The structure makes its decision before seeing the results on which it is tested, while the sample best decision is only picked after seeing the full sample. This will overestimate the mean return when using a rollout. This is for the same reason that the mean error on the training set is always biased towards underestimating the true mean error in a regression or classification problem. As such similar methods of alleviating the problem suggest themselves. The method we use is k-fold cross-validation. This is common when assessing the performance of classifiers

(Kohavi 1995). Ten divisions are used because it has been shown to be the lowest number with reasonable performance on classifiers. K-fold cross-validation is known to have high variance but that does not preclude its use here because the structure is quick enough to simulate a large number of hands and hence average out some of the variance. In any case, assessing the performance of poker playing programs is not a straightforward matter and is an area of active research (Billings and Kan 2006). We prefer to confine ourselves to a simple method which is easier to interpret and accept the resulting variance as the price for working on a stochastic problem.

To explain the process of calculating the rollout error in detail some mathematical notation is needed. For every element in a sample of size *n*, the probability that opponents have played their randomly drawn cards in the manner seen up to the decision being evaluated is p_j and the return achieved after taking decision *d* is v_{jd}. The value of each option is estimated by $v_d = \frac{t_d}{b}$ where $t_d = \sum_j p_j v_{jd}$ and $b = \sum_j p_j$. For the LFO calculations similar calculations are carried out for each *i* of *k* fractions of the sample. So $v_{id} = \frac{t_{id}}{b_i}$ where $t_{id} = \sum_{j=\frac{in}{k}}^{j=\frac{(i+1)n}{k}} p_j v_{jd}$ and $b_d = \sum_{j=\frac{in}{k}}^{j=\frac{(i+1)n}{k}} p_j v_{jd}$. Also used in the calculations is an estimate of the value of each decision removing one of the fractions from the sample; $v_{-id} = \frac{t_d - t_{id}}{b - b_i}$. For each fraction of the sample the decision that would have the best return on the remainder of the sample is identified; $d^*_{-i} = \arg\max_d (v_{-id})$. The value of following a rollout strategy is then identified by assuming each fraction of the sample would occur in proportion to the p_j with a decision selected by which action was best on the remainder of the sample; $\tilde{v} = \sum_i \frac{b_i}{b} v_{id^*_{-i}}$. An estimate of the return generated by the system is $\hat{v} = \sum_d \pi_d v_d$ where π_d are the action probabilities used by the system. The final value used in the system is that of the action with highest expected return over the whole system; $v^* = \max(v_d)$. The values tabulated are then averages of $(v^* - \hat{v})$ which is called the sample error, $(v^* - \tilde{v})$ which is called the leave-fraction-out (LFO) error and $(\tilde{v} - \hat{v})$ which is called the rollout improvement (RI)

error. The aim is for the RI error to estimate the gain that could be made from using a rollout sample. If there is no gain from the rollout sample either the system is making nearly perfect decisions or it would take a much larger sample to identify where those mistakes are.

5.2. Summary: Results of a training run of self-play

The results here refer to the system which was entered in the 2008 AAAI competition. The first thing to determine is whether any substantial learning is occurring. The quality of the decisions made by the system should improve as more hands are simulated. During a long run, the program was stopped every 10M (million) hands and the rollout improvement was calculated on a sample of 1000 decisions.

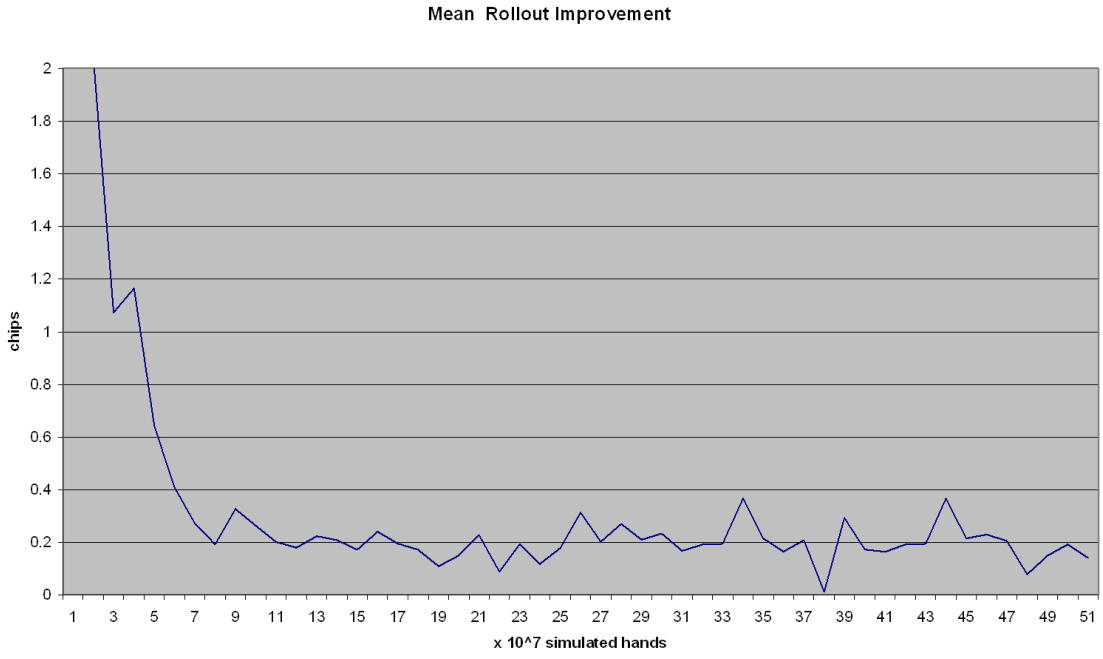


Figure 5-1: Learning curve for 2008 entry

Figure 5-1: Learning curve for 2008 entryFigure 5-1is typical of a learning curve. At the start there is a steady improvement, after which the performance bounces around a mean. This variation about a mean could be due to a number of sources; some of it could because this noisy error measure, some could be due to the natural over-shooting that occurs in a gradient ascent algorithm with a fixed learning rate and some could be because it could be because it is circling an equilibrium rather than converging to it.

Note that the large number of simulated hands shown is feasible because the system makes decisions so quickly. It can play out (without any learning) 1M hands in just under six minutes on a 2GHz processor.

To further investigate the performance of the system, a larger sample of 10000 decisions was used on the final weight set.

	Pre-flop	Flop	Turn	River	Overall
% of Decisions	52%	20%	16%	12%	100%
Sample error	0.219	0.573	0.814	1.037	0.484
LFO error	0.301	0.379	0.537	0.225	0.344
RI error	-0.082	0.194	0.277	0.813	0.140
Std. Dev. of RI error	0.008	0.030	0.089	0.058	0.021

Table 5-1: Rollout errors for the system entered in the 2008 AAAI competition.

The first row of Table 5-1 shows the percentage of decisions taken at each stage of the game. The majority of the decisions are at the pre-flop stage where the cards provide little information and players are trying to eke out small margins. The standard deviation of the RI errors is included. To interpret the data, a few numbers are relevant. The numbers are all in terms of chips where 2 chips is the allowed bet on the first two stages and 4 chips is the allowed bet on last two. The number of players dealt-in varied between 2 and 10 and followed the distribution in the IRC database which has an average of 5.2 players in each hand. The average number of decisions taken in each hand by a player is 2.1, so overall there are on average 10.9 decisions per hand. This reflects the fact that the bot folds the first chance it gets most of the time as is standard with most decent players in a ring game. The standard deviation of returns for the bot in this self-play scenario is 6.54 chips. Note that all the errors listed are less than this figure, indicating that it would take a number of hands for these errors to become noticeable.

The pre-flop play of the system is actually better that its rollout improvement. This is because the structure is learnt by averaging over millions of hands and this can do better than a much smaller rollout sample which is prone to sampling error. The advantage the rollout has is that its sample is specific to the decision being considered.

Later in the game there is more detailed information that the rollout can focus on (longer betting history and more board card already revealed) and less random variation to average out (less board cards to be revealed and less decisions to be randomly selected). The rollout generates a statistically significant improvement on the post-flop stages but not a very large one except on the river.

The LFO error indicates that the returns in poker have high variance and that this causes considerable difficulty when making decisions. If the decision based on a sample size of 900 was always still the best decision on seeing another 100 weighted sample points, the LFO error would be zero. Instead we see that a rollout of sample size of 900 still leaves considerable uncertainty in the decision making. Increasing the sample size would always improve the situation but it is subject to diminishing returns to scale.

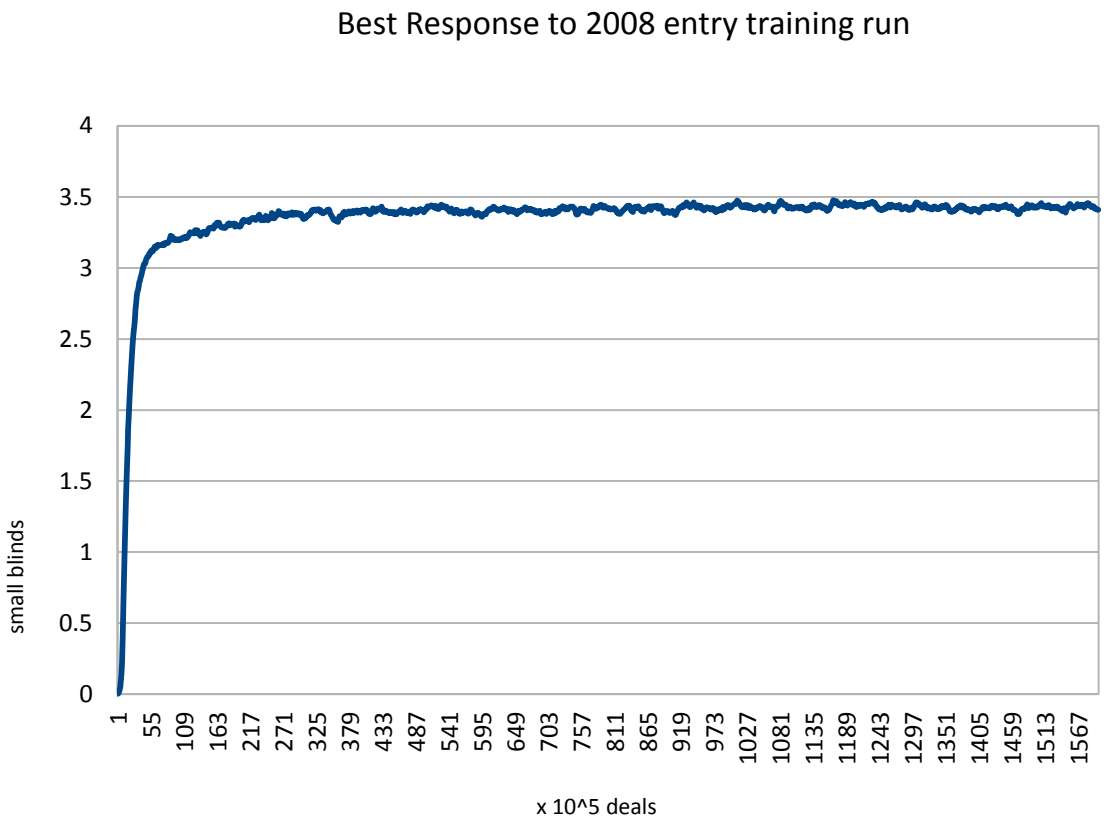
5.3. Best Response Evaluation

While the rollout error is a useful device for helping the relative performance of the bot in different situations; it does not give an overall performance measure. A good measure of how well a bot approximates equilibrium is to train a best response or "*exploiter"*. This foreshadows the work in the next chapter on best responses.

This is done here by playing out deals with the exploiter in one seat and the bot to be evaluated in all the others. Only the exploiter is allowed to learn. The difference in return between the exploiter and that of the bot being tested is an underestimate of how far that bot is from equilibrium; if the bot represents an equilibrium, it should be impossible to find a bot that does better than it does against itself. It is an underestimate because the structure limits the strategies the exploiter can use; it might be possible to win even more against an approximate equilibrium bot from a larger set of strategies.

Performing this test on the bot we entered in the 2008 AAAI competition yields some interesting results. The exploiter learns to win at a considerable rate: 3.43 small blinds per deal. This is a large sum by the standards of limit hold'em. But it requires a highly aggressive strategy to achieve this return with a standard deviation of 23.7 small blinds where a standard deviation of 6-8 small blinds is traditional when good players play

limit hold'em. A running average was maintained during the training run of 160 million hands and is reproduced in Figure 5-2 .



Inspecting a sample of deals from the final strategy suggests why that might be so. The exploiter learns to take advantage of a weakness in the strategy entered in the 2008 competition; it appears to fold too many hands to repeated raises.

To get a better feel for the strategy of the best response and the 2008 entry it learns to beat we include some basic statistics. The first two statistics are the mean and standard deviation. The next is VP$IP (voluntarily put chips into pot) which is common in poker discussions. This is the percentage of deals the player puts chips into the pot after the blinds. The next is the Fold%. This is the percentage of deals for this player that end in a fold after voluntarily putting chips in the pot. The next is Ww/oSD. This is the number of deals the player wins without a showdown (because the other players have folded) as a percentage of the deals that voluntarily puts chips into the pot.

Deals where a player doesn't fold and doesn't win must result in a showdown. The next statistic is the W$SD. This is the percentage of those showdowns won out of

showdowns contested. There is a slight difference from the common definition of this statistic as split pots are counted as a fraction of a win rather than being counted the same as if pots were won outright. The final statistic is the AF (Aggression Factor). This is the number of bets or raises divided by the number of calls (but not checks). Note a check occurs when a player calls without having to put any chips in the pot. It is not a distinction made elsewhere in the thesis but it is used here because this is a very common statistic in poker discussions. These statistics will be repeated in the next chapter of the thesis when we deal with best response to a population.

There is a considerable amount of comment on various internet forums that discuss poker strategy about the "correct" value for these statistics (and more detailed statistics). An article (Tanenbaum 2007)in the "Card-player" magazine offers one set of recommended values; 25%-35% for the VP$IP value and 1.4 -2.4 for the AF value while stating that average players will have an AF in the range 0.9-1.4.

	Mean	std.dev	VP$IP	Fold%	Ww/oSD	W$SD	AF
Exploiter of 2008 entry	3.44	23.75	99.6%	2.8%	49.2%	24.2%	28.94
2008 entry against exploiter	-0.69	11.27	33.8%	69.2%	1.8%	74.1%	0.57
2008 entry in self-play	0	8.28	39.9%	33.3%	18.8%	47.4%	1.14

Table 5-2: Statistics for best response to 2008 entry and the 2008 entry

The extremely high aggression (a value above 4 is considered high) and low rate of wins at a showdown indicate that this is a strategy that bluffs extensively. In fact the aggression factor indicates that it rarely calls. The high VP$IP and low number of later folds indicate that this is a strategy that basically raises constantly. Such a player is often described as a "maniac". Like many best response this strategy is "brittle". It is very strong as a counter to the strategy it was designed to play against but would be very weak against other strategies e.g. a strategy that called slightly more often than the 2008 entry would probably win at a brisk rate as it would catch a lot of the bluffs.

5.3.1. 2008 AAAI Competition

The bot generated by this self-play procedure was entered in the 2008 AAAI computer poker competition in the six-player limit hold'em category. There were exactly six entries that played in all matches. Details of the competition are available here http://webdocs.cs.ualberta.ca/~pokert/2008/results/index.html.

Position	Name	Average small blinds won per hand
1.	Poki0	0.646
2.	AKI-RealBot	0.573
3.	DCU	0.252
4.	CMURing-Prototype	0.163
5.	mcBotUltra	-0.135
6.	GUS	-1.500

Table 5-3 : Results of 2008 AAAI 6-player competition.

The team behind AKI-Realbot published a technical report describing their entry (Schweizer et al. 2009). Included in that is a simple analysis that offered a different view of the results. They generated a cross-table that kept track of the average transfer of chips between players i.e. who each player was winning and losing chips from. This analysis showed that our DCU won chips from most of the other entries except that it lost a small amount to Poki0 (0.052 small blinds per hand). AKI-Realbot finished ahead of DCU's entry because it won more from the last bot GUS. As they say themselves, it is likely that if GUS was replaced by a different bot, a different result is possible. But the aim of AKI-Realbot was to exploit weak bots and it achieved its aim in the 2008 competition.

This is an example of why it is incorrect to say that one bot is better than another in general. Often the relative performance of a strategy depends on the opposition it is facing. The six player limit competition was only run in 2008. Other researchers

moved their efforts to no-limit and 3-player limit hold'em which present different challenges.

5.4. Subsequent competition entries

5.4.1. 2009 AAAI Competition

The six-player limit competition was replaced in 2009 by a 3-player limit competition. A bot was entered in this competition as well as the heads-up (2-player) competition. Each entry was trained in the same manner as the entry for 2008 but using simulated deals with the appropriate number of players.

The heads-up bot finished tenth of eleven competitors in iterated run-off competition. Its worst result was in matches against MANZANA where it lost 0.538 small blinds per hand. MANZANA is an independent entry. Little is known about this entry except for some short discussions on www.pokerai.com . It lost at a rate of more than 0.4 small blinds against eight of the competitors.

The 3-player bot finished sixth of seven competitors. Its worst result was in matches against dpp (Technical University of Darmstadt) and Hyperborean-Eqm (University of Alberta) where it lost 0.536 small blinds per hand. It lost at a rate of more than 0.4 small blinds against five of the fifteen possible pairs of opponents.

The conclusion to be drawn is that the structure is not competitive with the best specialist bots in 2 and 3 player hold'em. This was not a surprise.

5.4.2. Self-aware: 2010 AAAI competition

For the 2010 AAAI competition, a small adjustment was made to structure. The aim of this adjustment was to allow the structure to represent some higher level thinking. An intelligent strategy for an imperfect information game should consider what type of private information its play suggests to its opponents; it should be self-aware. It should consider what cards it looks like it has as well as what cards it has. A crude way of including this is to allow the structure to look forward one step and see what the results from the observer model would be i.e. what the belief vector over card-types that its opponent would be using would be if it took each decision. After this look-ahead own card-type distribution is calculated for the legal decisions, it is added to the inputs

(with the actual holecards and all the other inputs) before the actual decision is taken. The observer model works as before and ignores these extra inputs. This avoids any circular calculations.

It was decided to concentrate efforts on the three player competition where the betting tree is larger than for heads-up and hence those techniques that rely on enumerating the tree are more likely to struggle.

When a rollout error was calculated and compared to that for the previous entry, there didn't appear to be much improvement. The 2009 entry had an average rollout error of 0.33 small blinds per decision, while the structure with the added self-aware element had a rollout error of 0.30 small blinds per decision. The difference is less than the standard deviation of the rollout error calculations. When the self-aware structure was entered in the 2010 competition though, it did considerably better. It finished second out of five competitors in the iterated rollout competition. This is not directly comparable to the previous result as the list of competitors changes each year. Its worst performance was when faced with two versions of Hyperborean (University of Alberta) where it lost 0.120 small blinds per hand. The previous year that figure had been 0.384 small blinds per hand. While there have been no published improvements in the University of Alberta entry over the year, it is unlikely to have been weakened by any changes.

5.5. Summary

The results of the self-play experiments show that this combination of structure and learning algorithm is capable of significant learning even without the additional self-aware inputs. Statistically significant learning may appear a low standard to set but in a high variance task such as poker it can be difficult to generate any statistically significant improvements in performance.

The results of the best response evaluation indicate that the self-play bot is a long way from being an approximate equilibrium bot but the roll-out errors indicate that it is hard to find that optimal response. This means that we can move onto the exploitative play elements of the research with some confidence. Or to put in another way, if there

were no positive results for these experiments there would be no point trying to add exploitation.

The 2008 AAAI competition results show that this structure was amongst the best academic poker bots at the time. The 2009 competition results show that it struggles to compete with the specialist two and three player systems. The 2010 results show how components can be added to the basic system to create more sophisticated systems.

The 2008 entry forms the basis for the agents used in the next chapter.

6. Exploitative play

The challenge to be addressed in this second phase is to generate a population best response; a strategy that does best against a specific population of opponents. Framing the problem in these terms means there is a single performance measure; the mean reward in matches with opponents drawn from the population. It is not possible for any strategy to be a best response to all populations. But good statistical techniques should be able to generate a good (if not the best) response to a population if it is given a reasonably large sample to work with. First a process that should be able to generate such a good response will be described and then an example population will be used to illustrate how well the process works in practice.

The process we use involves four steps;

1. Find (or generate) a sample of play by the population of opponents. Without a sample of play the process has nothing to work with. In poker as in many applications, the sample will usually have hidden data.
2. Fit an opponent model to that sample. Imputation is used to fill in any missing cards. Also fit an observer model. After the two models are fitted the sample is discarded.
3. Learn to optimise returns in simulated deals with that model. The reinforcement learning algorithms described in the previous chapter are reused here.
4. Test the exploitative bot by recording its performance in a sample of matches against opponents from the specified population.

Steps 2 and 3 would not be possible without a basic structure that allows the kind of learning algorithms suitable for this phase of the work.

Of course, this is not the only way to approach this task. The positive theoretical guarantees of maximum likelihood only apply when the model is correct; reality can be represented as setting of the parameters of the model. This is unlikely in any complex setting; a model is more likely to be an approximation with various compromises.

Direct methods of reinforcement learning aim to learn from experience without using a model. At the heart of a lot of these methods is an off-policy estimate of the returns of following any strategy given samples from following a known soft strategy. Effectively this is a weighted sample estimate of the mean. While this approach can be considered more robust because it does not depend on a good model of opponent play, it has some limitations. It can learn from a variety of strategies but only where the probability of generating the experience it is learning from is known. This is required to reweight the experience in the importance sampling formula. Importance sampling can be data inefficient when the strategy being evaluated is very different from the strategy that generated the data.

6.1. Opponent Model Fitting

The self-play experiments suggest that the structure used is capable of a reasonable standard of play. The third place finish in the 2008 AAAI 6-player limit competition also suggests the structure is sufficiently complex to enable it to play reasonably well. This structure is used as a model of opponent play.

Fitting this model consists of two stages; any cards not revealed at the end of the hand are imputed and then the structure is fitted to the decisions in the sample using maximum likelihood. In practice, the mean log-likelihood per decision is optimised. This is the natural log of the probability of the action seen in the dataset given the holecards from the model.

$$\frac{\partial \ln\left(P(A \mid H)\right)}{\partial \lambda} = \frac{1}{P(A \mid H)} \times \frac{\partial P(A \mid H)}{\partial \lambda}$$

(where $P(A|H)$ is the probability of a single decision given a hand and λ is a parameter of the structure.)

This formula is particularly simple to code as it involves only a small change to the code for the REINFORCE algorithm.

The observer opponent model is fitted to the same dataset using exactly the same process as that used in the self-play experiments. As before those parameters unique to

the observer model (the card types and transition matrices) are learnt separately by fitting the observer probabilities. The only difference is that it is learnt from a finite sample rather than an on-going self-play simulation.

6.1.1. Hidden Card Imputation

The opponent model is a strategy; a function that generates the probability of each legal action given all relevant information including the private information. In a record of poker play, there are two types of betting sequences a player can have in a hand; those where the private cards are revealed and those where they aren't. This makes fitting the opponent model a hidden data problem. Where the cards are revealed, the model fitting can proceed directly; the model can generate the probability that the player made those decisions with those cards and the learning algorithm can increase that likelihood. Sequences where the cards aren't revealed form a large majority with typical players. When the hand isn't revealed, imputation can be used. A hand is drawn from the distribution implied by the current modelling and the model fitting procedure is then used as if this imputed hand was revealed. The theory of EM methods tells us that this is a valid way of fitting a model.

Imputation involves sampling an instance of the missing data from the distribution implied by the current model. Most of the imputation will be done using rejection sampling but occasionally enumeration will be used.

To apply rejection sampling, we need a proposal distribution. The simplest is to sample the missing cards uniformly. The probability of accepting a set of cards is then given by the probability the model made those decisions with the proposed cards. An estimate of expected number of draws before an acceptance is useful.

Defining:

j- indexes the observations (the possible betting sequences)

d- the number of sequences

π_j – the true probability of that observation

$\hat{\pi}_j$ - probability of generating that observation with the model i.e. the acceptance rate for rejection sampling

Then the expected number trials with rejection sampling per observation should be

$$\sum_{j=1}^{d} \pi_j \frac{1}{\hat{\pi}_j} \approx d$$

assuming the model is reasonably accurate.

In limit hold'em, there are usually three options at every decision but occasionally two. Therefore $d=3^n$ is an overestimate of the number of distinct sequences of length n. Therefore rejection sampling should not take too long on average to accept an example for reasonably short betting sequences. This is convenient as long sequences that end in a fold are rare; a player who has put a lot of chips in the pot is usually reluctant to fold. But they are not impossible, and these betting sequences cannot safely be ignored when fitting the model. I will return to the problem of long sequences of decisions.

There is two ways rejection sampling could be used in this application. Firstly all the hidden cards in a single deal could be drawn and treated as a single hidden variable. Alternatively rejection sampling could proceed in two stages. In the first stage, rejection sampling could be used to draw a hand for each player that didn't reveal their hand. Secondly the hands drawn could be checked to make sure they don't contain the same cards. If they do the whole sample can be rejected. Of course in practice, this checking for overlapping cards would occur after each set of holecards is drawn, rather than working on a set that is definitely going to be rejected. Early testing indicated that the two stage sampling method is a lot quicker so that is what was used.

Another advantage of the two-stage method is that it allows an alternative method of dealing with the difficult long betting sequences; full enumeration. Because each player has only two cards, there are at most ${}^{52}C_2$ (=1,326) hands possible for a player. As such it is feasible to calculate the probability of a long betting sequence for each possible hand. A hand can then be drawn from this distribution in proportion to the calculated probabilities. So the complete method uses rejection sampling for the short betting sequences, full enumeration for the long sequences and an overlap test to make

sure the same card doesn't appear in two places. The final element is to decide where to switch between the two methods of sampling. Using full enumeration on sequences of more than six decisions didn't take an excessive amount of time over any sequence, so that was used without any further effort to optimise that setting. (If full enumeration was not used, the program could appear to stall as it took hours to accept a pair of private cards on a long betting sequence where it could have analysed millions of deals in the same time.) This is the only place in the thesis that enumeration over all possible pairs of hidden cards is used.

An alternative that is popular in the literature is to use Markov Chain Monte Carlo (MCMC) methods to draw the imputed data. These generate a chain of imputed values which eventually converge to drawings from the desired distribution. This would involve combining three iterative schemes; EM, stochastic gradient ascent and MCMC. Combining iterative schemes is attractive because in the early stages of each an overly exact method in one component is wasteful; the other components aren't in a position to use that precision. A disadvantage of using MCMC in this application is that we are often using relatively large datasets. MCMC methods require the last imputed value to be stored for every missing value. More importantly, in a large dataset each missing value may not be visited that often; as such the MCMC chain for that value may not have converged. As MCMC convergence is a sophisticated issue, it is advisable to avoid the issue wherever possible.

6.2.Exploiter learning

Having fitted the opponent model to a sample of typical play, the next step is to use that model to train an exploiter bot. The exploiter bot uses the same structure as has been used for the self-play experiments and the opponent model. Reinforcement learning is used to learn its parameter values from simulated deals. In the simulated deals, the exploiter sits in one seat and the opponent model plays in the other seats.

Note that there is no reason to learn a separate observer model for the exploiter. The observer model fitted to the sample will predict the card-types for a typical opponent which is exactly what the exploiter wants. On the other hand, unless the exploiter is in

some way flagged as distinct there is no way for the opponents to interpret the exploiter's actions any differently than any other player's actions.

If the exploiter plays in very different style to the members of the opponent population, it is possible that problem of extrapolation will be seen. The model cannot predict how opponents will react to the exploiter because there will be no similar situations in the database.

6.3.Player modelling

The general model predicts the average action distribution for the population of strategies it is fitted to. But the model could improve its predictions if it also modelled some of the variation between the different players in the population. The simplest way of doing this is to add a player type variable to the model. The player-type variable is categorical with a small number of values. At the moment, 8 categories are used. This number was chosen because it worked well with the similar card-types and to force the model to group opponents as there will be more that 8 distinct opponents.

The use of stereotypes in user modelling is a long standing tradition especially in recommender systems (Rich 1979). It has also been applied to poker in a paper (Layton, Vamplew and Turville 2008) that used a similar idea to improve a no-limit bot.

The overall model consists of an observer model and an action model. The player types are added to both. During a hand, the observer model can calculate the probability of the last action given a player's card-type and player-type. Bayesian updating can then be used to maintain a joint distribution over the card and player types for all players. The marginal distribution for both types is maintained by summing across the other type. The sum of the marginal distribution for the player types for all active opponents is added to the inputs for the action selection function beside the sum of opponent card-type beliefs. These form the belief vectors illustrated in Figure 6-1.

At the end of the hand, some players may be forced to reveal their hand in a showdown. If a player reveals his cards, a new player-type distribution is calculated. The player-type distribution for every player is stored at the start of a hand just in case

the hand is revealed. An extra routine calculates the probability of taking the observed actions with the revealed cards for each player type. Combining this with the stored prior player-type distribution results in a new updated player-type distribution. This replaces the player-type distribution calculated by the observer model using the card-types which was used during the deal. It replaces the observer model calculation on the basis that it is the better model which the observer model only approximates.

The joint player-type-card-type distribution at the start of a deal is calculated by multiplying the marginal player-type distribution at the end of the previous hand by the initial card-type distribution. The final piece of the model is the initial player-type distribution which is used at the start of the first deal of a match. Adding the player-types does not add many new parameters to the model. The initial player-type distribution uses the number of player types. The player-types themselves have a free parameter for every combination of game stage, player-type and hidden layer node.

To fit this model, each match is processed twice. The first time is merely to calculate the player-type distribution at the end of the match. A player-type is then drawn from this distribution. This is the imputed player-type and learning occurs as if this type was observed. The initial distribution is learnt as the average of these final distributions. On the second time through the match, the parameters for the player-type drawn learn to fit the recorded actions by maximum likelihood. The observer model is learnt as before but using the imputed types as if they were revealed. This is an implementation of Stochastic Expectation Maximisation.

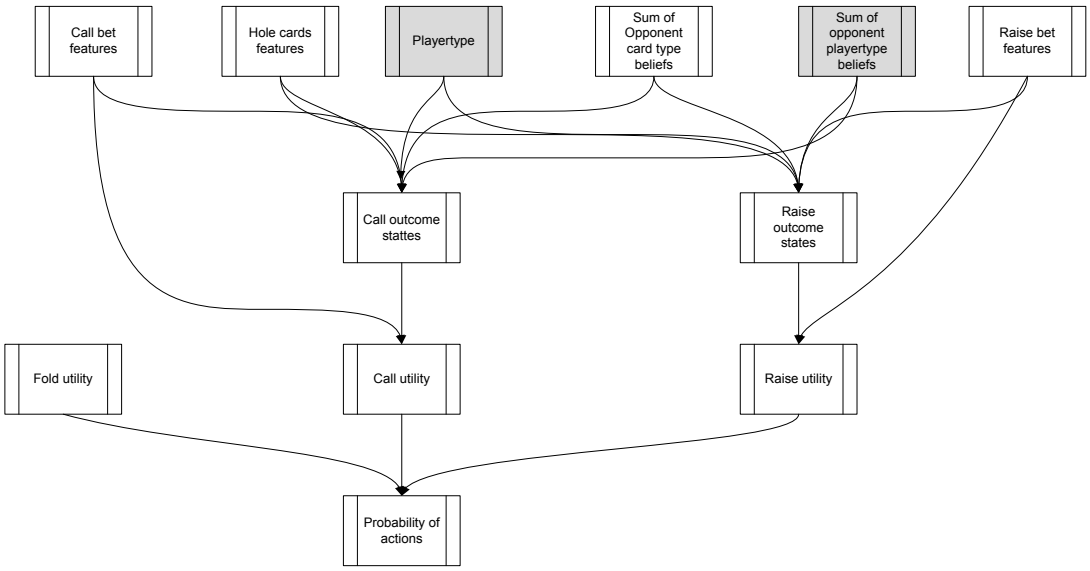


Figure 6-1: Action Selection Function with Player-types.

The new inputs added to the basic structure are shaded. The player-type is a simple categorical variable. The sum of the current belief vectors over player-types is also included. Note that most of the parameters are shared between the different player-types; only those weights coming directly from the player-type in the diagram are different.

The exploiter has exactly the same structure except without a player-type; only a single exploiter is trained.

This system is considerably slower than the original model. An easy speed-up was implemented. To update the joint distribution, a probability of action has to be calculated for every combination of player-type and card-type. These calculations can be speeded up by reducing the calls to the exponential function and replacing them with multiplications. Rather than call the exponential function for each combination of player-type, card-type and outstate node, it is called only once for each outstate node to calculate the effect of the other features. The effect of the card and player types is included by multiplying the exponential of those parameters which is stored. This

should have no effect on the outputs but does speed up the calculations. The aim is to replace calls to the exponential function with multiplications which are quicker.

The exploiter is learns to optimise its return in the same manner as before. Before each match, a player-type is drawn from the initial distribution for each of the exploiter's five opponents. Matches of fixed length are played out and the exploiter learns from each one individually as before. The only difference is that the exploiter has an extra input to learn from; the sum of opponent player-type distributions.

The player-types are similar to the card-types. The card-type distribution conveniently summarises the decisions of an opponent from the start of the hand. The player-type distribution summarises the play of an opponent from the start of the match. This allows the decisions of a bot to depend on the decisions of his opponents in previous hands.

6.4.Exploiter Extra Layer

As was mentioned previously in section 4.4.1, the structure has a limitation. By summing features across multiple opponents, the structure may not be able to distinguish between certain situations. For example the structure would react the same way whether a player whose play indicated he was type A was on the button and one whose play indicated type B was on the big blind or vice versa. To counteract this, an extra layer was added to the structure. For each active opponent, all the features relevant to that player feed into a tile node. Those features consist of the current player-type and card-type distributions and some simple features describing the betting position of that player relative to the player making the decision. The tile node variable is calculated for each opponent and the sum of these tile node variables is used alongside the other features in the main action selection function.

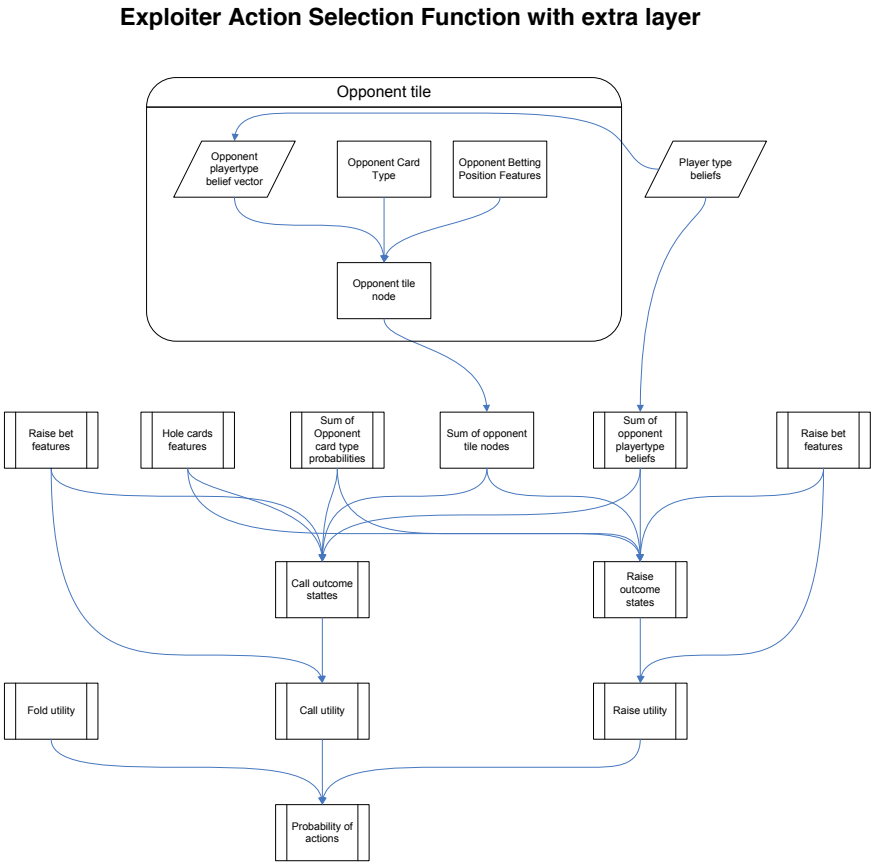


Figure 6-2: Action selection function with extra layer

In this way the various relevant features for an opponent can be connected together rather than being lost in a total. A disadvantage of using an extra layer is that it can be much slower to learn because it can be quite sensitive to the learning rate. This is why we add it in only after the rest of the structure has been learnt.

6.5.Experimental set-up

The experiments use Poker Academy (PA) (BioTools Incorporated 2007) to provide a population of opponents. Poker Academy is a commercial poker playing program that is advertised as a training tool for human players. There is also a facility for adding third party bots. Poker Academy comes with 43 default opponent names. There are 24 pokibot names; these are adaptive bots that explicitly model their opponents hand distribution but then uses a parameterised formula to form an action distribution. The pokibot family of bots are parameterised by 5 continuous controls and 20 discrete

options. There are 15 Jabot names but only 5 discrete settings; these are rule-based unadaptive bots. Finally there are 4 simbots; these are similar to pokibots but as well as putting opponents on two cards they simulate till the end of the hand to decide on the best action. These have 5 discrete settings and 3 continuous controls. Note that other than the Jagbots, the bots are not static. Their behaviour when seated with another bot will depend on the hand histories stored for that opponent's name. These 43 strategies are meant to provide a varied challenge for the purchasers of the software. The bots themselves (especially the Pokibot and Simbot types) are based on the work by the University of Alberta group. Note that we do not directly use this information. The procedure works from examples of play; it does not require any additional information about the strategies employed.

The experiments are based on short matches. Each match consists of 120 deals of 6-player limit hold'em. No bot has any memory of its opponents at the start of the match. This is important as it means each match independent and identically distributed, making statistical analysis of the results much simpler.

The matches are controlled by a series of scripts that automate the process of starting and finishing the matches by controlling the mouse using the free software Autoit (Bennett and Autoit Consulting). At the start of each match, a PA bot type is drawn, with Pokibots, Jagbots and Simbots drawn in the ratio 5:3:2. PA itself then draws a bot within the type for each seat. At the end of each match, another script deletes PA's memory of the match.

The first step is to generate a sample of the play of PA bots. 375 matches were played and the play recorded. Only hands that remained at a showdown were stored. Players were not allowed to conceal losing hands at a showdown. Concealing losing hands at a showdown is known as mucking. Allowing mucking would not block any of the techniques we use but would add another complication.

To get an idea of the general style of play of the bots in the population we use the same statistics as were used previously in describing the best response in section 5.3. The standard deviation between the different PA bots in the sample for the statistics is also calculated.

	mean	std.dev	VP$IP	Fold%	Ww/oSD	W$SD	AF
Average PA bot	0	8.38	35.0%	28.4%	27.5%	45.7%	0.90
Std. Dev. between bots	0.17	1.25	8.1%	6.4%	5.1%	3.8%	0.35

Table 6-1: Summary statistics for the PA population when playing against each other

This sample was divided into a hold-out set to test for overfitting and a main sample used to fit the opponent model. Each experiment involves the same three steps; fit an opponent model to the main sample, train an exploitative bot in simulated matches against the model and then play against the actual PA bots. The VP$IP of the various bots is at the top of the recommended range mentioned in the "Card-Player" article (Tanenbaum 2007) of 25%-35%. The aggression factors are at the bottom of the typical range identified in that article (0.9-1.4) and very few of the bots would be in the recommended range of aggression (1.4-2.4).

The experimental set-up is deliberately chosen to be more challenging than previous work. The match length is shorter than usual so if adaptation is to be successful it will have to be quicker than that shown in previous studies. The opponent population is much more sophisticated than usually used in these types of studies. In particular, the population does not include any trivial bots that always calls or always raises. Finally, using a multi-player game means that a bot has an incentive to learn from interactions that don't directly involve itself; if a bot folds early in a deal it should use the rest of the deal to help identify opponents.

It is worthwhile discussing the differences between the experimental set-up just described and the problem faced by an internet poker player. Internet poker is not divided into fixed length matches. Players have a lot of control over how long they play and who they play against. This is called table selection in the poker literature. Also most sites do not allow a player to use a new nickname every time he starts a session, so play in each session cannot be classes as independent. Most sites allow player to record deals that they played and maybe a finite number of others. Of course,

there is little they can do to stop these logs being swapped. This makes modelling the beliefs of players very difficult as it is hard to tell which data it is based on.

6.5.1. Alternative performance measures

Early testing indicated that the modelled performance seemed to be routinely optimistic. The performance in the simulated hands was significantly better than the performance when the bot actually played against the Poker Academy bots. It was also noticeable that the exploiter was playing a high variance strategy; its variance was much higher that is usually seen by quality bots. This seemed to be because it was optimistic about its return, it was willing to play more deals and take more risks. To counteract this tendency to select an aggressive high variance strategy, the target of the exploiter was changed. Instead of asking the exploiter to optimise its mean return, the Sharp ratio (SR) or the gambler ruin ratio (GR) was used as the target. These are explained in section 2.1.9 on page 20.

Changing the objective to one of these ratios is reasonably straightforward. Firstly express the ratio as a function of the first two moments about zero.

$$SR = \frac{E(x)}{\sqrt{E(x^2) - (E(x))^2}} \qquad GR = \frac{E(x)}{E(x^2) - (E(x))^2}$$

Then calculate the partial derivatives of the ratios with respect to the moments.

$$\frac{\partial SR}{\partial E(x)} = \frac{E(x^2)}{\left(E(x^2) - (E(x))^2\right)^{3/2}}$$

$$\frac{\partial SR}{\partial E(x^2)} = \frac{-\frac{1}{2} E(x)}{\left(E(x^2) - (E(x))^2\right)^{3/2}}$$

$$\frac{\partial GR}{\partial E(x)} = \frac{E(x^2)+(E(x))^2}{\left(E(x^2)-(E(x))^2\right)^2}$$

$$\frac{\partial GR}{\partial E(x^2)} = \frac{-E(x)}{\left(E(x^2)-(E(x))^2\right)^2}$$

The new reward functions are formed by adding a quadratic term to the basic reward so that the ratio of the quadratic term to the linear term is proportional to the ratio of the partial derivatives for the objective desired. Hence the reward function is proportional to the change in the objective given by a small change in the moments. Of course the true moments for the current strategy are not known so they are approximated by a running average during training.

So the reward for the Sharpe ratio is $x + \left(\frac{-\frac{1}{2}\bar{x}}{\overline{x^2}}\right)x^2$

And for the Gambler's Ruin ratio is $x + \left(\frac{-\bar{x}}{\overline{x^2}+\left(\bar{x}\right)^2}\right)x^2$

While this device did reduce the highly aggressive nature of the strategy found, a more principled approach would be preferable. This might allow an optimal correction to the bias generated by the opponent model. Of course, in some applications these ratios are the appropriate objective and hence are of interest in their own right.

Note that if a strategy is already generating a profit ($\bar{x} > 0$) then optimising either of these ratios will discourage the strategy from taking risks but if it isn't they will encourage it to take risks. If the strategy is generating approximately break-even results using one of these ratios will have no effect on the rewards perceived by the strategy.

6.6. Results

The experiments consist of testing different combinations of opponent model and exploiter.

The values reported for each system are:

- The entropy after imputation calculated on the holdout dataset; this is calculated as the average of the negative log-likelihood generated by the model of the decisions seen given any cards that have to be imputed.
- The modelled performance (mean and variance): This is the performance that the exploiter bot achieves in the simulated hands it learns from after the exploiter has converged. Convergence is diagnosed by charting the average reward; if the reward shows no upward trend over the last 50 million deals the process is stopped.
- The real performance (mean and variance): This is the performance in matches against the Poker Academy bots.

Ten systems are tested. The systems are named using the following codes.

Self-play is the system described in the previous chapter and entered in the 2008 AAAI poker competition.

OM – indicates the basic opponent model is fitted

PT – the model with player-types is fitted

The exploiter has the same structure as the model fitted unless the following letters are added to the name

-EL – the extra layer with the opponent tiles is used

The exploiter's target is to optimise its mean return unless the following abbreviations are added.

SR – the Sharp Ratio is used as the target

GR- the Gambler's Ruin ratio is used as the target.

	Fit	Model mean	Model std dev	run	mean	std dev
Self-play	81.00%	0	6.2	63960	0.008	7.004
OM	46.00%	0.69	12.03	92640	0.209	11.842
OM -SR	“	0.63	9.69	66960	0.285	9.276
OM-GR		0.395	6.924	66840	0.200	6.733
PT	34.00%	0.385	10.12	113160	0.201	10.256
PT-SR	“	0.373	9.052	51120	0.244	8.891
PT-GR	“	0.265	7.062	32160	0.238	7.197
PT-EL	“	0.632	12.232	53670	0.362	12.16
PT-EL-SR	“	0.585	9.832	52320	0.412	9.848
PT-EL_GR	“	0.405	7.833	52920	0.363	7.667

Table 6-2: Results for PA experiments

A couple of things are noticeable. Firstly the systems designed to optimise Sharpe Ratio also appear to generate higher mean returns. Also note that the performance generated in simulated deals against a model of opponents is always higher than that observed against the opponents themselves. The model always has an optimistic bias. This is possibly because there are some elements of the PA bots' strategy it can't model and these generate unmodeled profit for the PA bots.

We reproduce the summary statistics for each of the exploiter systems. (Definitions for these statistics are given in Section 5.3. The most notable difference between the exploiters and the PA population is that the aggression factor for the exploiters is always higher.

	VP$IP	Fold%	Ww/oSD	W$SD	AF
SP	32.8%	31.2%	28.2%	51.0%	1.02
OM	76.3%	38.0%	31.0%	52.2%	2.03
OM-SR	58.1%	36.9%	33.9%	54.6%	1.86
OM-GR	41.8%	35.3%	39.1%	57.8%	2.01
PT	66.1%	31.9%	33.5%	42.6%	2.50
PT-SR	45.8%	24.0%	38.0%	42.6%	2.63
PT-GR	34.1%	22.1%	41.3%	46.0%	2.60
PT-EL	76.6%	36.4%	33.3%	51.0%	2.55
PT-EL-SR	55.8%	34.5%	37.1%	53.5%	2.45
PT-EL-GR	39.2%	31.9%	40.1%	56.3%	2.71

Table 6-3: Statistics for the various exploiter systems

It is noticeable that the VP$IP figures is higher than that recommended in the "Card-Player" article (Tanenbaum 2007)that was mentioned previously. It recommends a VP$IP of 25%-35% for 6-player hold'em while the systems here (except self-play) are all considerably above that range. Only the GR systems approach the recommended range. The aggression factors show less variation and most are only slightly above the recommended range of 1.4-2.4.

The results of the systems are summarised in *Figure 6-3*. The central dash for each experiment represents the observed mean and the bar represents two standard errors of the mean either side of the mean. The systems are ordered by their observed mean.

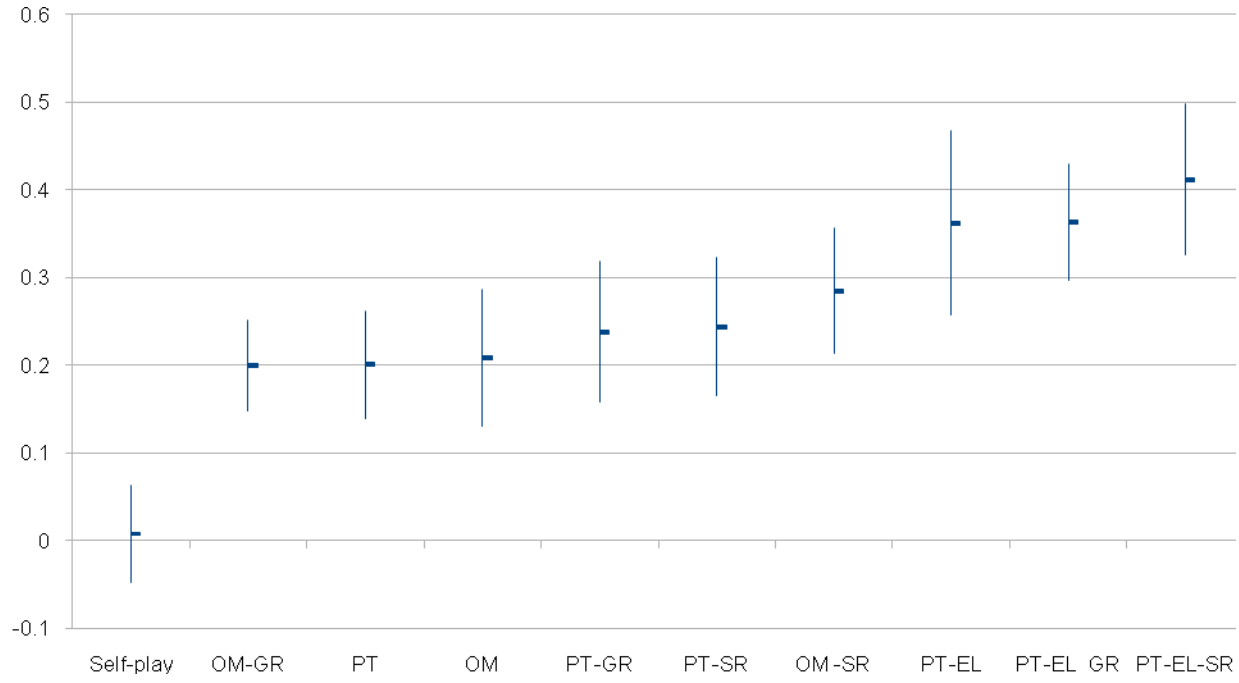


Figure 6-3: Performance confidence intervals for the systems tested

SPSS (IBM 2008) was used to check the data for statistically significant differences in mean performance between the different systems. The Dunnet's C procedure for post-hoc pair-wise comparisons was used. This corrects for the differing run lengths and return variances for each complete system. The results show that the performance of the self-play system is significantly worse than all of the modelled systems. At the 5% significance level, 5 other comparisons were found to be statistically significant out of a total of 36 possible pair-wise comparisons. The PT-EL-SR system is better than the OM, the OM-GR and the PT system. The PT-EL-GR system is better than the OM-GR system and the PT system. Note that all the significant results involve a system in which the exploiter has an extra layer. Learning that a larger system with more free parameters generates better performance is not surprising. So while certain patterns appear in the results they are not clearly statistically significant.

Rollout improvement errors were also calculated for some of the systems. Rollout improvement is not as important in this section of the thesis because the performance can be directly measured in matches with PA. But it is still of interest to generate the rollout errors for at least some of the simulated systems.

These are the calculations for the PT-EL system.

	Pre-flop	Flop	Turn	River	Overall
% of Dec.	37%	29%	18%	16%	100%
Sample error	0.572	0.819	1.042	1.031	0.805
LFO error	0.791	0.963	1.143	0.632	0.882
RI error	-0.218	-0.144	-0.101	0.399	-0.077
Std. Dev of RI error	0.036	0.047	0.125	0.158	0.046

Table 6-4: Rollout Improvement errors for the PT-EL system

The comments about the rollout error given in section 5 on page 76 still apply. Of course these errors are only with respect to the model; how well they reflect the true decision errors depends on how well the model matches the system.

The LFO errors are substantially larger than in the self-play experiments at all stages of the game. This is not surprising as this is a far more aggressive game than seen in the self-play section; pots are substantially larger which affects the scale of the errors.

6.6.1. Evidence of successful adaptation

The first reason why the adaptive systems could fail is if the system could fail to discriminate between any players within the short matches we are using here. To check this we use records of one of the experiments with Poker Academy. At the end of each deal, the information remaining (entropy) in the player-type distribution is calculated for all players. The average is then calculated for each deal number after separating the exploiter itself from the PA bots and the results plotted. *Figure 6-4* was generated by play between the PT system exploiter and the PA bots.

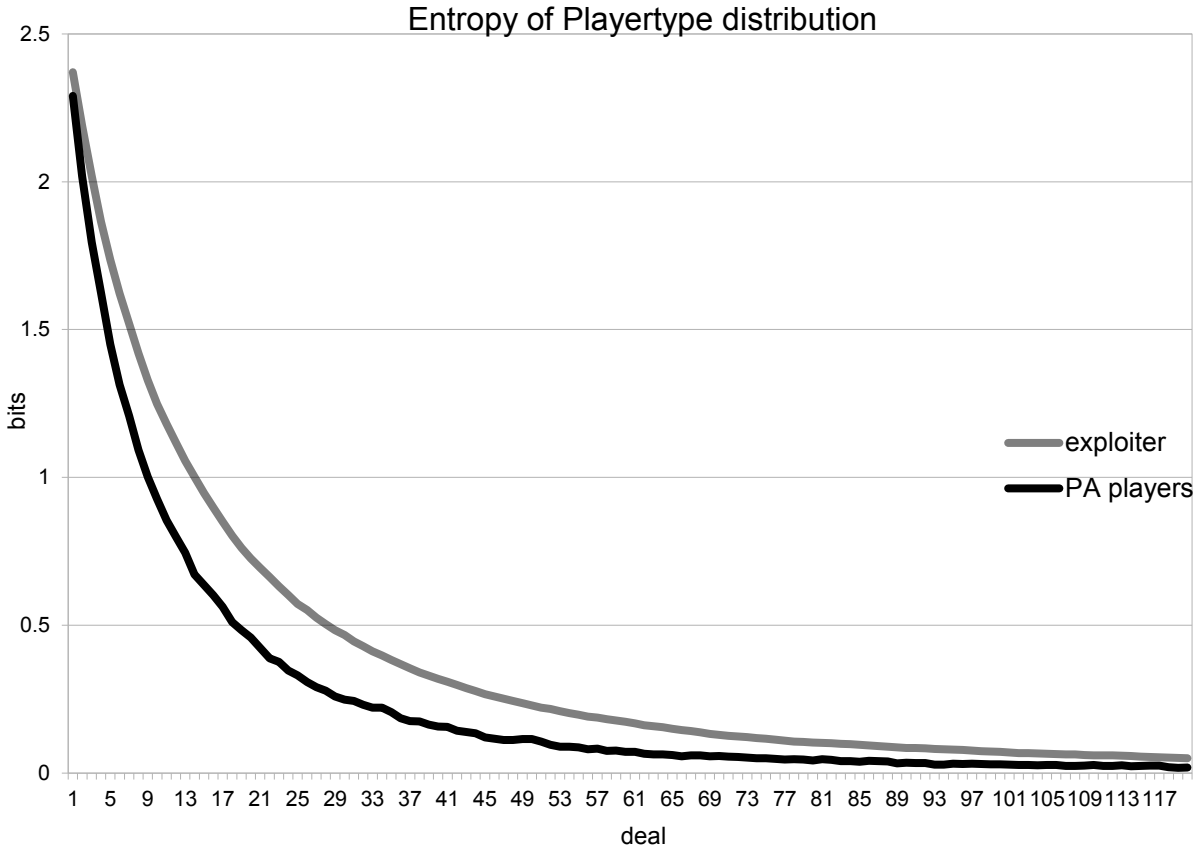


Figure 6-4: Entropy of player-type distribution by deal number

The graph shows that the model is able to distinguish types within a 120 deal match. Otherwise there would be no noticeable reduction in the entropy during the match.

Well before the end of the match there is little uncertainty about the type of each player according to the model.

As another check, the mean player type distribution at the end of a match is calculated for each named PA bot. Comparing the mean of the entropy in these distributions with the information of the original distribution shows that the model is discriminating between the different bots successfully. For the same sample, the mean information given the opponent's name is 0.225 bits as against the entropy of the initial distribution which is 2.482 bits. The figure for each name is higher than the average at the end of each match. Inspecting the information indicates that the some bot will be definitively assigned to a different type at the end of different matches. This may be because the play of these bots is between two or more prototypes and the different play of the other players in each match might mean that it appears more like one or other of the prototypes in different matches.

The next chart to look at is a graph of mean return by deal number for simulated deals. (Using simulated deals means we can use a large sample size)

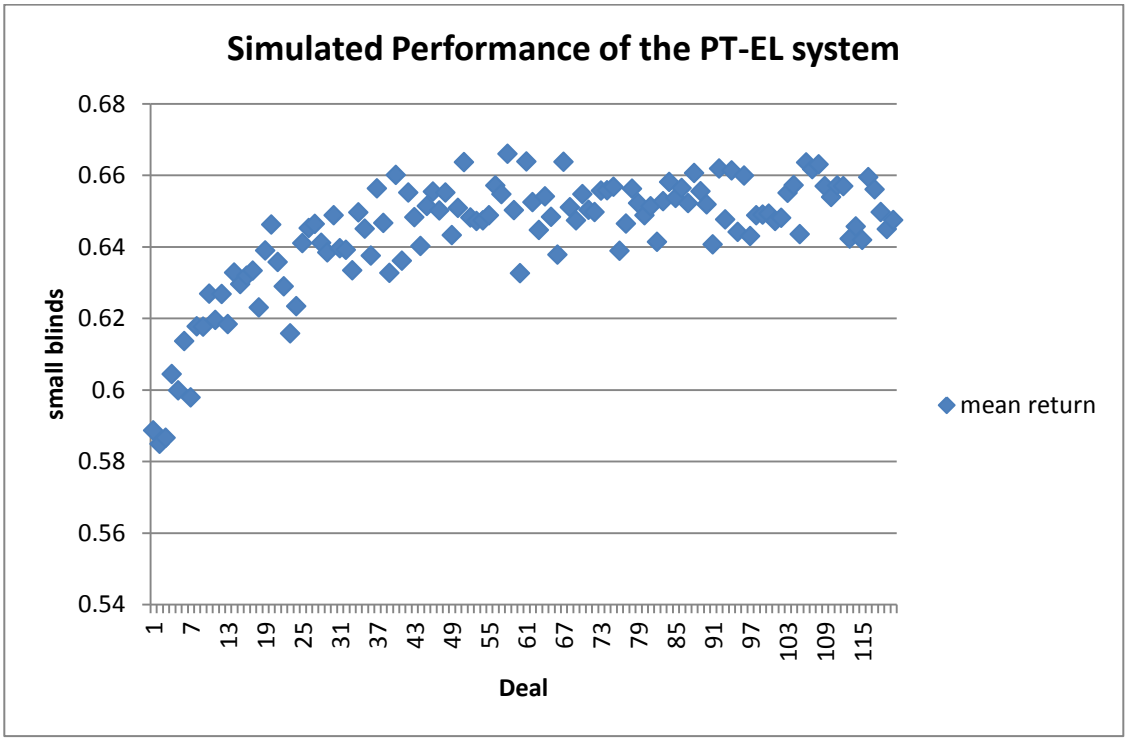


Figure 6-5: Mean return by deal number chart for the PT-EL system

Each point is the average of 28.6 million deals generated by the PT-EL system against the PT opponent models. Note that even with this large sample size, only the broad pattern is discernable. These large sample sizes are possible with this system because of its simplicity which allows it to generate actions quickly.

The shape of this graph is as expected. The curve roughly matches that for the information in the player type distribution. But the gap between the mean return at the start and end of the match is quite small. A possibility that must be considered is that the opponent models are doing some successful adaptation itself. (Remember the current player-type distribution for its opponents is an input in the PT model.) To check whether this is occurring, a special experiment was conducted. The exploiter plays each deal in the match as if it was the first but the opponent models play as normal. This means two player-type-card-type joint distributions must be maintained; the exploiter uses one that is reset to the initial player-type distribution at the start of each hand but the opponent models use one that carries forward from the previous deal. To aid comparison this set of results is added to Figure 6-6.

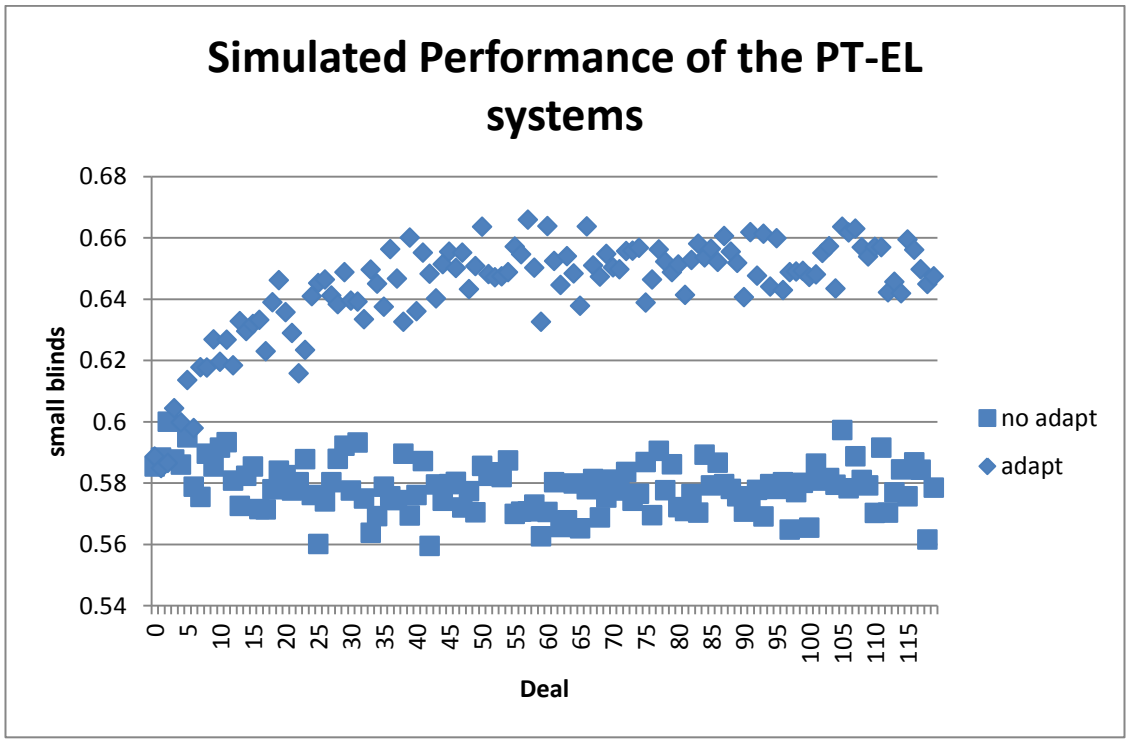


Figure 6-6: Comparison of adaptive and non-adaptive systems

Figure 6-7 reproduces the mean return by deal number charts, but using deals generated with the PA bots rather than their models. This illustrates the difficulty clearly. The adaptation observed with the opponent model may carry over to the experiment with the PA bots but if it is on the same scale as seen with the models, it will not be discernible with the small sample sizes possible with PA.

Small effects of the size seen with the opponent model (of the order of 0.09 chips) are going to be lost in the variation of a sample of 448 matches. (Note that while this is the same graph for the test as the one for the simulation the scale of the variation is much wider.)

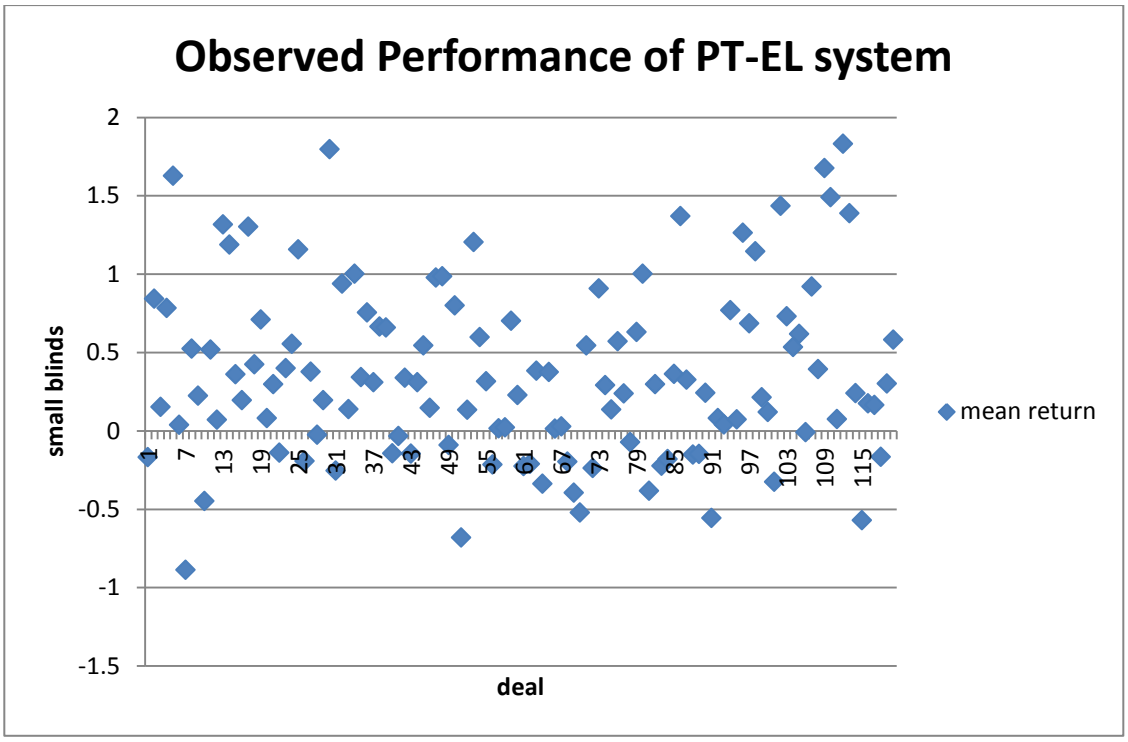


Figure 6-7: Observed mean return by deal number chart for the PT-EL system

6.7.Discussion

There are two main conclusions to be drawn from this set of experiments. Firstly, modelling the population results in a clear statistically significant improvement in

results (Table 6-2) compared to the results of a similar bot trained by self-play. As expected, using a sample is better than ignoring it. But there is no evidence that the method of adaptation used generates statistically significant improvements. A possible reason for this is that the variation between members of the population found by the strategy may be small. In that case it will take too long to identify the different types for the identification to be useful in the short matches we consider here. This is not the case with this experiment. Results show that after around 30 deals the method is fairly certain which type each opponent represents. This leaves three quarters of the match to use that information. Note that this is without an exploration bonus. A more sophisticated algorithm would include an estimate for the value of information and this would be added to the reward function to encourage the bot to take actions which reveal useful information. As learning the types seemed to occur without difficulty and rapidly, adding a potentially complicated estimate for the value of information was deemed unnecessary. In a two player game, exploration is likely more difficult because if a player doesn't explore an opponent's response to a particular bet sequence he can't hope that another player explores a similar sequence for him.

An alternative explanation is that the structure does not have enough flexibility to make use of the information available. This is certainly possible but the adaptive structure is based on the structure used in the other sections of the thesis where it has positive results. The rollout improvement errors show that while it does not produce perfect play, it does not generate too many gross errors.

The next explanation to consider is that the strategy types generated by the model represent low value information. Maximum likelihood finds models that generate the most reduction in entropy from data. This means that it will find types that it can distinguish as quickly as possible. But this mightn't generate the highest value of information. The final explanation to consider is that there isn't a lot of high value information in this experiment. While the strategies display distinctive styles, there may be little to be gained from adapting to the differences. This may be a property of the particular population studied or of multi-player limit hold'em in general.

Of course statistical significance could always be improved by increasing sample sizes. This presents practical issues because "Poker Academy" was not designed for testing

and it does not operate quick enough to generate very large samples. It is also unsatisfactory because effects that require a large sample size to uncover are unlikely to be practically significant.

7. Conclusions

7.1.Summary

The aim in this thesis was create a system that could be trained to play poker well by reinforcement learning. In particular, the aim was to have a system that could optimise its returns against an observed population. To do this a novel structure was designed. Three sets of inputs were used. The first set describes the state of the betting after any legal action. The second set describes the cards visible to the player at any point in the game. These sets of basic features are typical of the inputs usually used with a connectionist structure applied to perfect information game (i.e. a typical board game). The third set of inputs are more sophisticated. They are generated by a hidden Markov model (called the observer model) which approximates the play of a deal of poker by replacing the hidden cards by a set of hidden variables called card-types (one for each stage of the game). The belief-vectors over card-types generated by this model are then used as inputs to the model. They represent what can be deduced about an opponent's private information from their actions during a deal.

This structure is then tested in two settings. Firstly it trained by self-play. Most of the structure is trained by reinforcement learning to make decisions and the components only used by the observer model are trained to best fit the play without using the hidden cards. It is difficult to assess the performance of a self-play system in a zero-sum game as it the overall mean performance is by definition zero. Instead the quality of individual decisions was compared to that of would be generated by a rollout. This work was used to generate a series of entries to the AAAI hosted annual computer poker competition.

The second setting that the structure was tested in is that of generating a population best-response. A commercial software package (Pokibot) was used to provide the population of opponents. In this setting the structure was used in two ways; firstly as an opponent model and then as an exploiter which will optimise its return against the opponent model. A simple opponent model is compared to one that includes a hidden variable (the player-type) that models some of the variation between the different players in the population. The player-types are included to allow the possibility of

adapting to the variation between the members of the opponent population. Early experimentation suggested that the exploiter learn to be over optimistic when playing against the opponent model. This resulted in a more aggressive strategy that conventional thinking on the game of multi-player limit hold'em would consider wise. To investigate this some alternative performance measures to the mean reward were considered. These measures encouraged the structure to choose a less aggressive strategy. A series of experiments were then conducted to investigate the performance of the different variations of the structure (opponent model or exploiter) and the targeted performance measure. The opponents are redrawn every 120 deals so that if adaptation is to generate improvements, it will have to do so rapidly.

In the next 2 sections, the main findings and point of originality are summarised in the form of lists.

7.2.Findings

1. It is possible to design a connectionist structure that can play a sophisticated strategy but retains the advantages of this approach, that it acts quickly and that it is can be trained by the gradient ascent algorithms popular in machine learning. The flexibility of the structure is shown by the ease with which connections can be added such as the extra layer in section 6.4.

For the self-play work,

2. Reinforcement learning can be used improve the performance of such a structure. This is shown by the decreasing rollout error seen during the training run [Section 5.3] where a significant improvement is shown over the run.

3. The result of a self-play training run is a poor approximation to an equilibrium. This is evidenced by the significant gain found when a best response was trained. In fact, the population of Poker Academy used in the experiments in Chapter 5 and described in section 6.5 is a better approximation to an equilibrium. A best response to this population generates a much smaller gain.

   The theory of self-play gradient ascent learning says that it will not converge to a mixed equilibrium. It is still interesting though to see how significant this theoretical concern is in practice. In this application, the bot trained by self-play alone is a long way from equilibrium so concerns about convergence are significant.

   This does not mean that the resulting bot is easily dominated by current poker AI. In 2008, it had the 3rd best performance out of the 6 entrants and could have finished higher if the last placed finisher had been replaced by a different player. Even in the 2009 AAAI competition when it finished 2nd last in both the 2-player and the 3-player competitions, which was before the self-aware refinement (in 2010), the bot only lost by a small amount on average to all entries. With the refinement, it came 2nd in the 2010 3 player competition.

For the exploitation work,

4. Framing the task as one of seeking the best response to a specific population leads to a well-defined problem with a single performance measure; average reward in tests against opponents drawn from the population.

5. It is possible to fit the structure to samples of the play. The cards that aren't revealed at showdown can be imputed. Rejection sampling can be used to impute the hidden cards in the vast majority of cases. The rare exception is where a long sequence of actions ends without seeing the cards. In such cases, we revert to enumerating over all possible opponent two card combinations. This is the only point in the thesis that enumeration is used. A considerable improvement in the quality of the fit (as measured by the likelihood of the observed actions) is seen as compared to considering all actions equally likely.

6. It is possible to model the largest differences in play between different members of the population by adding a hidden variable. This hidden variable groups similar players into types. Adding these types causes a 12% improvement in the fit for the Poker Academy (BioTools Incorporated 2007) population studied.

7. These player-types make it possible to start to distinguish between opponents using relatively few deals. Well before the end of a 120 deal match, the model has assigned each opponent from Poker Academy to one of eight types with little uncertainty.

8. Being able to distinguish opponent playing styles rapidly does not generate a substantial improvement in performance for this population of opponents. This is a surprise because the importance of adapting to opponents is emphasised in most strategy guides. Possibly, it is better to find the most important

differences (which will reward adaptation most) rather than the most obvious differences (which maximum likelihood will find).

9. Fitting a model and then learning a strategy in simulations with that model shows an optimistic bias here. The actual performance achieved in testing is always less than that indicated in the simulation. This probably generalises to many situations where a similar process is used. If the model allows more reward than is really possible in some situations, the optimiser will tend towards those situations. Being aware of this problem means that corrective measures can be taken. Here we change the reward to ones that discourage the aggressive strategies that the optimistic bias tends to generate. The results suggest that this was somewhat successful without generating a statistically significant improvement on the sample sizes (~50,000 deals) used in our experiments.

7.3.Originality

1. The structure used is a new design. The aim was to design a structure so that reinforcement learning and parametric model fitting could be extended to poker. A connectionist structure in the form of three layer perceptron has been used before as an opponent model in later versions of Pokibot (Davidson 2002). This work though uses a novel structure and unlike a lot of the recent work on limit hold'em, it can work with any number of players instead of only two or three.

2. The use of reinforcement learning to train a poker AI system is novel. As far as we are aware, this is the first use of reinforcement learning in poker AI research. This is despite poker appearing to a natural domain for reinforcement learning; it is stochastic decision making problem.

3. Hidden Markov models (HMMs) have not been mentioned before in the published work in poker AI. This is surprising, as the use of HMMs is common in other applications where the true state is hidden. The card-type variable, on which the HMM is based, performs a somewhat similar function to the card buckets used in the game abstraction approaches. This is because at any stage of the game, a belief vector over the types or the buckets is available. The first difference is that the card-types don't limit the play of the structure; it can play every different set of private cards differently; in the game abstraction approach all cards in a bucket are player the same way. The second difference is that the card-types are trained specifically for a particular game and population of players. The buckets are in either designed from heuristics (Billings, et al. 2003) or learnt by a clustering algorithm (Gilpin and Sandholm 2006). The HMM model predicts the outcome of the rest of a deal from the play so far without using any private information.

4. While adapting to opponents has been a concern of much work in poker AI, it has mostly in terms of seeking a best response over a long contest against a single opponent. The question considered here of adapting quickly to new opponents from a population has only generated published work recently. Dividing members of a population into a small number of types is a popular technique in consumer marketing but has only been used in poker recently. Van der Klej's M.Sc. (van der Kleij 2010) uses a similar approach with a different base system.

5. Rejection sampling has not made an appearance in the published work in poker AI. The attractiveness of this method of sampling from an opponent's hand distribution is that it does not need to enumerate over all possible combinations of private cards. This is important if work is to be extended to games with more than two hidden cards per player. Rejection sampling was not a complete success as enumerating overall possible hands before sampling was required

for the rare long sequences that ended in a fold.

7.4.Future work

Because of the self-imposed constraints used in designing the basic system, specifically by avoiding any components whose time usage would scale badly, there is considerable room to extend this work in a number of directions. These possible directions are now outlined starting with the work that could be implemented almost immediately and then continuing to more speculative directions.

The first direction is actually the subject of early testing at the time of writing. As mentioned in section 2.3.7, self-play learning will not converge to a mixed equilibrium. The lagging anchor algorithm has been proposed as a solution to this problem. This algorithm involves two strategies; the self-play strategy that generates a training run of games and a "lagging anchor" strategy. The self-play strategy is trained by reinforcement learning with as its target a sum of its reward in the game and a measure of its distance from the lagging anchor. The lagging anchor is trained to represent an average of the self-play strategy over the course of the training run. The advantage of this approach is that it can be added to the current self-play set-up without too much coding work. A difficulty is the number of hand-tuned learning rates increases in this approach. Early testing indicates that performance is quite sensitive to the choice of learning rates. If successful this would result in a system that was able to converge to mixed equilibrium and hence better approximate an equilibrium.

The next direction to consider is to extending the work to other versions of poker. Extending the work to games other than hold'em would involve replacing the card features with features that are appropriate for the new game. Games where the raise amount can vary add a different challenge. Throughout the thesis the desirability of avoiding enumerating over a finite number of fixed cases is emphasised. But the system does rely on there being at most three options at every decision in limit poker. This is not a limitation of the policy-gradient reinforcement learning algorithm used which can learn a strategy over a continuous space of actions but of the way the structure is put together. One option that has been explored is to artificially constrain

the choice of raise amount to finite number of choices. Two difficulties have been identified with this approach. Firstly if it is combined with a solution technique that involves enumerating over all betting sequences, then only a very small number of different raise amounts can be specified without causing an exponential explosion in the size of the betting tree. Secondly how to interpret the actions of an opponent who is using different raise amounts; raise sizes that are allowed by the rules but eliminated from consideration when the bot was being designed. Any system based on our current system would not suffer from the first problem as increasing the number of options at each decision point would only result in a proportional slow-down in the decision making. The second problem could be overcome by forcing the system to select from a finite number of raise-size distributions; the actual raise size to be drawn from that distribution. This means that raises made by opponents could be interpreted as an observation drawn from one of the raise size distributions. Dealing with continuous action-spaces is an important problem if the work in poker AI is to be extended because many economic applications involve continuous action spaces.

The main limit on the use of Monte-Carlo methods is that it often takes a large sample to make effective decisions and hence struggle to make decision quickly. This limits their effectiveness in many applications. The results of rollout improvement (RI) error calculations suggest that using a sample could improve the performance of system. One idea is to allow the structure to use a sample like that used in the RI calculations but only on a subset of the decisions. If the subset was small there would not be a substantial slow-down. The design decisions to be made would then be: how to decide which decisions to sample? How to decide sample size? How to combine sample results with the original action preferences? How to penalise taking a sample so that the structure only uses samples sparingly? While that may appear like a long list, if a basic architecture can be designed then machine learning can be used to fine-tune the performance. Whether to take a rollout sample is a decision and therefore is amenable to reinforcement learning.

7.5.Final Comment

Ideally, success in designing poker AI systems will lead to insights in designing systems for a much larger set of games. Otherwise there is a danger that the work will become so specialised that it becomes divorced from the rest of academic research. The first targets for generalisation are obviously the many variations of poker but beyond that there is wide variety of games for which private information is central to the strategy. These include other card games but also games that are designed to model real world situations. For this reason, interest in poker AI should focus not only on the best performance on a specific version of poker but also the performance of approaches that leave room for extension. Hopefully this thesis will prove useful to future researchers that are considering using a connectionist approach to a hidden information game.

8. References

Aberdeen, D. 2003. *A (Revised) Survey of Approximate Methods for Solving Partially Observable Markov Decision Processes*. Technical Report. Research School of Information Science and Engineering, Australia National University.

Andrieu, C., De Freitas, N., Doucet, A. and Jordan, M.I. 2003. An introduction to MCMC for machine learning. *Machine Learning*, 50(1), pp.5-43.

Auer, P., Cesa-Bianchi, N. and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2), pp.235-256.

Auer, P., Cesa-Bianchi, N., Freund, Y. and Schapire, R.E. 1995. Gambling in a rigged casino: The adversarial multi-armed bandit problem. *IN: focs*. Published by the IEEE Computer Society.

Bard, N.D.C. 2008. *Using State Estimation for Dynamic Agent Modelling*. M.Sc. University of Alberta.

Bennett, J. and Autoit Consulting. *Autoit* (3) [Computer Program]. Available from: www.autoitscript.com

Bertsekas, D.P. and Tsitsiklis, J.N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Billings, D. 2006. *Algorithms and Assessment in Computer Poker*. Ph.D. University of Alberta.

Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T. and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI)*,

Billings, D., Davidson, A., Schaeffer, J. and Szafron, D. 2002. The challenge of poker. *Artificial Intelligence*, 134(1-2), pp.201-240.

Billings, D., Davidson, A., Schauenberg, T., Burch, N., Bowling, M., Holte, R., Schaeffer, J. and Szafron, D. 2004. Game tree search with adaptation in stochastic imperfect information games. *Computers and Games*, pp.21–34.

Billings, D. and Kan, M. 2006. A tool for the direct assessment of poker decisions. *The International Computer Games Association Journal*, 29(3), pp.119-142.

Billings, D., Papp, D., Schaeffer, J. and Szafron, D. 1998. Poker as a testbed for machine intelligence research *IN:* Mercer, R. and Neufeld, E. (eds.) *Conference on Artificial Intelligence*. Springer-Verlag, pp.228-238.

Billings, D., Pena, L., Schaeffer, J. and Szafron, D. 1999. Using probabilistic knowledge and simulation to play poker. *AAAI National Conference*, pp.697-703.

Billings, D., Schaeffer, J. and Szafron, D. 1998. Poker as a testbed for machine intelligence research.

Binmore, K. 1992. *Fun and games, a text on game theory*. DC Heath and Company.

BioTools Incorporated. 2007. *Poker Academy Pro* (2.5.9) [Computer Program]. Available from: www.poker-academy.com

Bowling, M. and Veloso, M. 2002. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2), pp.215-250.

Brecher, S. *Hold'em Showdown* [Online]. Available from: http://www.brecware.com/Software/software.html [Accessed 01/03/2008 2008].

Brunson, D., Baldwin, B. and Goldberg, A. 1979. *Super/system: A Course in Power Poker*. 3rd ed. Cardoza.

Chaslot, G.M., Winands, M.H.M., Herik, H., Uiterwijk, J.W.H.M. and Bouzy, B. 2008. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation (NMNC)*, 4(03), pp.343-357.

Chen, B. and Ankenman, J. 2006. *The Mathematics of Poker*. ConJelCo LLC.

Ciaffone, B. and Brier, J. 2002. *Middle Limit Hold'em Poker*. Bob Ciaffone.

Cover, T.M. and Thomas, J.A. 1991. *Elements of information theory*. New York: Wiley.

Crawford, V.P. 1974. Learning the optimal strategy in a zero-sum game. *Econometrica: Journal of the Econometric Society*, 42(5), pp.885-891.

Dahl, F.A. 2005. The lagging anchor model for game learning--a solution to the Crawford puzzle. *Journal of Economic Behavior & Organization*, 57(3), pp.287-303.

Dahl, F.A. 2002. The lagging anchor algorithm: reinforcement learning in two-player zero-sum games with imperfect information. *Machine Learning*, 49(1), pp.5-37.

Davidson, J.A. 2002. *Opponent modeling in poker: Learning and acting in a hostile and uncertain environment*. M.Sc. University of Alberta (Canada).

Dayan, P. 1991. Reinforcement Comparison. *IN: Proceedings of the 1990 Connectionist Models Summer School*. Morgan Kaufman.

Ferguson, C., Ferguson, T.S. and Gawargy, C. 2007. U (0, 1) two-person poker models. *Game Theory and Application, XII*, (12), pp.17-37.

Ganzfried, S. and Sandholm, T. 2008. Computing an approximate jam/fold equilibrium for 3-player no-limit Texas Hold'em tournaments. *IN: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems.

Gilpin, A. and Sandholm, T. 2006. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. *IN: Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Ginsberg, M.L. 1999. GIB: Steps toward an expert-level bridge-playing program. *IN: international joint conference on artificial intelligence*. Citeseer.

Gittins, J.C. and Whittle, P. 1989. *Multi-armed bandit allocation indices*. Wiley New York.

Halck, O.M. and Dahl, F.A. 1999. On classification of games and evaluation of players–with some sweeping generalizations about the literature. *IN: Proceedings of the ICML-99 Workshop on Machine Learning in Game Playing*.

Hoda, S., Gilpin, A., Peòa, J. and Sandholm, T. 2010. Smoothing techniques for computing Nash equilibria of sequential games. *Mathematics of Operations Research*, 35(2), pp.494-512.

Hoehn, B. 2006. *The effectiveness of opponent modelling in a small imperfect information game*. M.Sc. University of alberta.

Huang, X., Acero, A. and Hon, H. 2001. *Spoken language processing : a guide to theory, algorithm, and system development*. Upper Saddle River, NJ: Prentice Hall PTR.

IBM. 2008. *SPSS* (17) [Computer Program]. Available from: www.spss.com

Jebara, T. 2004. *Machine learning : discriminative and generative*. Dordrecht ; Boston: Kluwer Academic Publishers.

Jensen, F. 2001. *Bayesian networks and decision graphs*. New York: Springer.

Johanson, M. and Bowling, M. 2009. Data biased robust counter strategies. *IN: Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS-09)*.

Johanson, M.B. 2007. *Robust strategies and counter-strategies: Building a champion level computer poker player*. M.Sc. University of alberta.

Knowledge Engineering Group, TU Darmstadt *Computer Poker Bots* [Online]. Available from: http://www.ke.tu-darmstadt.de/resources/poker/ [Accessed 5/19/2011 2011].

Kocsis, L. and Szepesvári, C. 2006. Bandit based monte-carlo planning. *Machine Learning: ECML 2006,* pp.282-293.

Kohavi, R. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence,* 2pp.1137–1145.

Kotnik, C. and Kalita, J. 2003. The Significance of Temporal-Difference Learning in Self-Play Training TD-rummy versus EVO-rummy. *IN: Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*. Washington DC:

Kozek, A.S. 1995. A rule of thumb (not only) for gamblers. *Stochastic Processes and their Applications,* 55(1), pp.169-181.

Lanctot, M., Waugh, K., Zinkevich, M. and Bowling, M. 2009. Monte carlo sampling for regret minimization in extensive games. *Advances in Neural Information Processing Systems,* 22pp.1078-1086.

Layton, R., Vamplew, P. and Turville, C. 2008. Using Stereotypes to Improve Early-Match Poker Play. *AI 2008: Advances in Artificial Intelligence,* pp.584-593.

LeCun, Y., Bottou, L., Orr, G. and Müller, K. 1998. Efficient backprop *IN:* Orr, G. and Müller, K. (eds.) *Neural Networks: Tricks of the trade*. Springer, pp.546-546.

Little, R.J.A. and Rubin, D.B. 1987. *Statistical analysis with missing data*. New York: Wiley.

Long, J., Sturtevant, N., Buro, M., Furtak,T. 2010. Understanding the Success of Perfect Information Monte Carlo Sampling in Game Tree Search.

McLachlan, G.J. and Krishnan, T. 1997. *The EM algorithm and extensions*. New York: John Wiley.

Michie, D. 1982. The state of the art in machine learning *IN:* Michie, D. (ed.) *Introductory readings in expert systems*. New York Gordon and Breach, pp.208-229.

Miltersen, P.B. and Sørensen, T.B. 2007. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. *IN: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM.

North, M.J. and Macal, C.M. 2007. *Managing business complexity: discovering strategic solutions with agent-based modeling and simulation*. New York: Oxford University Press.

Papp, D.R. 1998. *Dealing with imperfect information in poker*. M.Sc. University of Alberta.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems :: networks of plausible inference*. San Mateo, Calif: Morgan Kaufmann Publishers.

Ponsen, M., Gerritsen, G. and Chaslot, G. 2010. Integrating Opponent Models with Monte-Carlo Tree Search in Poker. *IN: Proceedings of Interactive Decision Theory and Game Theory Workshop at the Twenty-Fourth Conference on Artificial Intelligence*. AAAI Press.

Ponsen, M., Ramon, J., Croonenborghs, T., Driessens, K. and Tuyls, K. 2008. Bayes-relational learning of opponent models from incomplete information in no-limit poker. *IN: Twenty-third Conference of the Association for the Advancement of Artificial Intelligence (AAAI-08)*.

Rich, E. 1979. User modeling via stereotypes. *Cognitive Science*, 3(4), pp.329-354.

Risk, N.A. 2009. *Using counterfactual regret minimization to create a competitive multiplayer poker agent*. M.Sc. University of Alberta.

Risk, N.A. and Szafron, D. 2010. Using counterfactual regret minimization to create competitive multiplayer poker agents. *IN: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems.

Rubin, J. and Watson, I. 2011. Computer poker: A review. *Artificial Intelligence*, 175(5-6), pp.958-987.

Rubin, J. and Watson, I. 2010. SARTRE: System Overview A Case-Based Agent for Two-Player Texas Hold'em. *Relation*, 10(1.156), pp.4234.

Russell, S.J. and Norvig, P. 1995. *Artificial intelligence: a modern approach*. Englewood Cliffs, N.J: Prentice Hall.

Schauenberg, T.C. 2006. *Opponent modelling and search in poker*. M.Sc. University of Alberta.

Schnizlein, D. 2009. *State translation in no-limit poker*. M.Sc. University of Alberta.

Schweizer, I., Panitzek, K., Park, S.H. and Fürnkranz, J. 2009. *An Exploitative Monte-Carlo Poker Agent*. TUD–KE–2009-2. TU Darmstadt. [Online]. Available from: http://www.ke.informatik.tu-darmstadt.de/publications/reports/tud-ke-2009-02.pdf

Singh, S., Jaakkola, T., Littman, M.L. and Szepesvári, C. 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3), pp.287-308.

Singh, S., Kearns, M. and Mansour, Y. 2000. Nash convergence of gradient dynamics in general-sum games. *IN: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI 2000)*. Morgan Kaufman.

Sklansky, D. 2005. *Theory of Poker*. Two Plus Two Publishing.

Smallwood, R.D. and Sondik, E.J. 1973. The Optimal Control of Partially Observable Markov Processes over a Finite Horizon. *Operations Research,* 21(5), pp.1071-1088.

Smith, J.Q. 1988. *Decision analysis: a Bayesian approach*. London ; New York: Chapman and Hall.

Southey, F., Bowling, M., Larson, B., Piccione, C., Burch, N., Billings, D. and Rayner, C. 2005. Bayes' Bluff: Opponent Modelling in Poker. *IN: 21st Conference on Uncertainty in Artificial Intelligence, UAI'05.*

Spence, M. 1973. Job market signaling. *The Quarterly Journal of Economics,* 87(3), pp.355-374.

Sutton, R.S. and Barto, A.G. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Swingler, K. 1996. *Applying neural networks: a practical guide*. London: Academic Press.

Tanenbaum, B. 2007. Shorthanded Metrics - Important online tracking statistics. *Card-player*. [Online]. 20 (15),

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM,* 38(3), pp.58-68.

Tesauro, G. and Galperin, G.R. 1997. Online policy improvement using monte-carlo search. *Advances in Neural Information Processing Systems (NIPS),* 9

Van den Broeck, G., Driessens, K. and Ramon, J. 2009. Monte-Carlo tree search in poker using expected reward distributions. *Advances in Machine Learning*, pp.367-381.

van der Kleij, A.A.J. 2010. *Monte Carlo Tree Search and Opponent Modeling through Player Clustering in no-limit Texas Hold'em Poker.* M.Sc. University of Groningen.

Von Neumann, J. and Morgenstern, O. 1944. *Theory of Games and Economic Behavior*. Princeton University Press.

Waugh, K., Schnizlein, D., Bowling, M. and Szafron, D. 2009. Abstraction pathologies in extensive games. *IN: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems.

Williams, R.J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3), pp.229-256.

Zinkevich, M., Johanson, M., Bowling, M. and Piccione, C. 2008. Regret minimization in games with incomplete information. *Advances in Neural Information Processing Systems*, 20pp.1729–1736.

9. Appendix – Complete List of Input Features

The basic features used as inputs in the structure include two main categories; betting and cards.

9.1.Bet Features

There are only nine betting features used. Before each decision, they are calculated for the state of the betting after either calling or raising. Many of them will be the same for both calling and raising.

- The number of active opponents;
- Action will end the stage;
- The size of the pot;
- Amount owed; the amount to be added to the pot if all players were to call the current bet.
- Fair share; amount if the pot was split between all active opponents after they all call the current bet
- Opponents to act after on this stage; *The number of opponents that will definitely have at least one more decision at this stage. After a raise, this will be the same as the number of active opponents. For a call this may be less as players that have already acted may not get another action*
- Opponents to act after on the next stage; *isn't used on the river*
- Heads-up; *only two active players left*
- Bets until cap; *The number of subsequent Bets/Raises allowed. Remember under the rules of multi-player poker the number of bets on a stage is capped unless only 2 players are left.*

9.2.Card Features

	Description of category	Ranking within category
Straight Flush	All 5 cards of the same suit and of consecutive ranks	Top card in the sequence
Quads	4 cards of the same rank and 1 other (kicker)	By cards of the same rank and then by rank of the kicker
Full House	3 cards of one rank (trips) and two cards of another rank (pair)	By the rank of the trips first and then be the rank of the pair
Flush	All 5 cards of the same suit	By the highest card among the five that differs
Straight	5 cards of consecutive ranks	Top card in the sequence
Two pair	2 cards of the same rank with another two cards of the same rank and another (kicker)	By the highest pair and then by the second pair and then by the kicker
Pair	2 cards of the same rank and three unmatched cards	By the rank of the pair and then by the highest kicker that differs
No pair	5 unmatched cards	By the highest card that differs

Table 1: Ranking of hands in hold'em poker.

The list of features is very long. The aim in designing the features is to ensure that the relative performance of any two hands can be deduced only from the features. The exception is straight-flushes which are ignored. They are rare in hold'em .The probability of holding a straight flush is 0.00031. As it would also take a considerable number of heuristics to consider, they are ignored.

Straights and flushes are considered separately to the other classes. This is because the other classes form a natural hierarchy. The lower classes indicate the possibility of the higher classes with cards still to be revealed. For example, with one card to be revealed, a full-house is possible if the current holding is trips, less likely if the current holding is two pair and impossible with any other holding.

For straights and flushes though extra classes have to be added to consider potential hands (draws) as well as completed hands. Good players note which of the draws possible at an earlier stage were completed, so they can assess how likely it is that an opponent had that draw.

A separate set of features are designed for each stage. They are also divided into public (Board) and private (Holehand) features. The public features are retained at subsequent

stages. This is because the order in which the public cards are revealed provides useful information that can help infer a player's hidden cards. For example, if there are three cards of the same suit on board at the river so that a flush is possible it is important to remember when the flush became possible. The private features from previous stages are not retained as the information they provide are superseded by the new features.

The features are nested with most features only being relevant when a particular top-level feature is on.

The public features are used in two ways. Firstly the features for that stage and all previous stages are used to calculate the transitions matrix used by the observer model (the hidden Markov model). The private features for that stage are then added to the list of all public features up to that stage and used in the action selection function. The exception is on the river where a shorter list of 30 public features is used in the action selection function. This list is formed by suppressing features that relate to the order in which the board cards are revealed but is otherwise the same as the public features. While the number of features is large, the number of features that are on (non-zero) for any set of cards is usually less than ten because most of the features are nested.

	No. of Board Features	No of Holehand Features
Pre-Flop	0	6
Flop	19	71
Turn	22	95
River	38	82

Table 9-1: Number of features at every stage

Some formatting conventions are used to try make the list readable. Indenting is used to indicate levels of nesting; indented features are only calculated when the previous un-indented feature is non-zero. Italics are used for comments. Bold is used for the features derived from the public board cards. Where a particular board could be made in different ways, those features are bracketed. The subsequent private features are used if any of the bracketed combinations are on. The public features retained from previous stages are listed but without bullet points. The features relating to flushes and

those relating to straights are under separate headings.

Pre-flop
There are 6 features before the flop. Two of the features are used if the hidden cards are paired and the other 4 if they are unpaired.

- Pocket Pair
  - Rank of Pair
- *Else*
- Hi-rank
- Lo-rank
- Suited
- Straights - *no. of straights that include both cards*

Flop

- **Three of a kind flop**
  - **Rank of three-of-a-kind**
  - Quads

  Else
  - Pocket pair
    - Rank of pair
  - Hi-rank
  - Lo-rank

- **Pair flop**
  - **Rank of pair**
  - **Rank of kicker**
  - **Is pair higher than kicker?**
  - Quads
  - Full house with pocket pair
  - Full house with unpaired hand
  - Trips
    - Is unmatched hole card above pair?
    - Is unmatched hole card above kicker?
  - Pocket Pair
    - Rank of pair
    - Is pocket pair above board pair?
    - Is pocket pair above board kicker?
  - Paired board kicker
    - Rank of unmatched hole card
    - Is it above board pair?
    - Is it above board kicker?

  Else
  - Hi-rank

- Lo-rank
- No. of hole cards above board pair
- No. of hole cards above board kicker

- **No pair flop**
  - **Rank hi flop card**
  - **Rank mid flop card**
  - **Rank lo flop card**
  - Set (Trips made with a pocket pair)
    - Rank of set
    - Which flop card is matched?
  - Two pair
    - Hi-rank
    - Lo-rank
    - Which flop card is missing?
  - Pocket Pair
    - Rank
    - Rank compared to board (i.e. how board cards are higher?)
  - Pair
    - Rank of pair
    - Rank of pair compared to board
    - Rank of kicker
    - Rank of kicker compared to the board
  - No Pair
    - Hi-rank
    - Hi-rank compared to the board
    - Lo-rank
    - Lo-rank compared to the board

<u>Flushes and potential flushes</u>

- **All same suit**
  - Flush
    - Higher ranks not on board in flush suit- *if two players both have a flush, the winner is decided by the rank of the highest card of the flush suit. If a card is on the board it can't be an opponent's hand, so we don't have to worry about an opponent beating us with that card*.
  - 1-hole card flush draw
    - Higher ranks not on board in flush suit
- **2 same suit**
  - Flush Draw
    - Higher ranks not on board in flush suit
  - Back door 1 holecard flush draw
    - Higher ranks not on board in flush suit
  - Back door 2 holecard flush draw
    - Higher ranks not on board in flush suit
- **All different suits**
  - Back door flush draw

- Higher ranks not on board in flush suit

Straights and potential straights

A straight can be made with 2 hole cards and 3 board cards, which will be called a 2gap, or with 1hole card and 4 board cards, which will be called a 1 gap. Straight draws should also be considered at all stages before the river, these are hands for which one more card will complete a straight. A distinction is normally made between single draws (inside straight draws) and double draws (outside straight or double inside draws). A player has a single draw if only one rank would give him a straight. A player who will have a straight if either of two ranks is revealed on the next card has a double draw.

A backdoor draw (one that requires 2 board ranks) is unlikely. The probability of getting the two missing ranks is 0.0148. Because of this we would not expect it to be the dominant factor in making a decision but may tip the balance in a close decision. Many authors don't mention it at all. Where it is mentioned it is in relation to a semi-bluff play against very sophisticated opponents who would discount the possibility of anyone playing such a hand (usually in relation to no-limit but occasionally to limit). A count of backdoor draws is included though.

- **Number of 2gap straights**
  - Straight
    - Higher straights possible
  - 1gap draw
- **Single draws possible**
  - Single draw
- **Double draws possible**
  - Double draw

The next heuristics apply if I have a straight or a draw, they count how vulnerable I am to higher straights.

- Higher 1gap if I make a straight on next card (*add numbers for each draw, all if I already have straight)*
- Higher 2gap if I make a straight on next card (*add numbers for each draw, all if I already have straight)*

The next heuristics count how disguised my draw is; what other draws will connect with mine

- Shared with 1gap draw
- Shared with single draw

- Shared with double draw

- Backdoor draws
  - Higher if I connect
  - Shared with 1gap
  - Shared with single
  - Shared with double

If I don't have a draw I want to know what draws will connect if the board pairs one of my hole-

- Single draws if I pair a hole card *-add for both holecards*
- Double draws if I pair

Turn

Again the different types of boards are described and then nested within the types are more detailed heuristics. But now a distinction is made as to how the board was made; i.e. which card came on the turn. Where a particular board could be made from different combinations of flop cards and turn card, the same heuristics are used for the players own hand.

Quads board
(three of a kind flop
Rank of three-of-a-kind)
- **Quads board**
- Hi-rank

Trips board
[(three of a kind flop
Rank of three-of-a-kind)
- **Rank kicker**
- **Is kicker above three-of-a-kind**

Or
(Pair flop
Rank of pair
Rank of kicker
Is pair higher than kicker?)]
- **Turn card matches flop pair**
- Quads
- Full house (hole card matches kicker)
  - Is hand kicker above board kicker?
  - Rank hand kicker
- Pocket pair
  - Higher full (pair matches board kicker and the kicker is above the trips)

        - Rank of pair
    - Hi-rank
    - Lo-rank
        - No. of holecards above kicker

Two-pair board

[(Pair flop

Rank of pair

Rank of kicker

Is pair higher than kicker?)

- **Turn card makes 2-pair board**
- Quads
    - Higher possible
- Full house
    - Higher possible
- Pocket pair in play *-pair will only play if it's above one of the board pairs*
    - Rank of pair
    - Number of lower board pairs
- Pocket pair out of play
- Hi-rank
    - Number of lower board pairs
    - Board pairs lower than lo-rank

Pair Board

[(Pair flop

Rank of pair

Rank of kicker

Is pair higher than kicker?)

- **Rank turn card**
- **Turn card above pair?**
- **Turn card above kicker?**

Or

(No pair flop

Rank hi flop card

Rank mid flop card

Rank lo flop card)

- **Turn card matches board**
    - **Which flop card is matched?]**
- Quads
- Full house with pocket pair
- Full house with unpaired hand
    - Higher pocket pair full houses
    - Higher unpaired hole hand fullhouses
- Trips
    - Is unmatched hole card above board pair?
    - No. of board kickers below hand kicker?
- Pocket Pair
    - Rank of pair

- Is pocket pair above board pair?
- No. of board kickers below pocket pair?

- Paired board kicker
  - Is board pair below?
  - No of board kickers below?
  - Rank of unmatched hole card
  - Is it above board pair?
  - No of board kickers below unmatched holecard?

Else

- Hi-rank
- Lo-rank
- No. of hole cards above board pair
- No. of hole cards above board kickers (*count all 4 possible comparisons)*

No pair board

(No pair flop

Rank hi flop card
Rank mid flop card
Rank lo flop card)

- **Rank turn card**
- **No. of flop cards below turn card**
- Set (Trips made with a pocket pair)
  - Rank of set
  - Which board card is matched?
- Two pair
  - Hi-rank
  - Lo-rank
  - Board cards lower than hi-rank
  - Board cards lower than lo-rank
- Pocket Pair
  - Rank
  - Rank compared to board (i.e. how board cards are higher?)
- Pair
  - Rank of pair
  - Rank of pair compared to board
  - Rank of kicker
  - Rank of kicker compared to the board
- No Pair
  - Hi-rank
  - Hi-rank compared to the board
  - Lo-rank
  - Lo-rank compared to the board

<u>Flushes and potential flushes</u>

(Flop all same suit)

- **Turn card of same suit**

- Flush *-one card in hand of flush suit*
    - Higher ranks not on board in flush suit
    -

[(Flop all same suit)

- **Turn card of different suit**

or

(Flop 2 same suit)

- **Turn card matches two of same suit]**
- Flush
    - Higher ranks not on board in flush suit
- 1 holecard flush draw
    - Higher ranks not on board in flush suit

(Flop 2 same suit)

- **Turn card's suit matches single card on flop -** *there are 2 possible suits for a flush draw. A front door flush using two cards on the flop, and a backdoor flush draw using one card on the flop and the turn card*
- Frontdoor flush draw
    - Higher ranks not on board in flush suit
- Backdoor flush draw
    - Higher ranks not on board in flush suit

[(Flop 2 same suit)

- **Turn card's suit doesn't match any on the flop**

Or

(All different suits)

- **Turn card's suit matches one of the flop cards]**
- Flush draw
    - Higher ranks not on board in flush suit

(Flop all different suits)

- **Turn card is in fourth suit** *-flush not possible*

<u>Straights and potential straights</u>

(Number of 2gaps on flop
Single draws on flop
Double draws on flop)

- **1gaps**
- **Single draws completed by turn card**
- **Double draws completed by turn card**
- **New single draws -***these are backdoor draws*
- **New double draws**
- **Double draws that were single draws on flop**
    - Single draw
    - Double draw
- *The next heuristics apply if I have a straight or a draw, they count how vulnerable I am to higher straights*.
- Higher 1gap if I make a straight on next card (*add numbers for each draw, all if*

I already have straight)

- Higher 2gap if I make a straight on next card (*add numbers for each draw, all if I already have straight)*

The next heuristics count how disguised my draw is; what other draws will connect with mine

- Shared with 1gap draw
- Shared with flop single draw
- Shared with new single draw
- Shared with flop double draw
- Shared with double draws that were single draws on the flop
- *Shared with new double draws*

River

The same process is used as on the turn but the number of different ways a category can be formed is much larger.

Quads board
{[(three of a kind flop
Rank of three-of-a-kind)
Quads board]
- **Rank river card**

or
[(three of a kind flop
Rank of three-of-a-kind)
Rank kicker
Is kicker above three-of-a-kind)
or
(Pair flop
Rank of pair
Rank of kicker
Is pair higher than kicker?)
Turn card matches flop pair]
- **River card matches trips on board}**
- Highest of hole cards or board kicker *-if board kicker is higher than both hole cards*

Full House board
{[(three of a kind flop
Rank of three-of-a-kind)
Rank kicker
Is kicker above three-of-a-kind
Or
(Pair flop
Rank of pair
Rank of kicker
Is pair higher than kicker?)
Turn card matches flop pair)]

- **River card matches board kicker**

Or

[(Pair flop

Rank of pair

Rank of kicker

Is pair higher than kicker?)

Turn card makes 2-pair board]

- **River card matches board**
  - **Hi or lo pair}**
- Quads
  - Higher possible
- Higher Full house *-higher than one on board*

Trips board

{[(three of a kind flop

Rank of three-of-a-kind)

Rank kicker

Is kicker above three-of-a-kind

or

(Pair flop

Rank of pair

Rank of kicker

Is pair higher than kicker?)

Turn card matches flop pair]

- **Rank river card**
  - **Is it above board trips?**
  - **Is it above board kicker?**

or

[(Pair flop

Rank of pair

Rank of kicker

Is pair higher than kicker?)

Rank turn card

Turn card above pair?

Turn card above kicker?

Or

(No pair flop

Rank hi flop card

Rank mid flop card

Rank lo flop card)

Turn card matches board

Which flop card is matched?]

- **River card matches pair on board}**
- Quads
- Full house
  - Higher full-houses possible
- Hi-rank
- Lo-rank

2pair board

{[(Pair flop
Rank of pair
Rank of kicker
Is pair higher than kicker?)
Turn card makes 2-pair board]
Rank river card

Or

[(Pair flop
Rank of pair
Rank of kicker
Is pair higher than kicker?)
Rank turn card
Turn card above pair?
Turn card above kicker?

Or

(No pair flop
Rank hi flop card
Rank mid flop card
Rank lo flop card)
Turn card matches board
Which flop card is matched?]

- **River card matches a flop card**
- **River card matches the turn card}**
- Quads
  - Higher Quads possible
- Full house
  - Higher full-houses possible
- Pocket pair in play *-pair will only play if it's above one of the board pairs*
  - Rank of pair
  - Is above kicker
- Pair in play -*hole card matches board kicker*
- Hi-rank

Paired board

{[(Pair flop
Rank of pair
Rank of kicker
Is pair higher than kicker?)
Rank turn card
Turn card above pair?
Turn card above kicker?

Or

(No pair flop
Rank hi flop card
Rank mid flop card
Rank lo flop card)
Turn card matches board
Which flop card is matched?]
Rank river card

Is board pair higher?
How many lower board kickers?

Or

[(No pair flop
Rank hi flop card
Rank mid flop card
Rank lo flop card)
Rank turn card
No. of flop cards below turn card]

- **River card matches a flop card**
  - **Which one?**
- **River card matches the turn card}**
- Quads
- Full house with pocket pair
- Full house with unpaired hand
  - Higher pocket pair full houses
  - Higher unpaired hole hand fullhouses
- Trips
  - Is unmatched hole card above board pair?
  - No. of board kickers below hand kicker?
- Pocket Pair
  - Rank of pair
  - Is pocket pair above board pair?
  - No. of board kickers below pocket pair?
- Paired board kicker
  - Is board pair below?
  - No of board kickers below?
  - Rank of unmatched hole card
  - Is it above board pair?
  - No of board kickers below unmatched holecard?

Else

- Hi-rank
- Lo-rank
- No. of hole cards above board pair
- No. of board cards below hi hole card
- No. of board cards below lo hole card

Unpaired board

[(No pair flop
Rank hi flop card
Rank mid flop card
Rank lo flop card)
Rank turn card
No. of flop cards below turn card]

- **Rank river card**
- **How many board cards are higher**
- Set (Trips made with a pocket pair)
  - Rank of set

        - How many board cards are higher?
    - Two pair
        - Hi-rank
        - Lo-rank
        - Board cards lower than hi-rank
        - Board cards lower than lo-rank
    - Pocket Pair
        - Rank
        - How board cards are higher?
    - Pair
        - Rank of pair
        - Rank of pair compared to board
        - Rank of kicker
        - Rank of kicker compared to the board
    - No Pair
        - Hi-rank
        - Hi-rank compared to the board
        - Lo-rank
        - Lo-rank compared to the board

<u>Flushes</u>

Because there are no more cards to come there are no flush draws to consider on the river.

Flush board
[(Flop all same suit)
Turn card of same suit]
- **Flush board**
    - **Bottom rank on board -***cards above that rank in flush suit will play*
- Hole card in play in flush suit
- Higher ranks not on board in flush suit

Four flush board
{[(Flop all same suit)
Turn card of same suit]
or
[(Flop all same suit)
Turn card of different suit
or
(Flop 2 same suit)
Turn card matches two of same suit]
- **River card makes four of same suit}**
- Flush *-one card in hand of flush suit*

- Higher ranks not on board in flush suit

3 Flush board

- **{[(Flop all same suit)**
  - **Turn card of different suit**
- *or*
- **(Flop 2 same suit)**
  - **Turn card matches two of same suit]**
- or
- **[(Flop 2 same suit)**
- **Turn card's suit matches single card on flop]**
  - **Frontdoor flush connects**
  - **Backdoor flush connects**
- Or
- **[(All different suits)**
  - **Turn card's suit matches one of the flop cards]**
  - **River card makes flush draw}**
  - Flush
    - Higher ranks not on board in flush suit

Straights

Because there are no more cards to come there are no straight draws to consider on the river.

[(Number of 2gaps on flop
Single draws on flop
Double draws on flop)
1gaps
Single draws completed by turn card
Double draws completed by turn card
New single draws
New double draws
Double draws that were single draws on flop]

- **Straight board**
- **1gaps made from single draws completed by turn card**
- **1gaps made from double draws completed by turn card**
- **Single draws completed by river card**
- **Double draws completed by river card**
- **New Single draws completed by river card**
- **New Double draws completed by river card**
- **Single to Double draws completed by the river card**
- 1gap straight
- 2gap straight
  - Higher 1gap straights
  - Higher 2gap straights