

# Using Reinforcement Learning and Neural Network to Design Computer Player for Card Game

**Daniel Patanroi**

Com S 573 Spring 2005 Final Project  
Department of Computer Science  
Iowa State University  
226 Atanasoff Hall  
Ames, Iowa 50011-1041  
danpatan@cs.iastate.edu

## Abstract

This paper presents ChapsaRL, a program which learns to play Chapsa game online using reinforcement learning technique and artificial neural networks. Results have shown that ChapsaRL is a much more competitive computer player than random-behavior player.

## 1 Introduction

Works in reinforcement learning can be traced back to the early days of cybernetics, statistics, psychology, neuroscience, and computer science. For the last twenty years, it has attracted rapidly increasing interest in the machine learning and artificial intelligence communities. Its promise is beguiling - a way of programming agents by reward and punishment without needing to specify how the task is to be achieved [1]. However, there are formidable computational obstacles to fulfilling the promise.

Reinforcement learning is the problem faced by an agent that must learn behavior through trial and error interactions with a dynamic environment [1]. The work described here has a strong family resemblance to eponymous work in psychology, but differs considerable in the details and in the use of the word *reinforcement*. It is appropriately thought of as a class of problems, rather than as a set of techniques.

This paper presents ChapsaRL, which is an implementation of TD( $\lambda$ ) method using neural networks. One of the most successful application of TD( $\lambda$ ) learning method is to the game of Backgammon, using artificial neural network representations [8]. This paper describes the fundamental theories of reinforcement learning in section 2, explains the game of Chapsa in section 3, and layouts detail implementations of ChapsaRL in section 4. It also presents the results and discussion in section 5, followed by a brief summary and conclusions in section 6.

## 2 Fundamental Theories

### 2.1 Reinforcement Learning

What is reinforcement learning? The easiest way to understand reinforcement learning is by comparing it to supervised learning. In supervised learning, an agent is taught

how to respond to a given situation. In reinforcement learning, the agent is not taught how to behave rather it has a *free choice* in how to behave [2]. However, once it has taken its actions, it is then told if its actions were good or bad (this is called the reward – normally a positive or negative reward indicates good or bad behaviour, respectively) and has to learn from this how to behave in the future.

However, it is very rare that an agent is told directly after each action whether the action is good or bad. It is more usual for the agent to take several actions before receiving a reward [2]. This creates what is known as the *temporal credit assignment problem*, that is if our agent takes a series of actions before getting the reward, how can we take that reward and calculate which of the actions contributed the most towards getting that reward.

This problem is obviously one that humans have to deal with when learning to play card games. Although adults learn using reasoning and analysis, young children learn differently. If one considers a young child who tries to learn a card game, the child attempts to learn in much the same way as we want our agent to learn. The child plays the card game and rather than deducing that they have won because action  $X$  forced the opponent to make action  $Y$ , they learn that action  $X$  is correlated with winning and because of that they start to take that action more often. However, if it turns out that using action  $X$  results in them losing games, they stop using it. This style of learning is not just limited to board games but extends many activities that children have to learn (i.e. walking, sports, etc.) [2].

There are two main strategies for solving reinforcement learning problems. The first is to search in the space of behaviors in order to find one that performs well in the environment. This approach has been used in genetic algorithm, genetic programming, and novel search techniques. The second is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in the states of the world.

### 2.2 Temporal Difference Learning

If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be *temporal difference* (TD) learning. TD learning is a combination of Monte Carlo ideas and dynamic programming (DP) ideas. Like Monte Carlo methods, TD methods can learn directly

from raw experience without a model of the environment's dynamics. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome. The relationship between TD, DP, and Monte Carlo methods is a recurring theme in the theory of reinforcement learning [3].

Both TD and Monte Carlo methods use experience to solve the prediction problem. Given some experience following a policy, both methods update their estimate  $V$  of  $V^*$ . If a nonterminal state  $s_t$  is visited at time  $t$ , then both methods update their estimate  $V(s_t)$  based on what happens after that visit. Roughly speaking, Monte Carlo methods wait until the return following the visit is known, then use that return as a target for  $V(s_t)$ . A simple every-visit Monte Carlo method suitable for nonstationary environments is

$$V(s_t) \leftarrow V(s_t) + \alpha [R_t - V(s_t)]$$

where  $R_t$  is the actual return following time  $t$  and  $\alpha$  is a constant step size learning rate parameter.

Whereas Monte Carlo methods must wait until the end of the episode to determine the increment to  $V(s_t)$  (only then is  $R_t$  known), TD methods need wait only until the next time step. At time  $t+1$ , they immediately form a target and make a useful update using the observed reward  $r_{t+1}$  and the estimate  $V(s_{t+1})$ . The simplest TD method, known as TD(0), is

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

where  $\gamma$  is the discounted rate for reward.

### 2.3 Value Prediction with Function Approximator

Since the number of possible states in the game of Chapsa is very large, the approximate value function at time  $t$ ,  $V_t$ , is represented not as a table but as a parameterized functional form with parameter vector  $\vec{\theta}_t$ . This means that the value function  $V_t$  depends totally on  $\vec{\theta}_t$ , varying from time step to time step only as  $\vec{\theta}_t$  varies. For example,  $V_t$  might be the function computed by an artificial neural network, with  $\vec{\theta}_t$  the vector of connection weights. By adjusting the weights, any of a wide range of different function  $V_t$  can be implemented by the network.

One class of learning methods for function approximation in value prediction is based on gradient descent. Gradient descent methods are among the most widely used of all function approximation methods and are particularly well suited to reinforcement learning. In gradient-descent methods, the parameter vector is a column vector with a fixed number of real valued components,  $\vec{\theta}_t = (\theta_t(1), \theta_t(2), \dots, \theta_t(n))^T$  and  $V_t(s)$  is a smooth differentiable function of  $\vec{\theta}_t$  for all states  $s \in S$ .

We assume that states appear in examples with the same distribution, over which we are trying to minimize the mean squared error. A good strategy in this case is to try to minimize error on the observed examples. Gradient descent methods do this by adjusting the parameter vector after each example by a small amount in the direction that would most reduce the error on that example:

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha [v_t - V_t(s_t)] \nabla_{\vec{\theta}_t} V_t(s_t)$$

where  $\alpha$  is a positive step size learning rate parameter,  $v_t$  is an unbiased estimate of  $V^\pi(s_t)$  for each  $t$ , and  $\nabla_x f(x)$  is the gradient of  $f$  with respect to  $x$ .

The same approach can be applied in TD( $\lambda$ ) learning method. In the context of Markov chains, TD( $\lambda$ ) is identical to value iteration with the exception that TD( $\lambda$ ) updates the value of the current state based on a weighted combination of the values of future states, as opposed to using only the value of the immediate successor state. This changes the update rule to

$$\vec{\theta}_{t+1} = \vec{\theta}_t + \alpha \delta_t \vec{e}_t$$

where  $\delta_t$  is the usual TD error

$$\delta_t = r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t)$$

and  $\vec{e}_t$  is a column vector of eligibility traces, one for each component of  $\vec{\theta}_t$ , updated by

$$\vec{e}_t = \gamma \lambda e_{t-1} + \nabla_{\vec{\theta}_t} V_t(s_t)$$

with  $\vec{e}_0 = \vec{0}$ .

## 3 The Game: Chapsa

### 3.1 The Gameplay

Chapsa is a famous card game in the continent of Asia. Although it may have been named differently in many countries, but the basic rules for playing this game is fairly the same. The name "Chapsa" itself comes from a Chinese dialect, which literally means 13. This is because Chapsa, at best, is played by four people, each having 13 cards at the start of the game.

Chapsa uses a regular playing deck. This game can be played by 2-4 people. At the beginning of the game, all players start with the same amount of cards. A player with the smallest card plays first and the first hand that is played must contain the smallest card. After that, in a clockwise direction, the other players may play a hand, that must be higher than the previous hand played. The goal of the game is a race to finish playing all the cards. The game could be stopped either after one player plays all his cards or after only one player still has some cards unplayed.

### 3.2 The Rules

In the game of Chapsa, the suit is ordered as diamond (lowest), club, heart, spade (highest). Strangely, three is the smallest value and deuce is the highest value. That is, three of diamond or deuce of spade is the smallest or highest card in the whole deck, respectively.

There are 4 different hands: one-card hands, two-card hands, three-card hands, and five-card hands. Any single cards can be played as one-card hands. Any pair cards (two cards with the same value regardless of their suits) can be played as two-card hands. Similarly, a three-card hand is a three-of-a-kind, that is, three cards with the same value regardless of their suits. Five-card hands include regular poker hands. Straight hands (smallest), flush hands, full-house hands, four-of-a-kind hands, and straight-flush hands (highest) are all valid to be played as five-card hands. Notice that four cards of all suits with the same value must be played as five-card hands and not as four-card hands.

When playing pairs, three-of-a-kinds, flushes, full-houses, and four-of-a-kinds, one must remember the rule of "deuce is the highest value". The following statements are true

- Pair of deuce is higher than pair of ace.
- Three-of-a-kind of deuce is higher than three-of-a-kind of ace.
- Flush hand of heart with a deuce is higher than flush hand of heart with an ace and no deuce.
- Full-house hand with three cards of deuce is higher than full-house hand with three cards of ace.
- Four-of-a-kind of deuce is higher than four-of-a-kind of ace.

However, straight and straight-flush hands do not follow the rule of "deuce is the highest value". The followings are some examples

- Straight hands of ace, deuce, three, four, and five are NOT higher than straight hands of ten, jack, queen, king, and ace.
- Straight-flush hand of spade with deuce, three, four, five, and six are NOT higher than straight-flush hand of spade with ten, jack, queen, king, and ace.

The last rule that makes this game a little bit complicated is that one must play the same kind of hand with previous hand. If a player plays a pair hand, then the next player who wants to play cannot use any other hands than pairs. If a player does not have the same kind of hand or one has it but does not want to play it, then this player may pass. If a player  $A$  plays a hand  $H$ , and all other players pass consecutively, then  $A$  may play a hand whose type is different than  $H$  or a hand whose type is the same as  $H$  but whose value is smaller. The former is called a *respond move* while the latter is called an *opening move*.

### 3.3 Basic Strategies

Since the probability of getting three-of-a-kind, straight, flush, full-house, four-of-a-kind, or straight-flush hand is very small, it is more likely that players are playing single or pair hands most of the time. Because of that, if a player has more than one deuce, he usually plays them as single hands instead of others. Thus, if the current type of hands being played is one-card, a player can use a deuce to beat previous hand, and then after all other players pass (which can happen with higher probability), this player can play other type of hand of his choice.

Because different types of hands cannot beat one type of hand, a player usually want to keep the type of hands being played to a type that he has the most. A player may also want to take risk by playing a high-value hand in the beginning in order for him to be able to change the type of hands being played. For example, if the current type of hands being played is one-card, a player with deuce of heart (and without deuce of spade) may want to risk playing the deuce of heart. The owner of deuce of spade may pass for some reasons (i.e. if the next lower value of his cards after deuce of spade is a jack).

Having hands other than single is not usually better than not having them at all. For example, having a four, five, six, seven, and eight exactly one for each can be considered not good. Although those five cards can create a straight hand, if no player would initiate to play five-card hand, then those five cards must be played as single hands, which have low values. Even if a player initiates playing five-card hand, that straight hand would probably not able to beat any five-card hands that are played.

## 4 Implementation

To simplify the formula, the following notations are used

- Suit of diamond = D
- Suit of club = C
- Suit of heart = H
- Suit of spade = S

### 4.1 Ranking

In the game of Chapsa, to come out as the first winner, a player must have a good set of hands at the beginning of the game. Assuming the other players play competitively to come out as the first winner, a player with a bad set of hands will be very unlikely to be a first winner. Thus, a ranking system calculation is taken on every game at the beginning of a game.

**4.1.1 Scoring Single Hands.** There are 52 possible single hands, each is assigned a unique score ranging from 1 to 52. A sequence of [3D],[3C],[3H],[3S],...,[2H],[2S] is assigned values of 1, 2, 3, ..., 52. Let  $S$  be a set of cards and  $s_i \in S$  be the  $i$ -th card that is also a single hand. Single hands' score  $score_1$  of a set of cards  $S$  is defined as follow

$$score_1(S) = \frac{\sum_{s_i \in S} single(s_i)}{|S|}$$

where  $single(s_i)$  is a normalized score of a card. That is,  $single([3D]) = \frac{1}{52}$  and  $single([2H]) = \frac{51}{52}$ . Thus,  $score_1$  is basically an average of normalized scores for all cards in the set  $S$ .

**4.1.2 Scoring Pair Hands.** There are  $13 \times 6$  possible pair hands, each is assigned a unique score ranging from 1 to 78. For each value, there are 4 different suits, of which 6 combinations of pair hands can be created. Within a value, the combination is ordered as

1. diamond-club (smallest)
2. diamond-heart
3. club-heart
4. diamond-spade
5. club-spade
6. heart-spade (highest)

The smallest pair hand, [3D:3C], is assigned a score of 1, while the highest pair hand, [2H:2S], is assigned a score of 78. Let  $S$  be a set of cards,  $PairOf(S)$  be a set of pair hands can be found in  $S$ , and  $p_i \in PairOf(S)$  be the  $i$ -th

pair hand. Pair hands' score  $score_2$  of a set of cards  $S$  is defined as follow

$$score_2(S) = \frac{\sum_{p_i \in PairOf(S)} pair(p_i)}{|PairOf(S)|}$$

where  $pair(p_i)$  is a normalized score of a pair hand. That is,  $pair([3D : 3C]) = \frac{1}{78}$  and  $pair([2S : 2C]) = \frac{77}{78}$ . Thus,  $score_2$  is basically an average of normalized scores for all pair hands in the set  $S$ .

**4.1.3 Scoring Three-of-A-Kind Hands.** There are  $13 \times 4$  possible three-of-a-kind hands, each is assigned a unique score ranging from 1 to 52. For each value, there are 4 different suits, of which 4 combinations of three-of-a-kind hands can be created. Within a value, the combination is ordered as

1. diamond - club - heart (smallest)
2. diamond - club - spade
3. diamond - heart - spade
4. club - heart - spade (highest)

The smallest three-of-a-kind hand,  $[3D:3C:3H]$ , is assigned a score of 1, while the highest three-of-a-kind hand,  $[2C:2H:2S]$ , is assigned a score of 52. Let  $S$  be a set of cards,  $ThreeOf(S)$  be a set of three-of-a-kind hands can be found in  $S$ , and  $t_i \in ThreeOf(S)$  be the  $i$ -th three-of-a-kind hand. Three-of-a-kind hands' score  $score_3$  of a set of cards  $S$  is defined as follow

$$score_3(S) = \frac{\sum_{t_i \in ThreeOf(S)} three(t_i)}{|ThreeOf(S)|}$$

where  $three(t_i)$  is a normalized score of a three-of-a-kind hand. That is,  $three([3D : 3C : 3H]) = \frac{1}{52}$  and  $three([2D : 2H : 2S]) = \frac{51}{52}$ . Thus,  $score_3$  is basically an average of normalized score for all three-of-a-kind hands in the set  $S$ .

**4.1.4 Scoring Straight Hands.** The score of a straight hand is the sum of individual cards participating in the hand. Each card, except the aces, is assigned a unique value ranging from 4,5,6, to 51, for  $[2D],[2C],[2H]$ , to  $[KS]$ , respectively. All aces can have either one of the two values, depending whether they are being used as a low straight (A:2:3:4:5) or high straight (10:J:Q:K:A). As a low straight, aces are assigned values from 0 to 3. As a high straight, aces are assigned values from 52 to 55. The highest possible score in the set of all straight hands is achieved by the royal flush hand ( $[10S:JS:QS:KS:AS]$ ), which has a score of  $39 + 43 + 47 + 51 + 55 = 235$ . A normalized score for a straight hand  $H$  is

$$score_{str}(H) = \frac{\sum_{h \in H} val(h)}{235}$$

where  $h$  is a card in a straight hand  $H$  and  $val(h)$  is the individual score for card  $h$ , as being described above.

**4.1.5 Scoring Flush Hands.** The score of a flush hand is the sum of individual cards participating in the hand. Each card is assigned a unique value ranging from 0 to 51. A

sequence of  $[3D],[4D],[5D], \dots, [KD],[AD],[2D],[3C],[4C],[5C], \dots, [KS],[AS],[2S]$  is assigned values of 0, 1, 2, 3, ..., 51. The highest possible score in the set of all flush hands is achieved by  $[JS:QS:KS:AS:2S]$ , which has a score of  $47 + 48 + 49 + 50 + 51 = 245$ . A normalized score for a flush hand  $H$  is

$$score_{flush}(H) = \frac{\sum_{h \in H} val(h)}{245}$$

where  $h$  is a card in a flush hand  $H$  and  $val(h)$  is the individual score for card  $h$ , as being described above.

**4.1.6 Scoring Full-House Hands.** The score of a full-house hand is the sum of individual cards participating in the hand. Each card is assigned a unique value ranging from 0 to 51. A sequence of  $[3D],[3C],[3H], \dots, [AC],[AH],[AS],[2D],[2C], [2H],[2S]$  is assigned values of 0, 1, 2, 3, ..., 51. The highest possible score in the set of all full-house hands is achieved by  $[2C:2H:2S:AH:AS]$ , which has a score of  $49 + 50 + 51 + 46 + 47 = 243$ . A normalized score for a full-house hand  $H$  is

$$score_{fullhouse}(H) = \frac{\sum_{h \in H} val(h)}{243}$$

where  $h$  is a card in a full-house hand  $H$  and  $val(h)$  is the individual score for card  $h$ , as being described above.

**4.1.7 Scoring Four-of-A-Kind Hands.** The score of a four-of-a-kind hand is the sum of individual cards participating in the hand. Each card is assigned a unique value ranging from 0 to 51. A sequence of  $[3D],[3C],[3H], \dots, [AC],[AH],[AS],[2D],[2C], [2H],[2S]$  is assigned values of 0, 1, 2, 3, ..., 51. The highest possible score in the set of all four-of-a-kind hands is achieved by  $[2D:2C:2H:2S:AS]$ , which has a score of  $47 + 48 + 49 + 50 + 51 = 245$ . A normalized score for a four-of-a-kind hand  $H$  is

$$score_{fourkind}(H) = \frac{\sum_{h \in H} val(h)}{245}$$

where  $h$  is a card in a four-of-a-kind hand  $H$  and  $val(h)$  is the individual score for card  $h$ , as being described above.

**4.1.8 Scoring Straight-Flush Hands.** The score of a straight-flush hand is the sum of individual cards participating in the hand. Each card, except the aces, is assigned a unique value. A sequence of  $[2D],[3D],[4D], \dots, [KD]$  is assigned values of 1 to 12. A sequence of  $[2C],[3C],[4C], \dots, [KC]$  is assigned values of 15 to 26. A sequence of  $[2H],[3H],[4H], \dots, [KH]$  is assigned values of 29 to 40. A sequence of  $[2S],[3S],[4S], \dots, [KS]$  is assigned values of 43 to 54. All aces can have either one of the two values, depending whether they are being used as a low straight-flush (A:2:3:4:5) or high straight-flush (10:J:Q:K:A). A sequence of  $[AD],[AC],[AH],[AS]$  is assigned values of 0,14,28,42 if they are used in a low straight-flush. A sequence of  $[AD],[AC],[AH],[AS]$  is assigned values of 13,27,41,55 if they are used in a high straight-flush. The highest possible score in the set of all straight hands is achieved by the royal flush hand ( $[10S:JS:QS:KS:AS]$ ),

which has a score of  $51 + 52 + 53 + 54 + 55 = 265$ . A normalized score for a straight-flush hand  $H$  is

$$score_{strflush}(H) = \frac{\sum_{h \in H} val(h)}{265}$$

where  $h$  is a card in a straight-flush hand  $H$  and  $val(h)$  is the individual score for card  $h$ , as being described above.

**4.1.9 Scoring Poker Hands.** Since all poker hands are considered the same type of play, the score of each poker hand must be adjusted according to how easy or hard to get it. For example, a normalized score of 50% for full-house hand is actually higher than a normalized score of 50% for straight hand. Total number of each poker hand is calculated as follow

- $10 \times 4^5 = 10240$  straight hands
- $4 \times \binom{13}{5} = 5148$  flush hands
- $13 \times \binom{4}{3} \times 12 \times \binom{4}{2} = 3744$  full-house hands
- $13 \times 48 = 624$  four-of-a-kind hands
- $10 \times 4 = 40$  straight-flush hands

The ratio 10240:5148:3744:624:40 can be simplified to 2560:1287:936:156:10. Thus, the normalized scores for straight, flush, full-house, four-of-a-kind, and straight-flush hands must be multiplied by 1,  $\frac{2560}{1287} \approx 1.989$ ,  $\frac{2560}{936} \approx 2.735$ ,  $\frac{2560}{156} \approx 16.41$ , and  $\frac{2560}{10} = 256$ , respectively. Thus, poker hands' score  $score_5$  of a set of cards  $S$  is defined as follow

$$\begin{aligned} score_{5SUM}(S) &= \sum_{h \in Str(S)} score_{str}(h) / |Str(S)| \\ &+ 1.989 \times \sum_{h \in Flu(S)} score_{flush}(h) / |Flu(S)| \\ &+ 2.735 \times \sum_{h \in FH(S)} score_{fullhouse}(h) / |FH(S)| \\ &+ 16.410 \times \sum_{h \in FK(S)} score_{fourkind}(h) / |FK(S)| \\ &+ 256 \times \sum_{h \in SF(S)} score_{strflush}(h) / |SF(S)| \end{aligned}$$

$$score_5(S) = score_{5SUM}(S) / 278.134$$

where  $Str(S)$ ,  $Flu(S)$ ,  $FH(S)$ ,  $FK(S)$ , and  $SF(S)$  return a set of all straight, flush, full-house, four-of-a-kind, and straight-flush hands from a set of cards  $S$ , respectively.

**4.1.10 Scoring Set of Cards.** Since the goal of Chapsa is to play all cards as fast as possible, five-card hand with 50% normalized score should worth more than pair hand with 50% normalized score. Given a set of cards  $S$ , the general score for set  $S$  is defined as follow

$$score_{SUM}(S) = score_1(S) + 2 \times score_2(S) + 3 \times score_3(S) + 5 \times score_5(S)$$

$$score(S) = \frac{score_{SUM}(S)}{11}$$

## 4.2 Algorithm

ChapsaRL implements TD( $\lambda$ ) learning method with a linear neural network served as function approximator for its value

function  $V^\pi$ . The approximation value for  $V_t$  is defined as

$$V_t(s_t) = \sum_{i=1}^n f_i \theta_t(i)$$

where  $f_i$  is a feature component in the neural network for a state  $s_t$  at time  $t$ ,  $n$  is total number of features representing a state, and  $\theta_t(i)$  is a weight component in the neural network related to feature  $i$  at time  $t$ .

The gradient of  $V_t(s_t)$  with respect to  $\vec{\theta}_t$  is defined as follow

$$\nabla_{\vec{\theta}_t} V_t(s_t) = \left( \frac{\partial V_t(s_t)}{\partial \theta_t(1)}, \frac{\partial V_t(s_t)}{\partial \theta_t(2)}, \dots, \frac{\partial V_t(s_t)}{\partial \theta_t(n)} \right)$$

$$\nabla_{\vec{\theta}_t} V_t(s_t) = (f_1, f_2, \dots, f_n)$$

A complete algorithm for online gradient descent TD( $\lambda$ ) that is used by ChapsaRL is as follow

```

01 initialize  $\vec{\theta}$  arbitrary
02 repeat for each game
03    $\vec{e} = 0$ 
04    $s \leftarrow$  initial state of a game
05   repeat for each step of a game
06      $a \leftarrow$  action given by  $\pi$  for  $s$ 
07     take action  $a$ , observe its reward  $r$ , and next state  $s'$ 
08      $\delta \leftarrow r + \gamma V(s') - V(s)$ 
09      $\vec{e} \leftarrow \gamma \lambda \vec{e} + \vec{f}_s$ 
10      $\vec{\theta} \leftarrow \vec{\theta} + \alpha \delta \vec{e}$ 
11      $s \leftarrow s'$ 
12 until  $s$  is the last move on a game

```

Note that  $\vec{f}_s$  is a vector of features extracted from state  $s$ .

## 4.3 Selected Features

In the game of Chapsa, decisions can be categorized into two: one for choosing opening moves and the other for choosing respond moves. ChapsaRL implements these two types of decisions as a separate two neural networks, each using a different set of features. Features that are used in opening-move-neural-network are as follow

1. Proposed action's value
2. Type of hand
3. Total current cards
4. Total played cards
5. Average score of current single hands
6. Average score of current pair hands
7. Average score of current three-of-a-kind hands
8. Average score of poker hands

Features for respond-move-neural-network are as follow

1. Propose action's value
2. Total current cards
3. Total played cards
4. Average score of hands of the same type with the proposed action

#### 4.4 Reward Calculation

Rewards is given at the end of every game. A player with initial card rank  $x$  is given a positive reward  $r$ , if it can finish the game with a position  $y \leq x$ , where  $r = x - y + 1$ . On the other hand, a player is given a negative reward  $r$ , if it finishes the game with a position  $y > x$ , where  $r = x - y$ .

### 5 Results and Discussions

All the results are run using  $\lambda = 0.9$ , discounted delay rate  $\gamma = 0.5$ , and learning rate  $\alpha = 0.5$ .

#### 5.1 Comparing Random and ChapsaRL Players

Table 1 shows the percentage of winnings for a random player versus three other random players, separated by initial card ranks. The rows represent initial card ranks while the columns represent winner position. For example, given a first rank cards, random players are able to come out as the first winner 31.1% of the time. Here, random player is a player who decides its moves randomly using a random number generator.

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	31.1	25.0	22.9	21.0
Rank 2	22.8	25.4	26.1	25.7
Rank 3	22.5	24.9	26.2	26.4
Rank 4	22.5	24.0	26.3	27.3

Table 1: Winning percentage for random player versus three other random players over 10000 games.

Table 2 shows the percentage of winnings for a ChapsaRL player, trained with three other random players for 100 games, versus three other random players over 10000 games. Compared to Table 1, it can be seen that the performance of ChapsaRL players trained with random players is higher than that of random players. The percentage of 1st and 2nd winnings increase while the percentage of 3rd and 4th winnings decrease in all ranks. This shows that reinforcement learning implemented in ChapsaRL player is working.

#### 5.2 Varying Amount of Training for ChapsaRL Players

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	51.7	25.2	14.7	8.4
Rank 2	35.9	27.9	20.7	15.5
Rank 3	36.0	27.2	20.7	16.0
Rank 4	35.4	27.2	20.9	16.4

Table 2: Winning percentage for ChapsaRL, trained with three random players for 100 games, versus three other random players over 10000 games.

Table 3 and 4 shows the performance of ChapsaRL players in the same situation as in Table 2, except that the amount of training is increased to 1000 and 10000 games, respectively. Comparing the values on Table 2,3, and 4, it can be seen that as the amount of training increases, the percent wins as first winner increases. All other values seem

to go up or down without specific pattern.

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	53.2	24.2	14.7	7.9
Rank 2	36.0	27.7	20.6	15.8
Rank 3	36.1	26.9	20.9	16.1
Rank 4	35.6	27.7	20.4	16.4

Table 3: Winning percentage for ChapsaRL, trained with three random players for 1000 games, versus three other random players over 10000 games.

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	53.6	25.1	13.8	7.6
Rank 2	36.6	27.1	20.9	15.4
Rank 3	36.6	26.7	21.0	15.8
Rank 4	36.6	27.7	19.8	16.0

Table 4: Winning percentage for ChapsaRL, trained with three random players for 10000 games, versus three other random players over 10000 games.

#### 5.3 Comparing Different Trainers

The performance of ChapsaRL player in Table 2,3, and 4 is based on a training with three random players. Table 5 shows the performance of ChapsaRL player trained with three other ChapsaRL players, starting from "baby" level. Comparing Table 4 and 5, it can be seen that the performance of ChapsaRL player trained with three other "baby" ChapsaRL players is not as good as that of ChapsaRL player trained with three random players.

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	52.8	24.9	14.1	8.2
Rank 2	34.9	27.4	21.6	16.1
Rank 3	35.6	26.5	21.9	15.9
Rank 4	36.1	27.4	20.5	16.0

Table 5: Winning percentage for ChapsaRL, trained with three ChapsaRL players for 10000 games, versus three random players over 10000 games.

Table 6 shows the performance of ChapsaRL player trained with three other "adult" ChapsaRL players. Comparing Table 4 and 6, it can be seen that the performance of ChapsaRL player trained with three other "adult" ChapsaRL player is better than that of ChapsaRL player trained with three random players.

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	53.4	24.8	14.2	7.6
Rank 2	38.1	26.9	19.8	15.2
Rank 3	37.4	26.9	21.2	14.5
Rank 4	37.7	26.0	20.8	15.6

Table 6: Winning percentage for ChapsaRL, trained with three adult ChapsaRL players for 10000 games, versus three random players over 10000 games.

## 5.4 Comparing Different Opponents

So far ChapsaRL player is only tested against random players. Table 7 shows the performance of ChapsaRL player against three copies of itself over 10000 games. Comparing Table 5 and 7, it can be seen that if ChapsaRL player is setup to play with more intelligent players, then its performance decreases. This shows that random players are not as competitive as ChapsaRL players.

	1st Win	2nd Win	3rd Win	4th Win
Rank 1	47.6	25.1	17.1	10.2
Rank 2	17.0	25.6	28.0	29.5
Rank 3	17.2	24.3	28.3	30.2
Rank 4	15.9	25.6	28.2	30.3

Table 7: Winning percentage for ChapsaRL, trained with three ChapsaRL players for 10000 games, versus three copies of itself over 10000 games.

## 5.5 Varying Different Number of Opponents

Here, ChapsaRL player is tested in one-on-one match with other player. Table 8 shows its performance against its own copy. Table 9 shows its performance against a random player. These two tables confirm once again that ChapsaRL player is indeed much more competitive than random player.

	Win	Lose
Rank 1	68.5	31.5
Rank 2	32.5	67.5

Table 8: Winning percentage for ChapsaRL, trained with three ChapsaRL players for 10000 games, versus a copy of itself over 10000 games.

	Win	Lose
Rank 1	82.4	17.6
Rank 2	68.7	31.3

Table 9: Winning percentage for ChapsaRL, trained with three ChapsaRL players for 10000 games, versus a random player over 10000 games.

## 6 Conclusion and Summary

Based on the results and discussions, ChapsaRL player is known to be more competitive than random player. It is also shown that amount of training and the types of trainers will affect the performance of ChapsaRL player. Moreover, in a head to head match, ChapsaRL seems to perform better.

## 7 References

- [1] Kaelbling, L. P. and Moore, A. W. **Reinforcement Learning: A Survey.** Journal of Artificial Intelligence Research 4 (1996) 237 - 285.
- [2] Ghory, I. **Reinforcement Learning in Board Games.** Project Report. May 4, 2004.
- [3] Sutton, R. S. and Barto, A. G. **Reinforcement Learning: An Introduction.** MIT Press, Cambridge, MA,

1998.

[4] Thrun, S. **Learning To Play the Game of Chess.** Advances in Neural Information Processing System 7. 1995.

[5] Barto, A. G. and Mahadevan, S. **Recent Advances in Hierarchical Reinforcement Learning.**

[6] Even-Dar, E. and Mansour, Y. **Convergence of Optimistic and Incremental Q-Learning.** NIPS 2001.

[7] Plett, G. L. **Punish/ Reward: Learning With a Critic in Multilayer Neural Networks.**

[8] Tesauro, G. J. **Practical Issues in Temporal Difference Learning.** Machine Learning, 8, 1992.