City University of Hong Kong
Department of Computer Science

BSCS Final Year Project Report 2003-2004

03CS057

Computer Go

(Volume 1 of 1)

Student Name : Wong Wai Hung

Student No. : 50249864

Programme Code : BSCCS

For Official Use Only

Supervisor : Chun, H W Andy

1st Reader : Yu, Y T

2nd Reader : Chow, K O

Abstract

Generally we can divide a Go game into three stages: beginning game, middle game, and end game; and there is one sub problem that could appear in all three stages of the game – tsumego (Life and Death).

Lift and Death problems concern about a group of stone can archive a life state, that are safe from being capture, by a specific sequence of moves or, in the counter part, to look for a sequence of play to kill opponent's stone no matter how she/he defense it. All Go players do practice on the Life and Death problem as it comes up very often in game. Also there are books written by professional players where Life and Death questions are ranked into different levels. Those questions usually display an enclosed group of stone in a corner or on the side of a board, which exist one correct sequence of play to archive a firm life or a firm kill[Appendix 1]. For this kind of enclosed group which cannot escape and outside condition are ignored, we call it closed Life and Death.

This project conducted two experiments on using ANN to recognize stones status enclosed in corner. The result is impressive and there maybe ways we can apply ANN not only to recognize the stone status but also to find the correct sequence of play. We believe that by creating a special partial connection between input layer and the first hidden layer can obtain more accurate result. With very similar shapes of life and dead patterns providing in our experiment, ANN showed its competent in distinguishing which is alive and which is dead. Also the performance of applying ANN to evaluate life status should have performance advantage over heuristic search.

Acknowledgment

I would like to give my warmest thanks to my supervisor, Dr. Andy Chun. During the whole process on doing this final year project, Dr. Chun gave me a lot of support and flexibility. Without his support and encouragement I am sure I will face more difficulties.

Besides, I also want to thank you the FANN library development team from the Department of Computer Science University of Copenhagen (DIKU), especially the project admin. Steffen Nissen. The FANN is free, open source and in the LGPL license. Without this library, I would not be able to spend such a long time on research materials. Also Steffen Nissen's prompt response in the sourceforge forum and helped me in manually connecting neurons with the FANN library.

Last by not least, I would like to thank you all my classmates, friends and certainly my family who gave me supports. Also thank you to Mr. Poon from 香港圍棋網 who borrowed me a book on life and death[14].

Table of Content

1. Introduction

Computer Go posed a great challenge to computer scientist. By using heuristic, we have archived great success in Chess and many other board games. However, there are a lot of different between Go and Chess, which make our classical heuristic method could not work in Go.

Features	Chess	Go
Board Size	8 x 8 squares	19 x 19 grids
Moves per game	~80	~300
Branching factor	Small (~35)	Large (~200)
Human lookahead	Typically up to 10 moves	Even beginner read up to 60 moves (deep but narrow search, e.g. ladder)
Evaluation of board position	Good correlation with number and quality or pieces on board	Poor correlation with number of stone on board and territory enclosed

Table 1.1

From the table 1.1 cited from a paper on comparing the difference on Go and Chess[1]. The large branching factor, deep lookahead, and difficult board position evaluation makes heuristic impossible as the tree growth quickly and we are hard to perform cut off. Up till now, no one has come up a solution or even a good way to program the computer to play or learn Go. However many people think that patterns and shapes is a key point of the game and the first program that can play a complete game of Go is also from a pattern recognition research. My project inherits this idea to focus on pattern and applied Artificial Neural Network (ANN) to perform static shape recognition to tackle the Life and Death problem in Go.

1.1 The chronology

Albert Zobrist created the first program that can play a complete game of Go in 1968 for his Ph.D [2]. The approach Zorbrist used is to look for patterns at each available location on board. There are a score for each pattern and the value can be negative if that pattern represented an unfavourable situation. The score will be summed up for each matched pattern and the highest scored position will be played. Although they got features and patterns to match against the board position, but it is unlikely they got some specific patterns for solving Life and Death.

Zobrist's creation is a landmark to programming computer Go. After that, Ryder did some improvement on Zobrist's program but idea is pretty much the same.

Next we can find the LISP program by Walter Reitman and Bruce Wilcox. In the Reitman-Wilcox program, they created a tactician. The task of the tactician is to find the right answer for the program, such as: can a string[Appendix 2] being capture?, can a stone being saved?. Their tactician will perform a lookahead by attempting to simulate reasoning. The tactician is limited to explore a reasonable numbers of variations and returns for an answer. If a few well-selected variations comes up with the same answer, the answer will probably correct and the tactician will stop and return the answer. This program is rated about 15-20kyu [Appendix 3] in the mid 1970s.

Since the first creation in the 1968, the progress in computer go is pretty slow. Now still no computer program can archive at amateur 1 dan strength. Also most of the Go programs we can get are proprietary and close source. Generally people still following the classical heuristic ways and keep adding more hard craft knowledge. This make the progress of computer Go to advance slowly and that is why this project tested the ability of ANN in solving the Life and Death problem instead of using heuristic.

1.2 Objectives

The objectives of doing this project are to learn more on computer game playing and machine learning. By experimenting ANN in recognizing alive and death patterns resided in corner to evaluate the possibility to use ANN in computer Go.

1.3 Scope of the project

At first I was targeting on a larger scope and want to create my own tactician and plug it into GNUGo. I studied quite some papers on other peoples Go program and the GNUGo document. However I found with the limited time this large objective seems infeasible to archive. Finally I come to this scope in tackling the Life and Death problem. Now the scope become:

- Study the techniques used in computer game playing and machine learning.
- Study related works on tackling the Life and Death problem in Go
- Study and design on the structure of ANN
- Evaluate the feasibility in applying ANN on the Life and Death problem on Go

1.4 Problem domain

Apply ANN to recognize life stone in final position, dead stone from real game, dead stone from book, and incomplete life shapes from book resided within an 8 x 8 corner with.

2. Background Information

2.1 Pattern Matching in Go

Pattern matching is very important in Go. Human expects see shapes and patterns of stones on board. Strong player and play with several weak players simultaneously by just looking on the shape and do a fast lookahead.

Quite some effort has been put on the exact matching algorithm in Go. Usually they got a hand crafted pattern database and use the matching algorithm to match against the database for features recognition and perform lookahead.

There is an isomorphic characteristic for patterns in Go. On Go board, a single pattern can appear in different orientation and swapped color of stones. Therefore, a single pattern can appear as 16 isomorphic patterns shown in the figure follow.

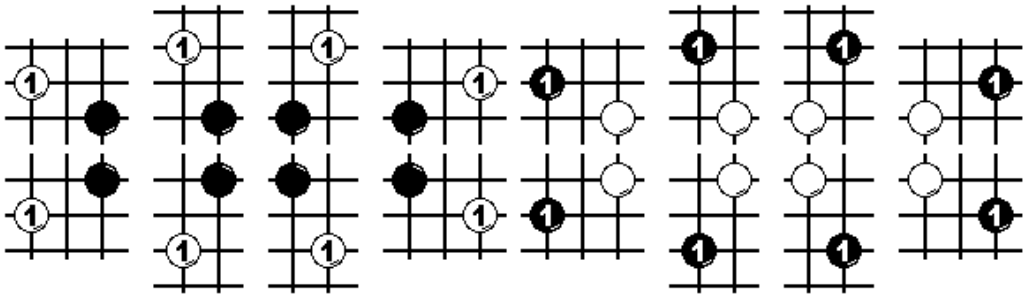


Fig. 2.1

Generally, a subset of pattern from the database will be match against the whole board and pick up some selected location for further matching for the whole set, but some may apply the whole set of pattern to the whole board position at once. Each turn a numerous of pattern matching have to be performed.

If we use a straightforward approach, we will have to perform matching on every location (19x19 = 361) and one pattern for 16 isomorphic. If there are M patterns and

cost to match one pattern is N, the total cost becomes: 361 x 16 x M x N. That is a lot of work for each turn and we have to do it quick.

2.1.1 Pattern matcher of Goliath [3]

Goliath is a Go-playing program, written by Mark Boon. In Goliath, it use integers as bitmap to store patterns and perform matching.

One integer contain 32-bit, for patterns within a 5 x 5 size (25 points), it can be represented by 3 32-bit integers. Each of the three integers represents the white stones, black stones and empty locations in the 5 x 5 pattern. It will covert the 2D patterns into a 1D array, but there is not details about it. The three integers will be used as a bitmap to store the stones location and empty spaces. Also for each single pattern we will have to rotate it to obtain the 8 isomorphic patterns.

After all the patterns being built into a set of three integers, board position are converted into 1D array in the same manner as the patterns. To compare the board position with a pattern, only three bit operations is required. Such as: if (BOARD-POSITION & PATTERN) == PATTERN. Computers are fast in doing bit-operations and it is found that we will get 99% mismatch if we compare the empty space first. Furthermore, matching can only performed around the location with new stone added but not on whole board. With all these fine tunes, we can match the board position with larger pattern database in a faster way.

To tackle the color swapped patterns, we can use the BLACK-PATTERN to match against WHITE-BOARD-POSITION and vice versa.

2.1.2 Deterministic Finite state Automaton (DFA)

DFA is the pattern matching algorithm that used in GNUGo. This algorithm is proposed by Tanguy Urvoy and GNU Go team [4] and it is powerful.

In DFA, 2D patterns will be converted into an 1D array.

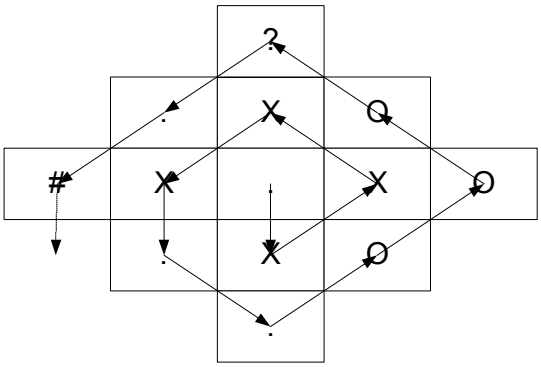


Fig 2.2

O	White
X	Black
.	Empty
?	Don't care
#	Don't care (can be out of board)
	Scanning Path

Fig. 2.2 showed how a pattern is converted into 1D array. To transform the 2D patterns, it start at the center of the pattern, like fig. 2.2, then follow the spinning scanning path we will obtain: .XXX.OOO?.#.

In DFA, every pattern will be converted in the way shown in fig. 2.2. Then a state diagram will be constructed according to all the 1D arrays obtained from converting the 2D patterns. After the state diagram is constructed, board positions are converted in the same way. The converted board position will pass though this state diagram and lookup what patterns did match with it.

For example there are only fig. 2.3 and fig. 2.4. in our pattern set. By performing the 2D to 1D conversion, we will obtain: .OXXXO?.?.?.? for fig. 2.3 and .OXX.O?.?.?X? for fig. 2.4.

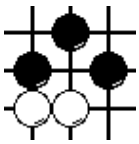


Fig. 2.3

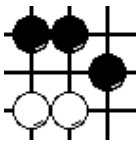


Fig. 2.4

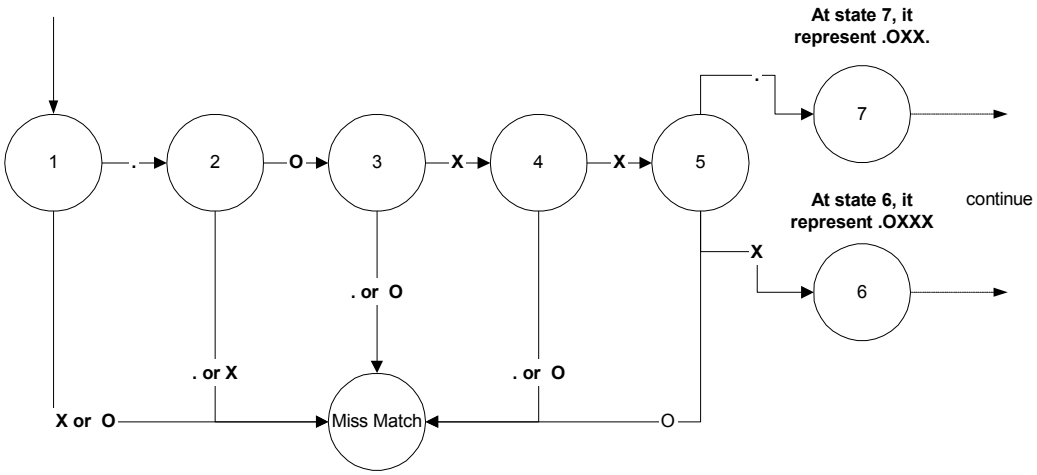


Fig. 2.5

Fig. 2.5 showed the beginning part of the state diagram constructed from fig. 2.3 and fig 2.4. For patterns that share the same prefix, there will be a common path for that prefix like state 1 to 5 in the figure. When a board position reached at state 5, it can go to state 7 to look up patterns with prefix .OXX. or go to state 6 for patterns start with .OXXX. No matter how many patterns we have, with the state diagram constructed, only one passes will be required to match against all the patterns.

This DFA matching algorithm is powerful. It reuse the previous result and easy to extend. For the pattern matching method in Goliath, it matches against fig 2.3 first and then matches with fig 2.4 for a board position. However in the state diagram, board positions are matched with the patterns location by location. If a board position reached state 6, we know that fig. 2.4 will never be matched. Effort was saved. Also duplicated patterns can be detected during the construction of the state diagram, even the pattern database growth.

2.1.3 Conclude

Although this project is not doing anything about exact matching, but the way other people formulate the matching problem in Go, like DFA, is interesting and worth to learn about. Also the isomorphic characteristic of patterns in Go is important.

2.2 Artificial Neural Networks

Definition:

The construction of Artificial Neural Network (ANN) is inspired by biological learning mechanism that learn by building up a complex web of interconnected neurons. ANNs model this biological behavior by creating massive paralleled processing unit with interconnection defined by a weight. ANNs provide a way for computer to learn by external stimulus and store information by adjusting the weights.

Brief History:

For the first attempt on building artificial neural network, we can trace back to 1943 by Warren McCulloch and Walter Pitts. They created a network of artificial neuron that, in theory, could compute any kind of logical functions. Then in the late 1950s, Frank Rosenblatt has invented the perceptron network with an associated learning rule. Almost about the same time, Bernard Widrow and Ted Hoff created another system called the Adative Linear Element network (ADALINE) and a new learning algorithm. However, the training algorithm using at that time cannot be used to train muti-layered complex network. In 1960s, peoples think that further research on the ANN will lead to a dead end. The development of ANN is suspended for a decade and become active again in the 1970s. The Hopfield network and the Adaptive Resonance Theory (ART) network was evolve at that time. Another key development was in 1986 by David Rumelhart and James McClelland. They created the backpropagation training algorithm for multilayered network. Since the first attempts in 1943, it takes more than 40 years for the ANN technology to become mature. Today, many of the algorithms we take for granted requires more than 40 years struggle by the pioneers. I am here to thank you them all.

Applications of Artificial Neural Networks:

Generally speaking, problems with the following characteristics are suitable for ANN.

- Instances can be represented by attribute vectors
- Target function output can be discrete-valued, real-valued, or a vector of several real- of discrete-valued attributes
- Long training time is acceptable
- Quick evaluation is required

Today ANN is widely used because of the robust training algorithm. It has been applied from securities for fingerprint recognition, to board games as human opponent, to medial for breast cancer cell analysis…etc.

Many successes have been found in applying the ANN in programming computers to learn or evolve its own way to play board game. David B. Fogel and Kumar Chellapilla made the first attempts to apply evolutionary algorithm on evolving ANN to play checker and archived expert levels[5]. TD-leaning ANN have been found success in backgammon too[6].

General Information:

ANNs are constructed by layers of interconnecting neurons. There will be a weight defining a connection between neurons. Initially, weights and some parameters in the ANN are generated randomly. The ANN will learn by the external stimulus (inputs). Weights are adjusted according to the learning rule and information is stored in it. For large ANNs, it may come up with hundreds or thousands of inter-neuron-connections. So manual adjusting the inter-neuron-connections weights is quite infeasible.

Neuron:

Artificial neurons are the fundamental units in an ANN. Fig 2.6 shown a very basic single-input neuron. It composed by a weight, a summer, and a transfer function. The scalar input p is multiplied by the weight w to form wp and the constant input 1 is multiplied by the bias b. These two value will be summed and output n to the transfer function or the output function where f(wp + b) = f(n) = a. w and b are scalar parameters of the neuron which we can adjust. The transfer function can be linear or nonlinear function but we have to determine it before training and use a corresponding learning algorithm.

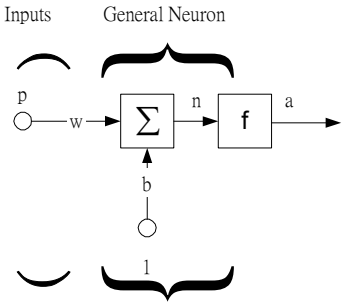


Fig. 2.6

Fig. 2.7 shown a multiple-input neuron. Each input will be multiplied with the associated weight and being summed up in the summer.

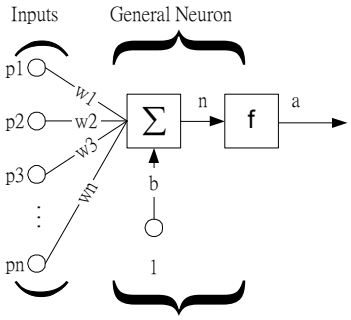


Fig. 2.7

$$n = p_1w_1 + p_2w_2 + p_3w_3 + \ldots + p_nw_n + b \quad (2.1)$$

Where equation (2.1) can be seen as product of a vector ($\vec{p}$) to a single row matrix (W):

$$n = \begin{pmatrix} p1 \\ p2 \\ p3 \\ ... \\ pn \end{pmatrix} \begin{pmatrix} w1 & w2 & w3 & ... & wn \end{pmatrix} + b = \vec{p}W + b \quad (2.2)$$

Similarly:

$$a = f(\vec{p}W + b) \quad (2.3)$$

By using the matrix and vector representation, the equations become less bulky for the multiple-input neuron. The ANN using in this project is composed by the multiple-input neuron

For recurrent networks like the Hopfield Network and Hamming Network, they got another component called delay block. And some outputs are feedback as inputs which is quite different to the feedforward network used in this project.

Learning Rules:

ANNs learnt by adjusting the inter-neuron-connection weight. The learning rules, or some referred as training algorithm, are the procedure to modify the weights and the biases of a network. In general we classify the learning rules into three categories:

1. Supervised Learning

   To use the supervised learning rule, we have to provide a set of training samples of proper network behavior. We provide a set of inputs ($\vec{p}_1, \vec{p}_2, \vec{p}_3 ... \vec{p}_n$) and our target output ($\vec{t}_1, \vec{t}_2, \vec{t}_3 ... \vec{t}_n$) for each training input. The network outputs and the target outputs are compared. The learning rule

will be used to adjust the weights and biases to shift the network output closer to the target.

2. Unsupervised Learning

   For Unsupervised learning, weights and biases are adjusted in responses to the input only and do not have target output.

3. Reinforcement Learning

   Reinforcement learning is similar to supervised learning, but only grades are given to the output of the network

The learning rule used in this project is supervised learning.

Perceptron network

One of the common learning rules is known as perceptron rule. The perceptron rule falls into the class of supervised learning and is able to train ANN with only single layer of neuron. We call ANN that only contains one layer of neuron as perceptron network. The simplest form of perceptron network is compose of input and a single neuron like fig. 2.7

The perceptron learning rule states that: $\Delta W_i = \alpha * (t - o) * X_i$

where α is the learning rate determined by heuristic, $(t - o)$ is the different between target t and the network output o and X_i is the input signal.

Perceptron learning rule only work for single layer ANN and it can only compute linear separable functions, like Boolean function AND and OR, but fail for XOR. Because of this fatal deficiency in the perceptron learning rule, we have another rule to solve this problem, the delta rule.

The delta rule is based on gradient descent to search the problem space to minimize the error and find the best fitting weights for the training dataset. Generally it used to train network that do not contain threshold transfer function. To train the network with delta rule, we need to find the error E defined as $E = \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$. D is the

whole set of train data where d is an instance of D. t_d is the target output of instance d and o_d is the network output of instance d.

The delta rule states that: $\Delta W_i = -\alpha \frac{dE}{dW_i}$ (2.4)

where α is the learning rate and E is the error. By substituting the E and do the differentiation:

$$\frac{dE}{dW_i} = \frac{d}{dW_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$\frac{dE}{dW_i} = \sum_{d\in D}(t_d - o_d)(-X_{id}) \quad (2.5)$$

Substitute (2.5) into (2.4) we obtain:

$$\Delta W_i = \alpha \sum_{d\in D}(t_d - o_d)X_{id} \quad (2.6)$$

The delta rule is very important, as it is the base of the backpropagation algorithm, which is widely used in training multilayer networks, and the delta rule is able to converge for nonlinear separable function.

Backpropagation and Multilayer Perceptron Network

When multilayer ANN is used, another learning rule called backpropagation rule is used. Fig. 2.8 shown a sample of multilayer perceptron network.

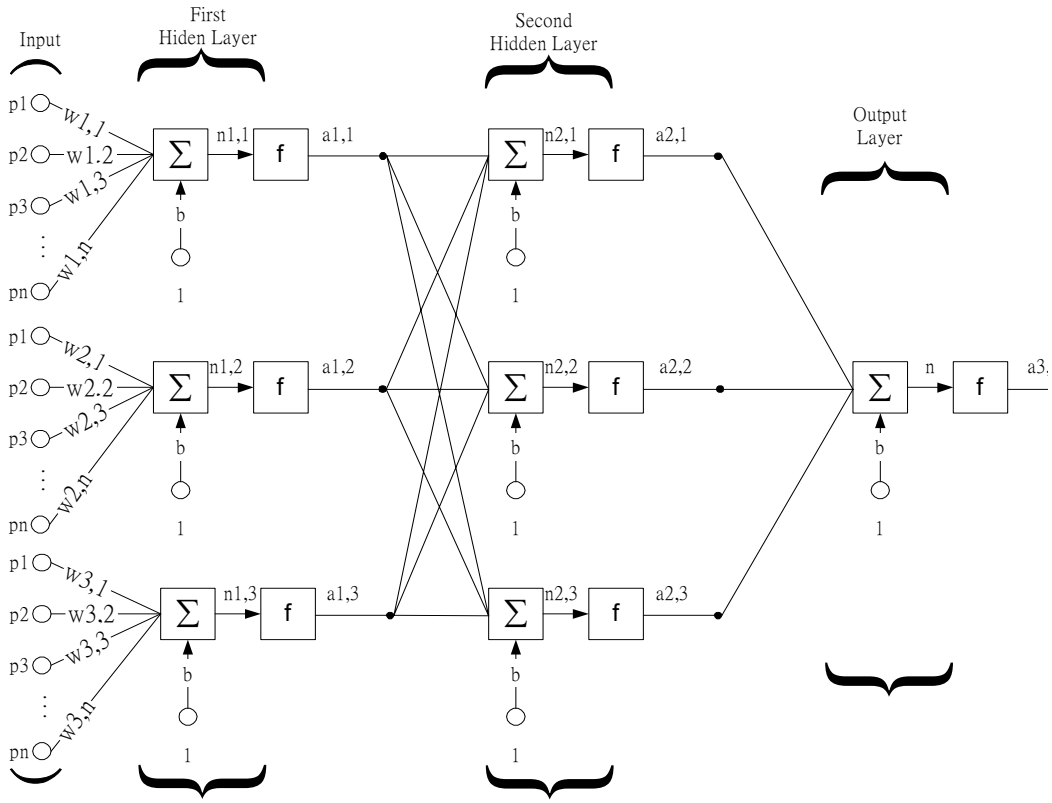


Fig. 2.8

The backpropagation rule is developed base on the delta rule. It calculate the overall error and then back propagate the error from the output neuron back to the hidden neurons by calculating the contribution of error of a particular neuron in the previous layer. For the delta rule it calculate error after the whole set for training at once. However in the backpropagation rule, the error is calculated for each training cases.

$\Delta W_{ji} = -\alpha \frac{dE_d}{dW_{ji}}$ where E_d is the error of training cases d.

The error E_d is defined as: $E_d(\vec{W}) \equiv \frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$ where output is the set of output neurons.

$$\Delta W_{ji} = -\alpha \frac{dE_d}{dW_{ji}} = -\alpha \frac{dE_d}{dnet_j}\frac{dnet_j}{dW_{ji}} = -\alpha \frac{dE_d}{dnet_j} X_{ji} \tag{2.6}$$

where $net_j = \sum_i W_{ji} X_{ji}$ (the weighted sum of inputs for unit j)

The training rule for the output units' weight:

$$\frac{dE_d}{dnet_j} = \frac{dE_d}{do_j}\frac{do_j}{dnet_j} \tag{2.7}$$

$$\frac{dE_d}{do_j} = \frac{d}{do_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2 = -(t_j - o_j) \tag{2.8}$$

For the transfer function, we take it as sigmoid function. So $o_j = \sigma(net_j)$

$$\frac{do_j}{dnet_j} = \frac{d\sigma(net_j)}{dnet_j} = o_j(1 - o_j) \tag{2.9}$$

Substitute the results we have back into equation (2.6)

$$\Delta W_{ji} = \alpha(t_j - o_j) o_j (1 - o_j) X_{ji} \tag{2.10}$$

Equation (2.10) is the learning rule for the output neurons.

The training rule for the hidden units' weight:

$$\frac{dE_d}{dnet_j} = \sum_{k \in Downstream(j)} \frac{dE_d}{dnet_k}\frac{dnet_k}{dnet_j}$$

where Downstream(j) is the set of all units immediately downstream of unit j.

We denote $\frac{dE_d}{dnet_k}$ by $-\delta_k$ and $\frac{dE_d}{dnetj}$ by $-\delta_j$

$$\frac{dE_d}{dnet_j} = \sum_{k \in Downstream(j)} \frac{dE_d}{dnet_k}\frac{dnet_k}{dnet_j}$$

$$-\delta_j = \sum_{k \in Downstream(j)} -\delta_k \frac{dnet_k}{do_j}\frac{do_j}{dnet_j}$$

$$\delta_j = \sum_{k \in Downstream(j)} \delta_k W_{kj} o_j (1 - o_j)$$

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k W_{kj}$$

Hence:

The training rule for the hidden units' weight is: $\Delta W_{ji} = \alpha \delta_j X_{ji}$

3 Related Works

After searching for quite a long time, very few efforts dedicated to the life and death problem was found. Although there quite some go playing programs which should developed its own way in tackling this problem, but most of those go playing programs are proprietary and closed source. We could hardly know the underlying algorithm and technique used in it. The only open source go program with complete document is the GNUGo[7]. Besides, a closed source program called GoTools[8], created by Thomas Wolf, is an application that can play the life and death problem with human and response to the human's move. Finally, I found a chapter from a Mphil thesis[9] is about applying ANN on the life and death.

3.1 GNUGo

General

GNUGo is one of the open source project from GNU, thank you to Richard M. Stallman who created the wave of open source and the abundant of open source software from GNU. GNUGo is a complete go playing program that able to play a complete game of Go. It contains different modules to tackle the different part of the game. There is a joseki database; the play in fuseki and end game is different. More importantly to this project, in the GNUGo document it revealed the way how GNUGo deal with the life and death reading.

Life and Death Reading

According to the GNUGo document, they use two module to do the life and death reading. The first module perform lookaheads till a stage for the next module can decide that position is alive or not. They call the two module OWL (Optics with Limited Negotiation) and optics respectively.

In defining the life and death reading, they think the defender will try to expand for eyespace while attacker will try to limit it, this is the meaning of Optics With Limited Negotiation. Until neither side find effective moves on scoping the eyespace, that position will be passed to optics to evaluate it. The OWL code perform lookahead depends on two pattern database. One is attack pattern and one is defense pattern.

The optics can determine the life status quite accurate if the position is in a terminal position, which no moves left which can expand the eyespace. Beside optics, they got a third pattern database that could override the eyespace analysis by optics. The third database defines patterns about shortage of liberties [Appendix 4] and cutting points which optics will overlook.

As we can see, the approach GNUGo use is depend on the hand craft pattern databases. The problem with hand crafted knowledge is difficult to maintain when the database growth. Also it completely rely on human knowledge and it is always require a lot of effort to capture human knowledge and covert it into a program to teach the computer. There are no information about the strength of the OWL and optics module. However the overall strength of GNUGo version 3.2 obtained a 6k* from an Internet Go Server called Legend Go Server (LGS) [Appendix 5].

3.2 GoTools

GoTools is a proprietary software that can solve Life and Death problem and play against with human in Life and Death problem. By input a life and death problem to it, it can response to the move made by human.

From the publications[10][11] by Thomas Wolf, it reveals that GoTools used a heuristic approach. In the homepage of GoTools, it state that a Nihon Kiin[Appendix 6] amateur 6 dan player rated GoTools as a 5 dan (in solving life and death). Also according to IntelligentGo.org [12], it claimed that GoTools can solve high dan-level tsumego problems but, still, no actual data and performance have been shown.

There is a Java applet version of GoTools and a test was carried out. Two 1k questions, one 4k question and five 5k questions take from a tsumego book written by Japan professional player[13] were input to test this Java applet version of GoTools. All these questions are closed Life and Death. There are five level of the applet version. When I select the medium speed search, it got the two questions correct, one time out and all other missed. For the other two slower search methods (perform fewer cut off in the search tree), it almost time out in all questions and only returned wrong answer for one question. However this is a test on the Java applet version of GoTools, does not directly reflect the real strength of the actual program.

3.3 ANN for recognizing Final Position

To the best of my knowledge, the only resource on applying ANN to the problem of life and death is in a chapter of a Mphil thesis. Horace Chan, Wai-Kit on his report of "Application Of Temporal Difference Learning And Supervised Learning In The Game Of Go" did an experiment on applying ANN to recognize alive stones and dead stones in final game position within a 8x8 corner.

Horace's experiment – Target and Result

The focusing target of Horace's experiment is on life group and dead group in final position from real games played by amateur players. He obtains games from Internet Go Servers like NNGS, IGS, and LGS. Then life groups and dead groups resided in an 8 x 8 size corner are extracted as input and test case. He separated the samples into two categories, one is direct sample from the game another is human selection. The following table is quote from the original source.

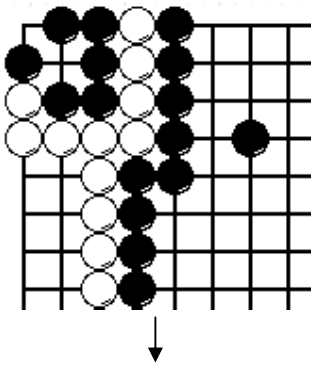
No. of training examples	Type of training sample	Correct Percentage
1391	Direct game input	95.87%
969	Human selection	96.90%
1495	Human selection	98.45%

Table 3.1

The ANN was tested against 387 patterns which is separate from the training dataset. These 387 patterns were selected by Horace himself. He suggests that the increase in training sample will increase the accuracy.

Horace's experiment – ANN design

In Horace's design, for each 8 x 8 size corner input of original board map, it will generate another 8 x 8 location map.



0	1	1	-1	1	0	0	0		0	1	1	0	0	0	0	0
1	0	1	-1	1	0	0	0		1	0	1	0	0	0	0	0
-1	1	1	-1	1	0	0	0		0	1	1	0	0	0	0	0
-1	-1	-1	-1	1	0	1	0		0	0	0	0	0	0	0	0
0	0	-1	1	1	0	0	0		0	0	0	0	0	0	0	0
0	0	-1	1	0	0	0	0		0	0	0	0	0	0	0	0
0	0	-1	1	0	0	0	0		0	0	0	0	0	0	0	0
0	0	-1	1	0	0	0	0		0	0	0	0	0	0	0	0

Board map Location map

Fig. 3.1

So the ANN will have 128 inputs for an 8x8 board (64 points).

There is a special partial connectivity between the 128 inputs and the first hidden layer. Horace divide the 8 x 8 board in to a series of subsections. Fig. 3.2 shown two sample subsection of 4x4 and 3x3 over a 8x8 corner board. For a board with 8x8 size, there will be 36 different 3x3 subsection, 25 4x4 subsection, 16 5x5 subsection and so on.

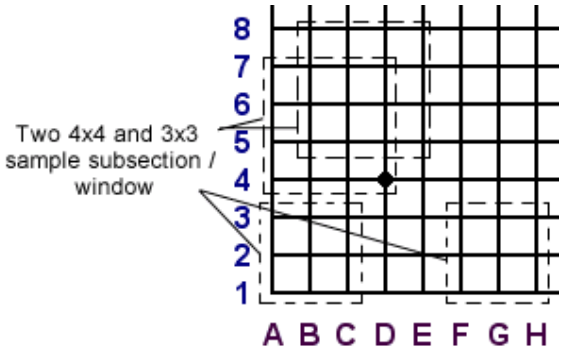


Fig. 3.2

Table 3.2 summarized the number of subsection of different size. From the table 3.2 below, the connection to input is multiplied by 2 because of the extra 64 input of the location map. Each neuron in the first hidden layer representing a subsection will connect to the corresponding inputs on both maps. Therefore neurons representing subsections contain connections double on its size.

Subsection size	Number of subsections	Connection to the input neuron
3 x 3	36	9 x 2 = 18
4 x 4	25	16 x 2 = 32
5 x 5	16	25 x 2 = 50
6 x 6	9	36 x 2 = 72
7 x 7	4	49 x 2 = 98

Table 3.2

Hence there will be 36+25+16+9+4 = 90 neurons in the first hidden layer in representing subsections. Besides these 90 neurons, in the first hidden layer there are another 60 neurons fully connected to the inputs. According to Horace, these 60 neurons are responsible to capture special shapes. Fig. 3.3 summarized the partial connection design between the input and the first hidden layer of Horace's ANN used in his experiment. The complete architecture of this ANN is quite large. It is in 128-150-70-40-30-2.

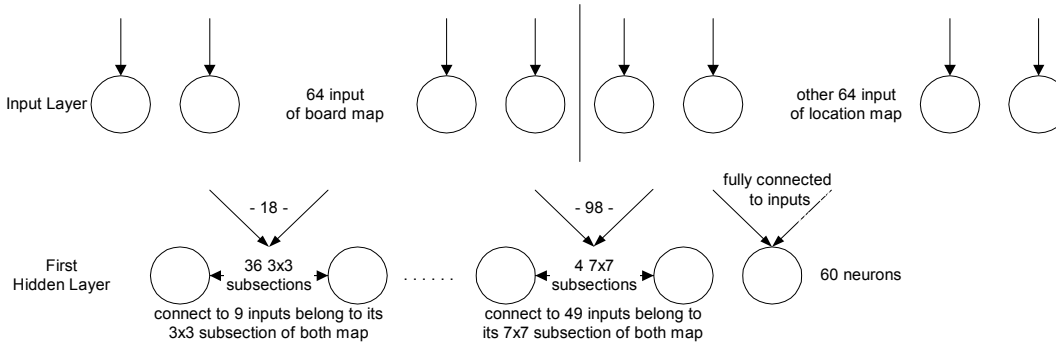


Fig 3.3

This ANN performed very well on evaluating life status of corner stones in final positions. However, this tricky design of the extra 64 input of the location map got a fatal problem in our experiment two and will discussed together with my design later on.

4 Experiments

Two experiments were conducted in this project. The first one is a repetition of Horace's experiment with a different ANN design, while the second one is the extension of the first experiment. For both experiment, a fully connected ANN and a partial connected ANN were used and their performance is compared.

4.1 Experiment One

For every Go game, which played till the end without either side resigned the game, both players will have to determine the life status for every group of stone on board. Dead stones will be removed and left those life groups on board. Then the score is counted. It is the same for the games played on the Internet. So when we open the games that reached the end and have a score, the life and death information for every group of stone on board, which is agreed by both player, will be left. In this experiment, we rely on the games with final score and life and death information to extract patterns that resided within an 8 x 8 size corner. This type of pattern being extracted from final position game, we define it as life shape in final position and dead shape in final position. The ANN in this experiment is trained and tested against life shape in final position, dead shape in final position and dead shape from book.

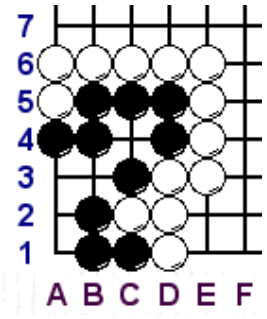


Fig. 4.1 Life stones in final position

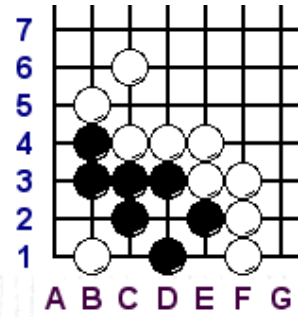


Fig. 4.2 Dead stone

Fig. 4.1 and fig 4.2 shown a life shape in final position and a dead shape. For dead shape, in fig 4.2, it means no matter black play on which location, white will always be able to counter it. The dead shapes in games are quite similar to dead shape from

book. When player recognized the corner stones are dead, they will not add any extra move on it. This is quite different for life groups like fig 4.1, both side will play till a very closed contact form in order to reduce opponent's enclosed points. So dead shape from real game will leave in loosely contact while life group will in close contact with opponent. Beside the dead shape in final position, some of the dead shape are extracted from books. The books used are listed in reference[13][14], and both book were written by professional 9 dan players which is the highest professional ranking.

4.2 Experiment Two

In the second experiment, new sample and test case is added. Life patterns not in final position is added to the data pool in experiment one. Those patterns are captured from the same life and death books listed in experiment one. All patterns used are below the level (easier than) 3 kyu strength according to the ranking of the questions from the book.

Fig 4.3 shown a Life stone not in final position. The black stone remains in loose contact with opponents which is quite similar to the dead shape in fig. 4.2. However in fig. 4.3, no matter where white put his stone, black always can counter it and keeping the whole group alive. With this added samples into training and test. The boundary between life shape and dead shape become more ambiguous.

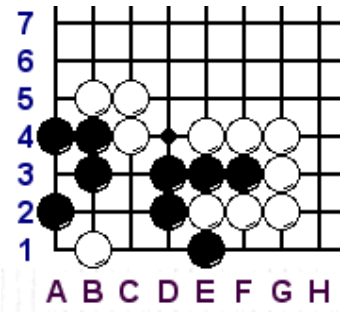


Fig. 4.3 Life stone not in final position

Obtaining good result seems is easier for the first experiment, but here it is a doubt on how accurate can ANN get in this experiment

For both experiment we used an ordinary fully connected ANN and a specially designed ANN, we call it 2D Go board ANN, containing partial connection between the inputs and the first hidden layer.

5 Sample Specification

As mentioned in previous chapter, corner patterns with alive and dead shapes with in an 8 x 8 size corner is extracted from games played by amateur players in the Internet and extracted from books. There is the specification on how the patterns are extracted.

5.1 From Internet

The LGS server and the client LGC

There are some famous Internet Go Servers can be found easily from search engine, like IGS, LGS and NNGS. Many of the Go servers are based on the NNGS source code. Although LGS and IGS did developed its own client to connect to their Go server but the core of the Go server were still the same. The server uses the Go Modem Protocol or the Go Text Protocol. Clients that able to speak those protocols can be used to connect to the server and play games on it. The patterns captured from game were all from LGS.

There is a client tool that is dedicated for LGS is called the LGC. LGC is rich in function and allow user to view other players' game history and result. As the resigned games are not useful to this project, this function can sieve out those unwanted game easily.

Also, there is a preview function for a finished game. Fig. 5.1 shown the screen captured from LGS of browsing the game history of a 5d* player fed. For the first 7 games listed, the games were ended by resignation or by forfeit of time. Those game will be ignored. The 8th listed game: feb 5d* played with ACDaee 2d* and won by 30.5 points. This is the targeted type of game.

		勝方	棋力	敗方	棋力	授子	貼目	棋盤	結果	對局日期
查看	寄回	(白)feb	5段*	(黑)aliex	1段*	授4子	0.5	19路	白中盤勝	2004年03月20日 13時34分
查看	寄回	(白)feb	5段*	(黑)eskk	3段	授0子	0.5	19路	白中盤勝	2004年03月20日 12時26分
查看	寄回	(黑)lingofour	4段*	(白)feb	5段*	授0子	0.5	19路	白超時負	2004年03月20日 12時04分
查看	寄回	(白)feb	5段*	(黑)Stanelys	2段*	授3子	0.5	19路	白中盤勝	2004年03月20日 07時16分
查看	寄回	(白)feb	5段*	(黑)Stanelys	2段*	授3子	0.5	19路	白中盤勝	2004年03月20日 06時36分
查看	寄回	(白)feb	5段*	(黑)toog	3段*	授2子	0.5	19路	白中盤勝	2004年03月20日 03時03分
查看	寄回	(白)feb	5段*	(黑)toog	3段*	授2子	0.5	19路	白中盤勝	2004年03月20日 01時46分
查看	寄回	(白)feb	5段*	(黑)ACDaee	2段*	授3子	0.5	19路	白勝30.5目	2004年03月19日 20時46分
查看	寄回	(黑)dogone	4段	(白)feb	5段*	授0子	0.5	19路	黑中盤勝	2004年03月19日 20時09分
查看	寄回	(黑)wumegan	4段*	(白)feb	5段*	授0子	0.5	19路	黑勝4.5目	2004年03月19日 19時16分
查看	寄回	(黑)feedom	3段*	(白)feb	5段*	授2子	0.5	19路	黑中盤勝	2004年03月19日 18時24分
查看	寄回	(白)feb	5段*	(黑)aronton	2段*	授3子	0.5	19路	白勝5.5目	2004年03月19日 17時41分

Fig. 5.1

By clicking the button at the left, the actual game will show up. If there are no suitable patterns that a group of stone being enclosed in 8 x 8 corner, that game will be ignored and only those game contain life or dead patterns within a 8 x 8 corner will be downloaded. Fig 5.2 shown the game of fed 5d* played with ACDaee 2d*. However, there is no information about which groups or which stones were dead and need to be removed. The only detail is white win by 30.5 point with 0.5 komi[Appendix 7] shown at the menu bar of the window. So the actual different in points is 30.5 – 0.5 = 30. The next step is to use the scoring function that LGC provide. Which will help user to count the points.

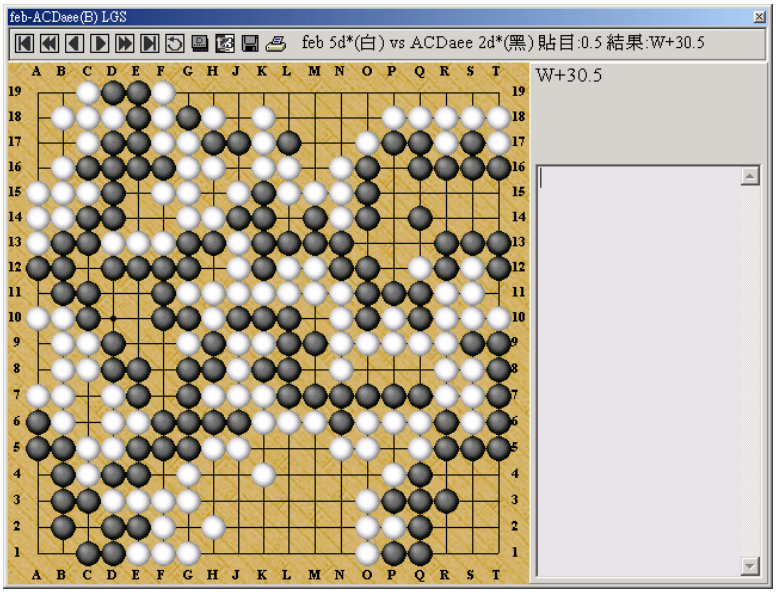


Fig. 5.2

Fig 5.3 is the initial scoring window. It counted white has 45 points and black has 19 points. The red dots in the scoring means neither side own that particular point. White circle means that's white color territory and black circles for the black territory. This dose not agrees with the game result which white won by 30.5 points with 0.5 komi. By manually deciding the dead stones, it will update the score. Fig. 5.4 is the result by removing dead stone by myself. It now counted white has 75 points and Black has 45 points and agrees with the game result now. This is a manually process, but I am an amateur 2k* player in the LGS and being the judge for several Go competition organized by 香港圍棋網 [Appendix 8]. Determining dead stone at final position is quite a handy job to me.

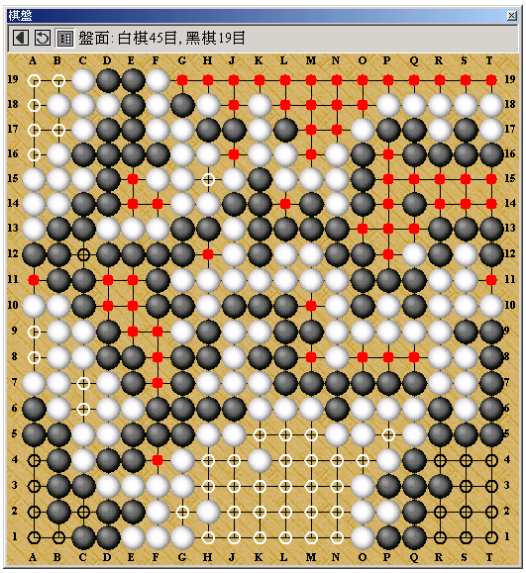


Fig 5.3

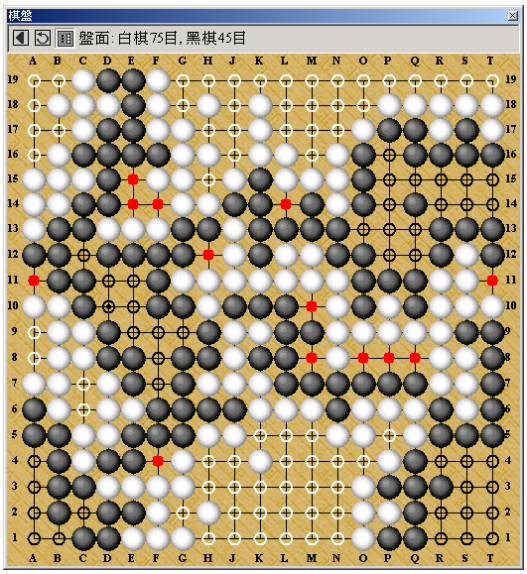


Fig 5.4

After getting the correct result from the scoring window, the important thing lefts are the top left corner and the two bottom corners. Then this game will be downloaded and the patterns are being extracted with noise data removed.

Another advantage of extracting game form the LGS server is the ranking in LGS is quite accurate. It is generally one to two kyu stronger than Hong Kong ranking. That means a 2k* in LGS probably has 1 kyu to 1 dan strength in Hong Kong. If only games played by 1d* or above is being used, there are unlikely to be any kyu players. Because the end game position require both play to agree on the life status for every

group of stone on board, weak player may make mistakes on it while dan players are almost impossible to miss when the board comes to its final position.

The process

After the scored games with suitable patterns were found and downloaded, the complete game record contains a lot unwanted information to the experiment. To eliminate it, the useful patterns were extracted from the complete game record and re-input into another software manually to save it down as the SGF format. SGF is the standard go game record format. The game played by fed 5d* and ACDaee 2d* is now become the figure below.

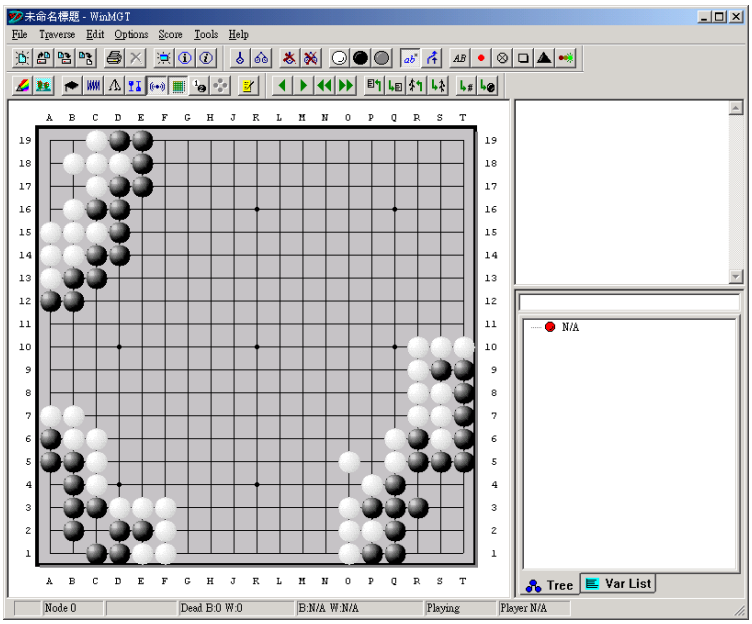


Fig. 5.5

There is a problem for the bottom right corner, as the size of the pattern exceeded the 8 x 8 size. A process of manual collapse will be performed.

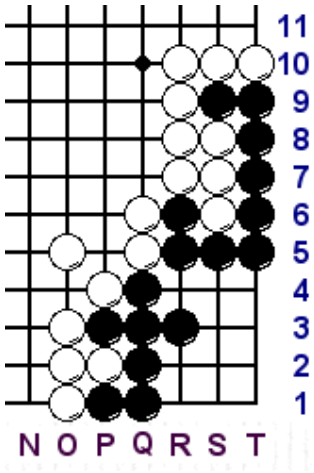


Fig. 5.6 Pattern before collapses

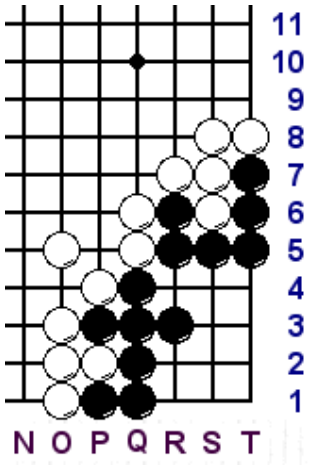
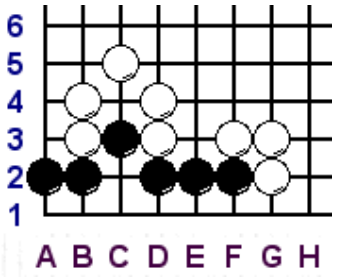


Fig. 5.7 Pattern after collapses

From fig. 5.6 and fig. 5.7 you can see black stone in T8, T9 and S9 will not affect the life state and these stone make the pattern exceed the 8x8 corner size. Patterns in fig. 5.6 will be transformed into fig. 5.7 to meet our requirement. Stones can be collapsed only if the surrounding 8 immediate positions were all filled by black stone, white stone, margin or empty space connected to outside board. So stone Q4 will never be removed, as R4 is an empty space not in outside board. After the useful patterns were extracted and collapsed into an 8 x 8 size corner, it will be saved and named accordingly. Life pattern and dead pattern were saved into separate file and named by different prefix.

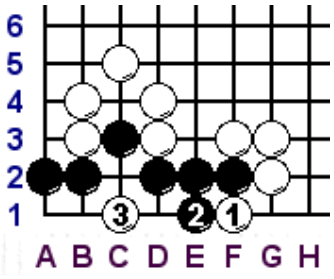
5.2 From Book

For patterns extract from the tsumego book, another kind of transformation will be performed. Each question in the book will show, at least, one correct sequence of play and one wrong sequence of play.



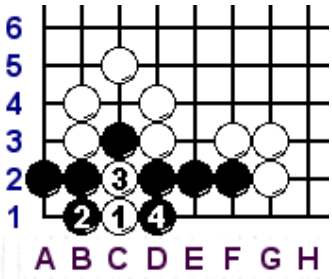
Question

Fig. 5.8



Correct Play to Kill

Fig. 5.9



Miss Play Failed to Kill

Fig. 5.10

Fig. 5.8 shown a question, which is white play first to kill black. Fig. 5.9 is the solution and fig. 5.10 is a sample of wrong answer. The question itself (fig. 5.8) will not be converted as a pattern for training or test case because it is ambiguous. In fig. 5.8, if it is white's turn, corner can be killed like fig. 5.9, while if it is black's turn, black can play on either B1, F1, C1 or C2 to archive a life state. The question itself dose not show any evidence on who will play first, hence it is ambiguous and will not be captured as samples for the experiment. For Answers, like in fig. 5.9, a snap shot on each odd number moves will be made and each snap shot is stored as an individual pattern. Fig. 5.9 will be converted into two patterns shown in fig. 5.11 and fig. 5.12.

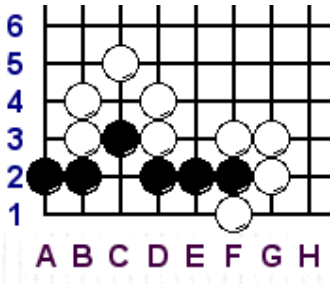


Fig. 5.11

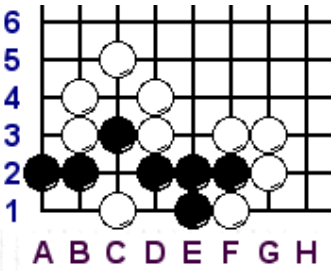


Fig. 5.12

While for the samples of wrong answers, snap shot will be made on the even number moves and each snap shot is stored as an individual pattern. Fig. 5.10 will be converted into two patterns shown in fig. 5.13 and fig. 5.14.

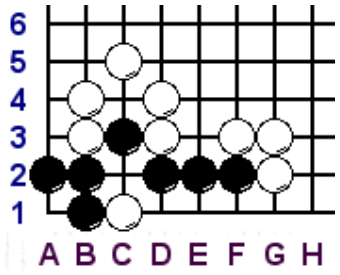


Fig. 5.13

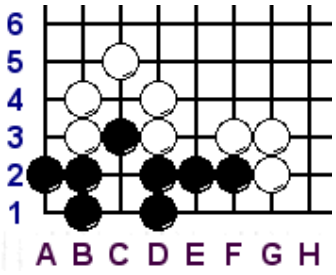


Fig. 5.14

Patterns of odd number moves in correct play and even number of wrong answer is unambiguous. In fig. 5.11 and 5.12 the black corner is dead no matter next turn black play on which location, while in fig. 5.13 and fig. 5.14, black is alive no matter white play on which location. For patterns we extract from book, no manual collapse will be performed. Similarly, life pattern and death pattern will not present in a same file but separately saved into SGF files. The incomplete life patterns from book will be named into another prefix, but the dead patterns from book will be stored with the same prefix as the dead pattern from the Internet.

5.3 Conversion from SGF to ANN inputs

With the patterns all stored in SGF format, there still a lot to be done before it is used as training sample and test cases. First is the orientation problem were the ANN expects inputs are at the top left corner. Second is the color consistency problem where the enclosing stones should be white and the enclosed stones need to be black. Third is the isomorphic patterns problem. With the three problems solved, finally, a single text file for the training sample and another text file for the test case will be generating. To tackle these problems, four PERL scripts were written.

Orientation

The first PERL script will lookup the four corners one by one in each SGF file. If it found some stones in the corner, the pattern will be flipped accordingly.

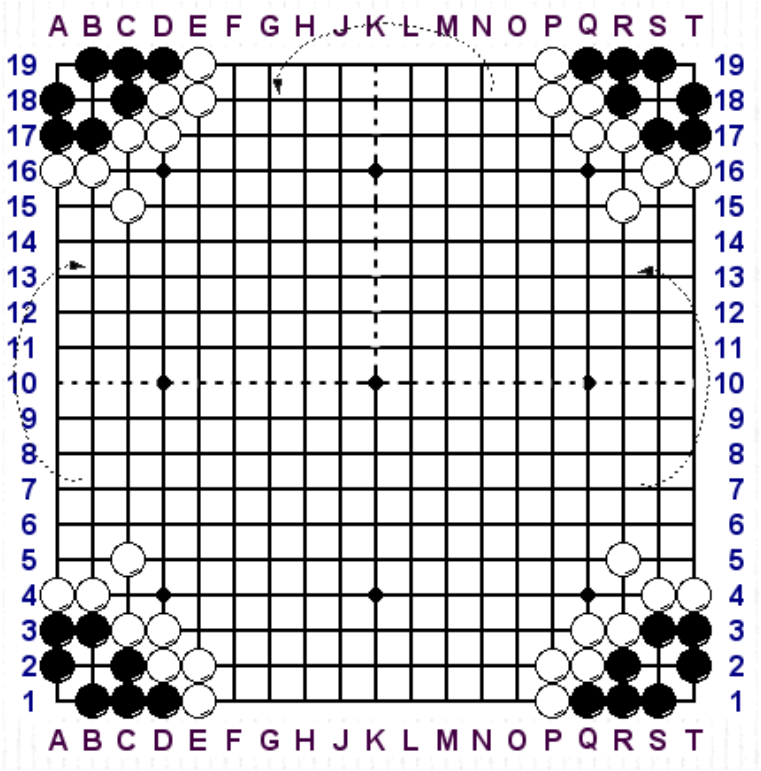


Fig. 5.15

Fig 5.15 illustrated the conversion. Pattern at the bottom left is mirrored against the horizontal line 10 and obtained the same pattern at the top left. The one at the top right will be mirrored against the vertical line K to obtain the same pattern at the top left. For pattern at the bottom right it will mirrored against the horizontal line 10 then do another mirror against line K.

Each non-empty corner this PERL script encountered, it will output the corner with correct orientation into a text file. Each text file contains 8 lines and each line compose by 8 characters. 'X' represent the black stones, 'O' represent the white stones, and '.' represent empty location. The pattern in 5.15 will be output as:

```
1:.XXXO...
2:X.XOO...
3:XXOO....
4:OO......
5:..O.....
6:........
7:........
8:........
```

Fig.5.16

Color Consistency

The PERL script maintaining the color consistency will open the text files output from the first PERL script. It read in the content, builds up the array then performs a scan. It start to lookup from the 8th line and inspect each character in the 8th line from right to left. Take fig. 5.16 as example, 8th line only contain empty location. Then it trace up one line to the 7th line, again it is empty. Same for the 6th line, it is empty again. When it reached the 5th line, the first stone it encountered will be 'O' which means white. So this pattern must be white enclosing black, which is correct. On the other hand, if the first stone it found is 'X' which means black, then the color of the pattern need to be swapped and it swap it and overwrite the original file.

The reason for this scanning algorithm to work is that if black is being enclosed, it is impossible to found black stones before white for closed life and death when we scan from the outside to the inner up corner.

Isomorphic Pattern

For patterns that are used for training, an isomorphic pattern will be generated. Generating the isomorphic pattern is quite a handy for PERL, just take out the first character of each line then output it to a file. Repeat this process 8 times an isomorphic will be created. Fig. 5.17 is a copy of fig. 5.16 while fig. 5.18 is the isomorphic of fig. 5.17.

```
1:.XXXO...  1:.XXO....
2:X.XOO...  2:X.XO....
3:XXOO....  3:XXO.O...
4:OO......  4:XOO.....
5:..O.....  5:OO......
6:........  6:........
7:........  7:........
8:........  8:........
```

Fig. 5.17 Fig. 5.18

Final Output

After all the patterns at the correct orientation and with color consistency check, required number of test cases will be randomly select the from the pool of patterns. For those selected to be the test case, it will be copied to a folder and copy those remaining samples to another folder as training samples. The script generating isomorphic patterns will be applied to the training samples only. This make sure training sample contains no information about the test case. If the isomorphic patterns are generated before selecting test case, the training samples may contain the isomorphic of test cases.

The final script generates two files, one is for training and another is for testing. Each pattern in text format will be transformed into a single line of 64 numbers from the 1st character at the first line to the last character in the 8th line sequentially. 0 represent '.', 1 represent 'X' and –1 represent 'O'. Fig. 5.17 will become:
0 1 1 1 –1 0 0 0 1 0 1 –1 –1 0 0 0 …
The expected output will be generating at the next line according to the prefix of the file name. That is the whole process of converting patterns to ANN's inputs.

6 ANN Designs

The backpropagation ANN used in this project is constructed by the FANN library[15]. As mentioned in the chapter 4 Experiment part, a general fully connected ANN and a 2D Go board ANN are used in both experiment. The fully connected ANN with configuration in 64-64-30-2 and 64-64-30-1 were used in the first and the second experiment respectively. The transfer function used is log-sigmoid $f(n) = \frac{1}{1+e^{-n}}$ where n is the output of the summer in a neuron. The only exception that did not use the log-sigmoid as transfer function is the two output neurons used in the first experiment. It used a hard limit threshold function.

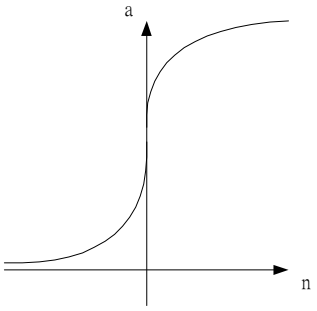


Fig. 6.1 The curve of the log-sigmoid function. When $n \to +\infty$, $a \to 1$. When $n \to -\infty$, $a \to 0$.

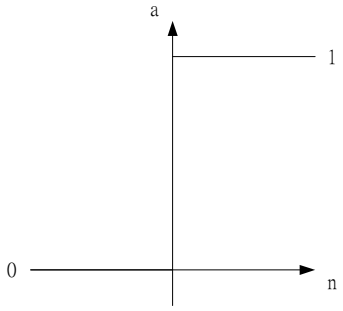


Fig. 6.2 The behavior of the threshold function. When $n \geq 0$, $a = 1$. When $n < 0$, $a = 0$

In the first experiment, output 1 0 means life position, while output 0 1 means dead position. The output of the second experiment is log-sigmoid, so 0 means dead position and 1 means life position. The learning rate used is 0.5. Some other network configuration like 64-128-30-2, 64-128-50-2, and 64-128-50-30-2 were tried, but the result was not any better than the smaller 64-64-30-2 network. Also the 0.3 and 0.7 learning rate was tried and the result obtained is similar.

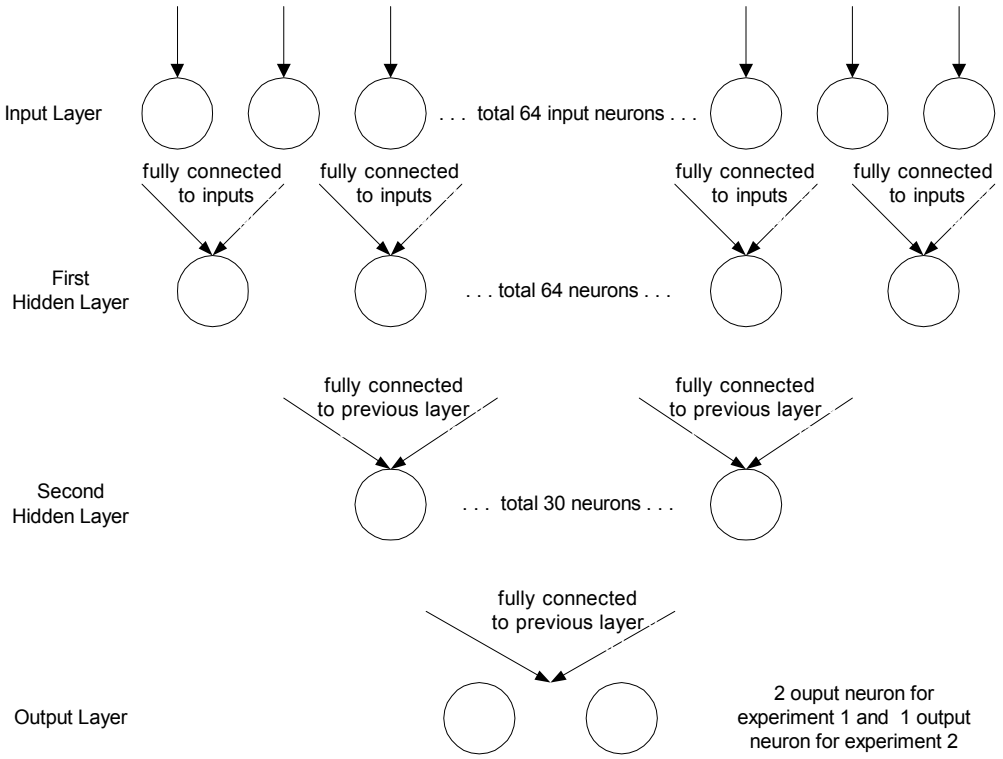


Fig.6.3 The architecture of a fully connected ANN

Beside the fully connected ANN, a 64-140-50-30-2 and 64-140-50-30-1 2D Go board ANN with special designed partial connection between the input and hidden layer is used in experiment one and two respectively. Similar to Horace's design mentioned in Chapter 3.3 Related Works. However, the 64 extra inputs were not employed in 2D Go board ANN. Also Horace starts the partition from size 3 x 3 to 7 x 7. 2D Go board ANN started the partition from 2 x 2 to 8 x 8.

Subsection size	Number of subsections	Connection to the input neuron
2 x 2	49	4
3 x 3	36	9
4 x 4	25	16
5 x 5	16	25
6 x 6	9	36
7 x 7	4	49
8 x 8	1	64

Table 3.2

So there are 49+36+25+16+9+4+1 = 140 neurons in the first hidden layer with each neuron represent a single subsection. Below is the architecture of the 2D Go board ANN.

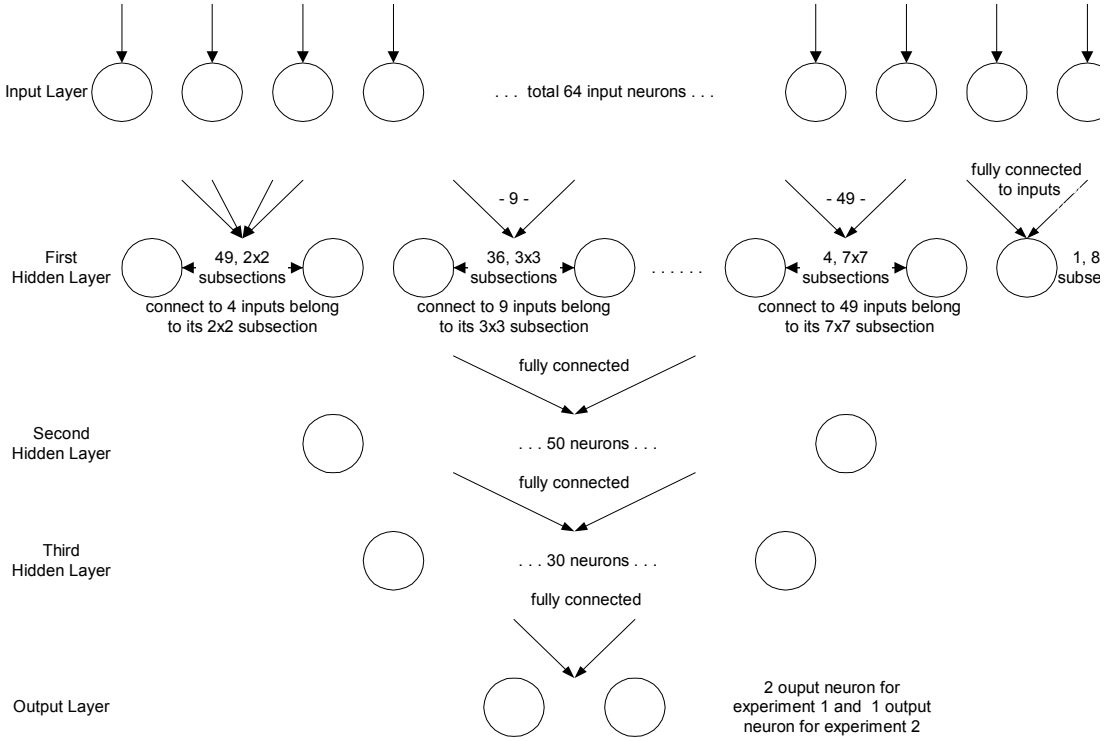


Fig. 6.4 The architecture of the ANN with special designed partial connection between inputs and the first hidden layer.

Without the subsection design, the ANN is like playing on inputs of single dimension array. This kind of special design implicitly let the neural network know which inputs are close together which are not. So the ANN may get a better idea that it is operating on a 2D board not a 1D array. This similar designed have been applied on Checker[16].

Deficient of the extra 64 input of location map

For Horace's design, he used 128 inputs to intake the 64 board map input and the 64 location map input. The location map is generated from the board map with only the corner stone to be evaluated left. This tricky design works well for evaluating final

position because at the final position, if the black group from the location map cannot show up two separated obvious eyes, it must be dead. However when it comes to evaluate on datasets contain life groups not in final position. This design is doubtful.

.

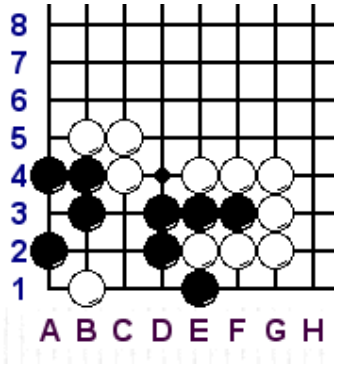


Fig. 6.5 life group

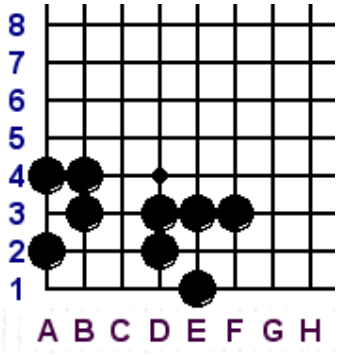


Fig. 6.6 location map of fig. 6.5

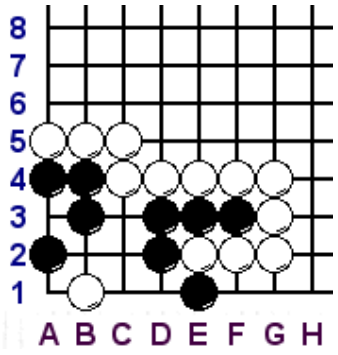


Fig. 6.7 dead group

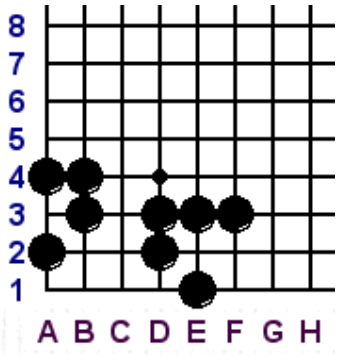


Fig. 6.8 location map of fig. 6.7

Fig. 6.5 is a life group. According to Horace's design, the location map like Fig. 6.6 will be generated. However, fig.6.7 is a dead group and still generate the same location map as fig. 6.5. In fig. 6.7 white can play C3 to kill the black group. If the ANN depends heavily on the location map inputs, which only contains the corner stones to be evaluated. It got a great chance to work well on evaluating final position but seems it will lead to mistake for dataset contains life group not in final position. Those inputs from the location map become noise data to the ANN. Hence we keep our design with only 64 inputs without the extra-generated location map input.

7 Results

There are 748 patterns for the first experiment (contain dead pattern from book, life pattern and dead pattern from final game position, but without life patterns not in final position). With 450 life pattern in final position and 298 dead shape. We randomly select 200 out from the pool as test case. Then the remaining 548 patterns will be flipped to obtain another 548 isomorphic patterns for training.

In the second experiment, 252 life patterns not in final position are added. The network configuration is similar to experiment one. We only made changes to the output layer. This time we randomly select 250 out from the pool for test case. As we got an extra type of pattern added to our pool, so we select 50 more out from the pool in order to obtain a more accurate strength of the ANN in recognizing all these three type of pattern. Similarly, isomorphic patterns of the remaining samples will be generated for training purpose.

We used an approximation:

If $(\text{EXPECTED-OUTPUT} - \text{ANN-OUTPUT})^2 < (0.25)^2 = 0.0625$

We count it as correct. EXPECTED-OUTPUT will be 1 life pattern (no matter it is in final position or not) and 0 for dead pattern.

Results	Experiment 1 Correct rate	Experiment 2 Correct rate
64-64-30-2/1 fully connected ANN	94% 188 / 200	89.6% 224/250
64-140-50-30-2/1 2D Go board ANN	97% 196 / 200	94.8% 237/250
Horace – human selected training dataset with 969 samples	96.9%	Nil
Horace – human selected training dataset with 1495 samples	98.45%	Nil

Table 7.1

Table 7.1 summarized the result obtained and with the comparison with Horace's result. For the first experiment we obtained 97% which is similar to Horace's result with ANN trained with 969 samples, but 1.45% worse than the ANN trained with 1495 samples. Horace did not mention about isomorphic patterns, so assume he did not generate the isomorphic pairs for his training input. In our first experiment, after the 200 test cases being selected out, there will be 548 samples left. 548 x 2 = 1096 samples. So for the ANN in this project it is trained with 1096 samples. The result is really similar.

Besides, it is more important to see how different in accuracy of the 2D Go board ANN and the fully connected ANN in recognizing stones not in final position. There is a general growth in correct rate with 2D Go board ANN which enlarging a fully connected ANN could not archive. 64-128-50-30-2/1 fully connected ANN was tested but the results are not any better than the smaller 64-64-30-2/1 ANN.

ANN type	Mean Correct Answer	Mean Correct Rate
Fully connected	217.4 / 250	86.96%
2D Go board ANN	227.4 / 250	90.96%

Table 7.2 Mean Correct Answer and Correct Rate for 10 consecutive random generated and fully trained networks in experiment 2.

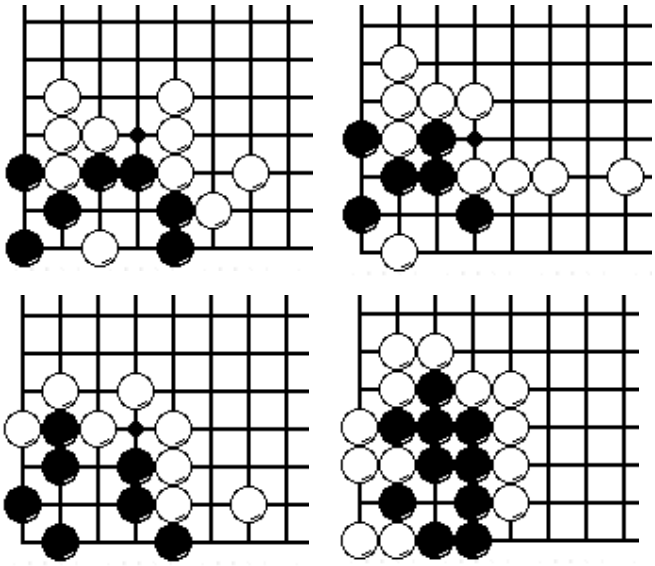


Fig. 7.1

Fig. 7.1 shown 4 samples of correctly recognized patterns in the experiment 2. The 2 patterns on top are dead and the other 2 at the bottom are alive. There is no significant characteristic to tell which is alive and which is dead without really look at it and read. It is impressive to get 94.8% correct on 250 test cases contain more than half of the patterns like above.

8 Discussion

Horace did the first attempts to apply ANN to recognize alive and dead stones in final position and obtained good result. However it is too early to say that ANN can do anything for programming computer Go as the ANN only works when the game is almost finish where many groups of stones reached its final position. The major doubt in Horace's experiment is the ANN design. He used 128 inputs for an 8 x 8 (64 points) board position. The first 64 inputs is the board position, while the extra 64 location map inputs seems problematic. The reason is elaborated in the chapter 6 ANN Design. We think the extra 64 inputs is an added burden. Without the extra 64 inputs, similar result as Horace did was obtained in experiment one. Also when we come to evaluate the dataset contain incomplete life stone, the data from the extra 64 inputs become noise to the ANN. As dead stone and incomplete life stone will up into the same shape after removing opponent stone like we have shown in fig. 6.5 and fig. 6.7

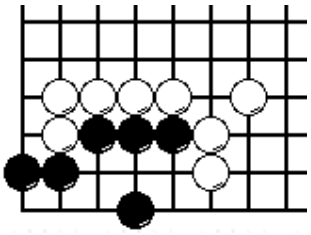


Fig. 8.1 Life shape being trained

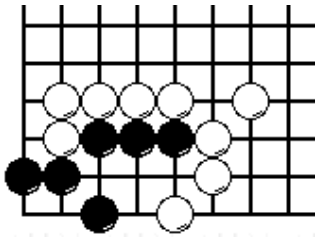


Fig. 8.2 Dead shape being trained

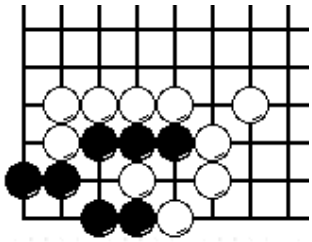


Fig. 8.3 Dead shape in test case

As stated in the chapter 5.2 Sample Specification, for a single question in the book, both correct and wrong answer will be captured down. This result in a lot of similar shapes but their life status is different. As shown in above, fig. 8.1 and fig. 8.2 are two patterns being trained but fig. 8.3 is not. The three patterns are very similar and the ANN can correctly recognize fig. 8.3 is dead shape. It seems that after the training the ANN discovered the key to determine what is alive and what is dead.

In the experiment two, ANN showed its competence in evaluating the life status of corner stone not in final position. This is important to positional judgment system. When a board position is being evaluated, the life statuses of stones on board are

required[17]. If we are doing heuristic to obtain life and death information, it will take a long time and computing power is wasted as the exact solution was not needed. To speed up the search will require cutting off the tree, but it would result in inaccuracy. For the computed solution, even though it can be stored for later uses but it is hard to know how long it had to be stored. Also, later a group of stones maybe used to exchange something else with opponent then the stored solution is no longer valid. This is what ANN can fit in with its capability to quickly evaluate the life status.

9 Project Experience

By doing this project a lot have been leant, in both technical and non-technical aspect.

At first, I was targeting at problem that is too large for me to solve with this limited time. Although I changed the topic a last, but the background research was not waste. It enriched my general knowledge to the problem in computer Go and how other people address the problems in Go, like the isomorphic patterns, the DFA algorithm. All are very interesting and worth to learn about.

Actually this is a challenging project to me. Before digging hard on it, I know nothing about ANN and computer Go. However as I keep reading on, I get more interested. The ability of ANN is quite amazing. Also the book "Blondie 24" is a very funny book. It is like a novel, but the evolutionary algorithm they using is quite a stunning. The partial ANN design used in this project was learnt from this text. Although they use it for checker, but the idea is similar.

Furthermore, I learnt to be initiative and not to being frustrated when problem arise. During the process of doing the project, I gradually improved my self-initiative to search to texts, articles, papers, and other resources when getting stuck.

However there are quite some difficulties I have faced. First is the data collection process. This requires a lot of manual work. I have to search for game in the LGS to found suitable inputs. After found the patterns needed, I need to input them manually with noise data removed. Also there are orientation problem, color consistency problem, and isomorphic patterns I have to solve. Luckily with the excellent language in text manipulation, PERL, I can just write some handy scripts to automate some of the process.

Also, I have to improve my project management skill; I am always off the schedule and was so stressed about it.

10 Possible Extensions

From the result of the experiment two, it indicates there is a great chance to apply the techniques in ANN to solve certain degree of life and death problems. It shows that ANN is capable to recognize life and dead stone not in final position. Actually we may go one step ahead and try if ANN itself can be used to find the exact sequence of play in a life and death problem. The change in design of the output layer in experiment two is going to try this. By using a minimax game tree, we ask the ANN to give a score to each board position and see if it is correct. However there is not enough time to conduct this experiment. Only a board position generating functions is created. Even though by ANN itself may not be accurate enough to find the whole sequence of move, but if we can create an ANN that is accurate enough to provide the first move that will greatly decrease the search space already.

Besides, there is an evolving computer learning technique on the ANN – the evolutionary algorithm. It had been shown great success in creating ANN to play checker in expert level without using human expertise [5][16][18][19]. By applying this algorithm – generate a set of ANN to play with the life and death question among themselves then pick a set of them and generate offspring and test again. See if a life and death expert could evolve.

There still a lot of possibility for ANN to be applied in solving the life and death problem in Go. Many people are doing pure ANN Go playing programs and pure heuristic. Maybe we can try to use the hybrid approach for ANN to take care some part of the game and left the other to our classical heuristic.

11 Reference

[1] Jay Burmeister, Janet Wiles “The Challenge of Go as a Domain for AI Research A Comparison Between Go and Chess”

[2] Bruce Wilcox, “Computer Go”, Computer Games II, pp 94-135

[3] Mark Boon, “Pattern Matcher for Goliath”

[4] Tanguy Urvoy, GNU Go team, “Pattern Matching in GO with DFA”

[5] Kumar Chellapilla , David B. Fogel “Evolving an Expert Checkers Playing Program without Using Human Expertise”

[6] G. Tesauro. TD-gammon, a self-teaching Backgammon program, achieves master level play. Neural Computation, 6(1), 1994.

[7] GNUGo document version 3.2

[8] GoTools: http://www.qmw.ac.uk/~ugah006/gotools/

[9] Horace Wai-Kit Chan, “Application of Temporal Difference Learning and Supervised Learning in the Game of Go” Chapter 5.4

[10] Thomas Wolf “The program GoTools and its computer-generated ysume go database”

[11] Thomase Wolf “OPTIMIZING GoTools’ SEARCH HEURISTICS USING GENETIC ALGORITHMS”

[12] http://www.intelligentgo.org/en/computer-go/overview.html

[13] 戸沢昭宣九段“九級から一級までの詰碁”

[14] 林海峯九段 死活的基礎

[15] FANN Library: http://fann.sourceforge.net/

[16] David Fogel, “Blondie 24”

[17] Shi-Jim Yen, Shun-Chin Hsu, “A positional Judgement System for Computer Go”

[18] Kumar Chellapilla, David B. Fogel “Evolution, Neural Networks, Games, and Intelligence”

[19] Kumar Chelloapilla, David B. Fogel “Evolving Neural Networks to Play Checkers without Relyig on Expert Knowledge”

[20] Tom M. Mitchell “Machine Leaning”

[21] Martin T. Hagan, Howard B. Demuth, Mark Beale “Neural Network Design”

[22] Shi-Jim Yen, Weh-Jyh Chen, Shun-Chin Hsu, “Design and Implement of a Heuristic beginning Game System for Computer Go”

[23] 顏士淨, 嚴祁麒, 許舜欽, “Move Strategies in Middle Game of Computer Go”

[24] JULIAN CHURCHILL, RICHARD CANT, DAVID AL-DABASS “A NEW COMPUTATIONAL APPROACH TO THE GAME OF GO”

[25] Andrew Blais, David Mertz “ An introduction to neural networks”
Available: http://www-106.ibm.com/developerworks/library/l-neural/

Appendix - Go related terms

1. firm life or a firm kill:

Between a firm life and firm kill, there are ko life and seki which are ignored in this project as those patterns are too hard to collect. For the ko life, it is a bit complicated and require the knowledge on the Go rule. Please reference to sensei's library at: http://senseis.xmp.net/?KoFights. To understand seki, you only have to know the liberties or condition to capture stones.

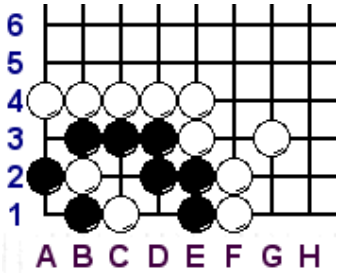


Fig. 13.1 ko life

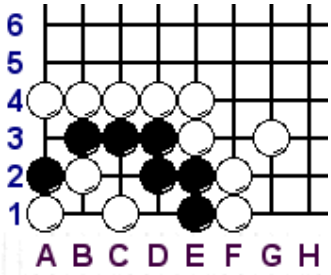


Fig. 13.2 ko life

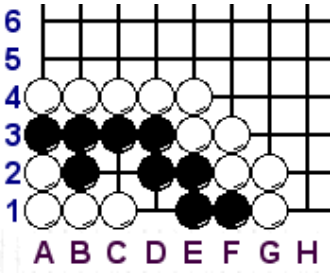


Fig. 13.3 seki

For ko life, like fig 13.1 and 13.2, there is a ko at A1 and B1. If black can play C2 in fig. 13.1 it will be alive. However, if white played B1 in fig. 13.2, black is dead.

For seki, like fig 13.3, neither side can kill each other. If white play either D1 or C2 black will play another location to kill white C2 and archive a life state, so white will not make any move there. However, if black play either D1 or C2 white will play another location to kill black. In this situation, neither side will make a move in D1 and C2 and neither side can kill each other. This is seki, both alive.

Both kind of pattern are hard to collect, especially the seki, I even cannot find 10 seki out of 231 games collected from the Internet.

2. String:

String is a maximal set of vertices on the board which are connected along the horizontal and vertical lines, and are of the same color[7].

In fig. 13.4, C5, D5, D4, and D3 is one string, while F4, G3 are itself a string which contain only one stone.

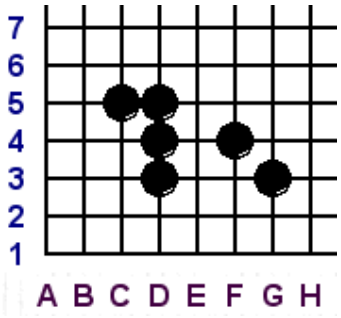


Fig. 13.4

3. Go strength ranking system on amateur player.

Weakest	Strongest
30 kyu< 29k < … < 1k < 1 dan < 2d < … <7dan	

4. shortage of liberties

For life and death involve shortage of liberties, it is illustrated in fig. 13.5 and fig 13.6. Fig. 13.5 is a dead shape, the reason is the shortage of liberties of the C3, D3, and E3 group. If black play C2, it odiously do not have two eyes. However, if black play E2, like fig 13.6, white will capture C3, D3, E3, and E2 by play at "a". If there is a loose in liberties, like fig 13.7, this time black can play E2 without being capture and is in a life state. (The stone at C6 and E6 in fig. 13.7 is to prevent black from escape, as the case of escape is not considered)

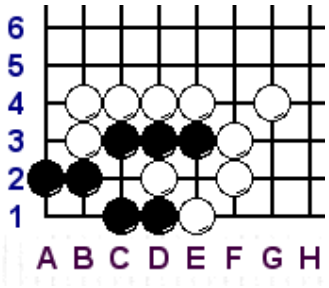


Fig. 13.5

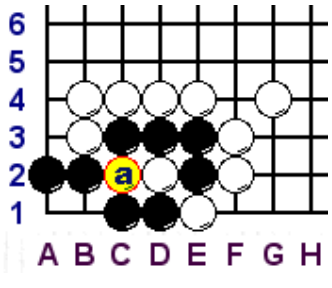


Fig 13.6

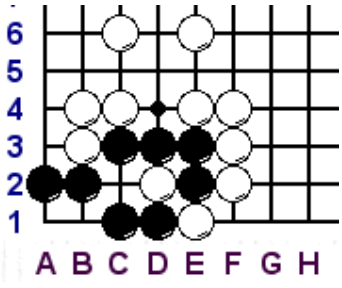


Fig. 13.7

5. 6k* and Legend Go Server (LGS)

LGS is a Go server basically built based on the source of NNGS (No Name Go Server). NNGS itself is a Go server in US, also it is the name of Go server project in sourceforge (http://sourceforge.net/projects/nngs/). In this server, player's Go strength with an '*' at the end means that ranking is estimated by the server. Players are allowed to set their own rank, but the rank will not got a '*' at the end. After the players played more than 15 games with other '*' player and got some won and lost. Then the server will base on that 15 games to estimate the rank of the play. For GNUGo, the LGS server estimated it has 6k* strength.

Link to LGS: http://lgs.taiwango.net

6. Nihon Kiin

It is the Go school where they train professional players and issue professional and amateur rankings.

Link to Nihon Kiin: http://www.nihonkiin.or.jp/

7. Komi

Black players always play first in Go. So black got some advantage over the white player. The Komi system is to balance this for white player to make the game even. Komi means after counted the territories and the captured stones, black need to give the extra komi points to white. 0.5 Komi means white will win by 0.5 if the points both side gain is the same.

8. 香港圍棋網

香港圍棋網 is a location organization that aim at increase the popularity of Go. It organizes Go competition, teach Go, sell Go books and Go related tools, for examples Go board and stones. There is 圍棋研修中心 of 香港圍棋網 in Chai Wan.

Link to 香港圍棋網: http://www.hkgo.com.hk

City University of Hong Kong
Department of Computer Science

BSCS Final Year Project 2003-2004

Interim Report

Project Title:

Computer Go

(Volume 1 of 1)

Student Name : **Wong Wai Hung**

Student No. : **50249864**

Programme Code : **BSCCS**

For Official Use Only

Supervisor : **Dr. Chun, Andy H W**

Date : **17-11-2003**

Table of Content

1.0 Introduction

1.1 About Go

Computer performs very well in varies board game, such as Chess, Checker and Othello. Strong Chess programs can even challenge players like Garry Kasparov. However, no go program are as strong as amateur shodan exist. Since Albert L. Zobrist has created the first program that plays a complete game of Go in the 1968, the progress in Go program is very slow.

Go is a board game origin from China. Its rules are simple. Two player alternately place stone of their color on the board with 19x19 intersections (Smaller size like 9x9, 11x11, 13x13… are possible). The objective is to enclose more territory than the opponent. Stones that cannot be captured separately are defined as a string. A stone or string will be captured if all its adjacent intersections (diagonal intersections are not required) are occupied by opponent. The empty adjacent intersections are known as liberties. There is an important pattern known as eye. Eye is a liberties enclosed by a single-colored string. When a string has two eyes, it is known as unconditional life, which is safe from being capture. A race for one side to kill a string and the other to make life is called a life and death question.

Generally we divide a Go game into three stages: starting, middle game, and end game. Players are allowed to play on every intersection on the board but we always start from the corner. The human go expert have researched and formulated a large set of standard move sequence around the corner starting fight, we call it joseki [2]. However, it is often that we found players do not following those joseki strictly and

new variations from professional players are keep coming out. While in the middle game and end game, many tesuji [2] and life and death problem are involved.

1.2 Problems

Compare to Chess, programming Go is a more difficult [1]:

- The large board size in Go make heuristic search impossible.
- When dealing with life and death and tesuji, moves are played at unusual location and even make suicidal move. We can hardly found a formulated solution to all these problems; also a little change in the shape may result in a different tesuji.
- Evaluation of board position is hard. Territories are easy to count, but thickness, influence, and moyo [2] are hard to evaluate. Also the eye patterns are difficult to examine – hard to evaluate a stability of a string (can it make two eyes for unconditional life?).
- Cannot evaluate a single move, but need to examine a variation at the final position. When good move played with out consecutive support, good move can become bad move.
- To examine a variation, we need a correct look ahead while brute force is infeasible.
- Require tactical planning and deep look ahead.
- Complex logics in ko[2] fight

2.0 Project Objectives

The problems motioned in 1.2 are not easy to solve. However, with the current technology in AI and algorithm, we still can create a Go program to play certain degree of Go. By doing this project, the following are target to achieve:

- Learn more on computer game playing, artificial intelligence and algorithms then apply them to programming computer to play a complete game of Go
- Get familiar to GNUGo and learn its methodology, algorithms, and implementation
- Make use of neural network and GNUGo to create my own Go program
- Evaluate the strength of my Go program
- Review for improvements and weaknesses

3.0 Scope of the Project

The Scope of the project are as follows:

1. Study publication on Computer Go and neural networks
2. Study the concepts and implementation on neural network
3. Study the GNUGo document and learn there implementation
4. Test of the GNUGo API and implement sample neural networks for testing.
5. Design and create my own Go program with the use of GNUGo
6. Evaluate the strength and review the game played by my own Go program
7. Review for improvements and weaknesses

4.0 Background Research

4.1 Zobrist, Ryder, and Reitman-Wilcox[4]

Zobrist created the first go program, which can play a complete game of go, in November 17, 1968, which is a landmark in computer Go. Zobrist's program separates the board into black and white region by using influence function. The influence function will evaluate the potential territories and influence for wall and thickness. Then the program will apply a set of pattern on the whole board in all area and orientations. If a pattern is match a score will be given for a specific location. After all the patterns were applied and scores were summed up for each location. The one with the highest score will be played. To eliminate illegal move, location that match a illegal move will be given a large negative value, so it will not turns up at the final choice.

Ryder, followed the underlining spirit of Zobrist's program, created his own go program in 1971. He improved the performance of Zobrist's entire board examination by only apply simple features for the entire board on the first round and select 15 moves with the highest score for the second round. Also, more important, features are examined for both side. So the program will have the intension to play on opponent's good point.

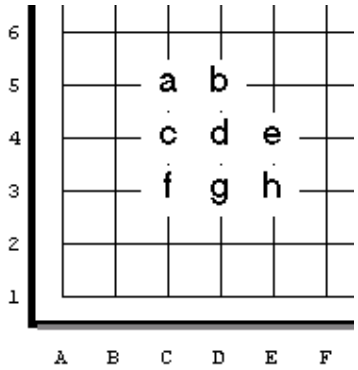
Reitman and Wilcox created a large LISP program in the seventies. In Reitman and Wilcox's program, they have implemented a tactician. The task of the tactician is find the right answer for the program, such as: can a string being capture? , can a stone being saved?. Their tactician will perform a look-ahead by attempting to simulate reasoning. The tactician is limited to explore a reasonable numbers of variations and

returns for an answer. If a few well-selected variations comes up with the same answer, the answer will probably correct and the tactician will stop and return the answer.

4.2 Jimmy

Jimmy is a go program created by a Taiwan professor Dr. Shi-Jim Yen. There are not any official document about the internal of Jimmy. However from the publication [5], [6], and [7], Dr. Yen has evaluated his methodology and implemented on Jimmy.

These three papers proposed a beginning game system [5], middle game strategies [6], and a positional judgment system [7]. Three publication all talk about how to quantify moves while at different stage of game apply different quantifying rules.



At the beginning of the game, player always start at the corner like a-h. When the board is empty, it is no different for position c and g, a and h, and b and e, those pairs of location just different in orientation. So there are only 5 choices in occupying the corner. In [5], a pattern matching approach was used to evaluate the shapes of occupied corners because of options are pretty limited. Also, it use a joseki tree to predict the location of next move and by searching the tree down to find the quantified values of a variation in the joseki tree. After occupying the corner, moves on extending edges will be generated. Here it used a similar approach as Ryder did, to evaluate a point for both side. This mean that if black plays at point 'K', black gain 2-point influence and 4-point land, while white plays at point 'K', white gain 2-points influence and 4-point land. Then the quantified value of point 'K' is 12 (4+2+4+2).

Because of efficiency, we start the go game from corner and we have joseki at the corner. While in middle game, it becomes more complicated to quantify moves. It

search the entire board select a set of moves and evaluate the values of those moves in different aspect. It use five different sub-system / module to quantify moves. The five sub-systems are: Territory sub-system, eye shape sub-system, enclose-and-escape sub-system, connection sub-system, and good-point sub-system.

In handling attack and defends of weak strings, it will also evaluate the value of capturing or defending of the weak string instead of capture or defend it immediately. To quantify the value of capture, first to count for the territories gain then evaluate the change in influence because capturing may result in eliminating cutting points. When decided to capture, it will search for different ways to capture and choose the most beneficial one.

Territories are easy to count while influences are not. In the positional judgment system [7], it uses a pattern matching system and a string capturing system to evaluate influence. The pattern matching system recognizes special pattern and shape, and the string capturing system recognizes dead string. It introduces a rough influence, which applies an influence function on every string on board. Then search for links and forming groups of linked strings. After all these information gathered, the rough influence will be adjust, for stronger groups the influence will be stronger, weak stone influences will be turned down while influence will be removed for dead groups. Summing up the territories and the final influence of all the groups on board.

Although there are not any official ranking for Jimmy, but from the observation of Jimmy's performance it at least has amateur 8 kyu strength. To conclude, from the 3 publications [5][6][7], Shi-Jim Yen described many ways to produce accurate method

to quantify moves. With the aid of pattern matching / recognition, searching and joseki tree, we can find move by its value and made position judgment.

4.3 GNU Go

4.3.1 Overview

GNU Go is a free and open source program that plays the game of Go. [8] Many Go programs are distributed in binary format and the underlying algorithms are unknown. GNU Go is a rare one that is open source and very well documented. My project will make use of GNU Go to create my own go program. Current version of GNU Go is 3.4, but the document is still for version 3.2, so my project will still use the version 3.2 GNU Go.

GNU Go is a very complex program. From compilation to execution, you are allowed to alter the behavior of GNU Go. You can choose what kind experiment modules to compile with, the default level of GNU GO, trigger which experiment module on this execution, play mode (console, Go Text Protocol or Go Modem Protocol)…etc.

It contains an interface and a Go engine, which deals with I/O and move generation respectively. While in the move generation, pattern matching will also performed.

4.3.2 Interface

The default interface for GNU Go is ASCII, which it will print out the board in ASCII format. However it can also be played with third party GUI interface like gGo or Cgoban I. More important is that GNU Go understands the Co Text Protocol (GTP) and Go Modem Protocol (GMP). These two protocols are used in Internet Go Servers. With a proper client (e.g. gnugoclient which written in Pike), GNU Go can connect to Internet Go Server, accept match, and play with human plays on the server with its own. Another useful feature is GNU Go can read and write Smart Go Format (SGF, which is a standard file format in storing go games). GNU Go can output the game

with it's evaluation of location to sgf file while it is playing. Beside, it can read a sgf file and re-evaluate the scores of the board for each move. The reading result of GNU Go can be traced by using sgf file too. Input a sgf file with a location of a string, GNU Go will start the look-ahead to evaluate if that string can be attacked or not and the variation it checked will be written down to a output sgf file which you can see which location it checked.

4.3.3 Go Engine

The engine is the core of GNU Go and very complex. There are modules for information gathering, move generation, eye shape recognition, pattern code, pattern matching, tactical reading, life and death reading, influence function

- Information Gathering:

For information gathering, it will collect information for *worms* (worms means connected black stones, connected white stones, and connected empty space. Strings are non-empty worms). Then it will compute the influence. Next it will gather information about *dragon*, connected strings. After the number of eyes, life status, and connectedness between strings are evaluated, it will go back and re-compute the influence. Which is very similar to the idea of rough influence and final influence presented in Dr. Yen's publication [7].

- Move Generation

Once all the information about position are found, it will start to generate move.

Moves are generated by different modules / functions called *move generators*. The move generators will not assign values for its proposed moves, nut enumerated justifications for them, called *move reasons*. The value of moves come after all the

proposed moves and moves reason are generated. The best 10 moves will be selected to review for more features. This incremental approach is similar to Ryder's.

- Eye Shape Recognition

Module used in life and death reading and information gathering in evaluating eyes of dragon.

- Pattern Code

Pattern code is a module that generates the pattern database files. These include: joseki db, eye db, attack and defence pattern db, and connection db

- Pattern Matching

A DFA pattern matcher is implemented. This will be further described in the Pattern matching part.

- Tactical Reading

GNU Go performs three distinct types of reading: *Tactical reading* which typically is concerned with the life and death of individual string, *Owl reading* which is concerned with the life and death of dragons, and *life reading* which attempts evaluate eye spaces. Attack and defense functions are called each other recursively to carry out the reading.

- Life and Death Reading

This is the *Owl reading* mentioned above

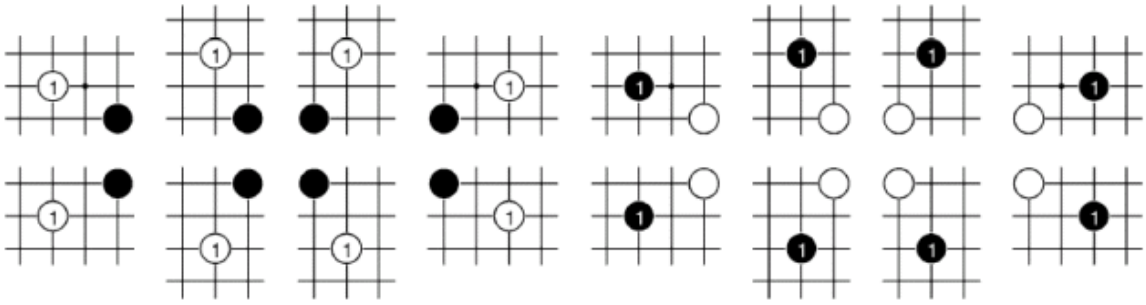
- Influence Function

Influence functions evaluate the potential territories and thickness with the aid of patterns. Patterns contain eye shape are regarded as thicker and extra influence will be added.

4.4 Pattern Matching

Pattern matching is very important in Go. Human expect see shapes and patterns of stones on board. For such a large board size, brute force searching is impossible which implies we have to perform look-ahead in a clever way.

To perform pattern matching in Go, we have to take care about isomorphic patterns. On Go board, a single pattern can appear in different orientation and swapped color of stones. Therefore, a single pattern can appear as 16 isomorphic patterns shown in the figure follow.



If we use a straightforward approach, we will have to perform matching on every location (19x19 = 361) and one pattern for 16 isomorphic. If there are M patterns and cost to match one patter is N, the total cost becomes: 361 x 16 x M x N.

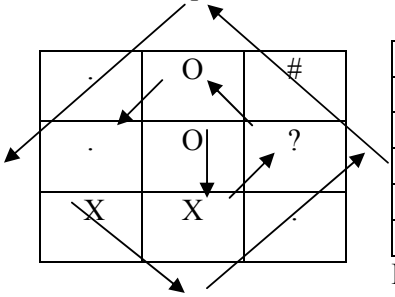
4.4.1 Pattern matcher of Goliath [13]

Pattern in Goliath are represented by 3 32-bit integers, which are used as bitmap for representing black, white and empty. To compare a board position with a pattern, we only have to perform 3 bit operation such as: if (position & pattern) == pattern. Only 8 x 3 bitmaps have to be created for each pattern. For color swapped pattern, it can be matched by using white's pattern bitmap for black's position bitmap

4.4.2 Deterministic Finite state Automaton (DFA)

DFA is the pattern matching algorithm that used in GNU Go. This algorithm is proposed by Tanguy Urvoy and GNU Go team [9].

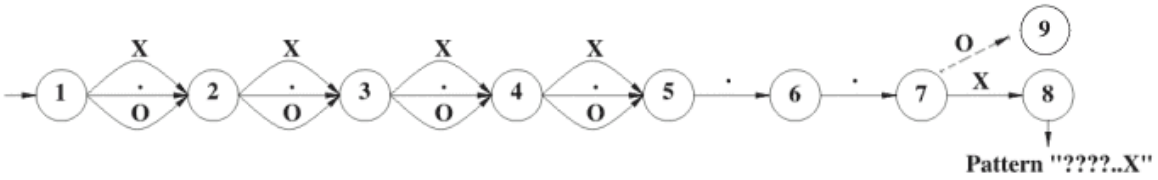
In DFA, 2D patterns will be converted into 1D patterns.



O	White
X	Black
.	Empty
?	Don't care
#	Don't care (can be out of board)
⟶	Scanning Path

Pattern become: OX?O.X#.###.#

Then a finite set of status and transitions will be created like the following figure.



This is a set of state and transitions to match a pattern "????..X"

Assume we have another pattern like "????..O" we can add a "O" transition to state 7 which lead to state 9

This method is efficient in exact matching. It "reuse" the previous result. When we got the pattern "????..O", it will pass though the above transition to state 7. Although we found the last symbol is "O" and cannot go to state 8, but we can immediately found the other transition that lead to state 9 without start matching again from the first symbol.

4.5 Neural Network

Computers are good at exact matching and perform programmed actions, but it is hard for computer to interact with noisy data and response to unknown data and environment. However, neural network helps the computer to perform task that we cannot formulate an algorithm by giving training to the neural net. [12]

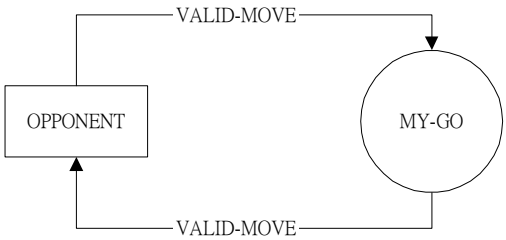
Neural Network is a simplified model of human brain's neuron network. It is composed by network of Threshold Logic Units (TLU). TLU model the neuron of human brain. It collects values from other TLU and only activated after the threshold is reached. TLU are connected to other TLUs with a weight. By giving inputs and the expected output for a neural net, whenever a wrong outcome comes up, it will adjust the weights by different learning rules like: perceptron rule, delta rule, back propagation…etc. [10] After providing several set of input and correct output to train the network, it is expected the neural net will output approximately correct result for untrained data.

In the game of Go, which brute force searching is impossible to perform, neural network should be useful to indicating important locations to perform searching.

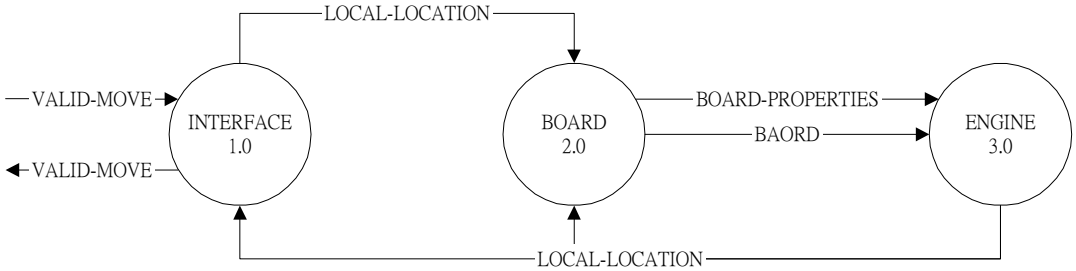
5.0 Design

The preliminary design of my Go program in DFD graph.

Level 1 DFD



Level 2 DFD

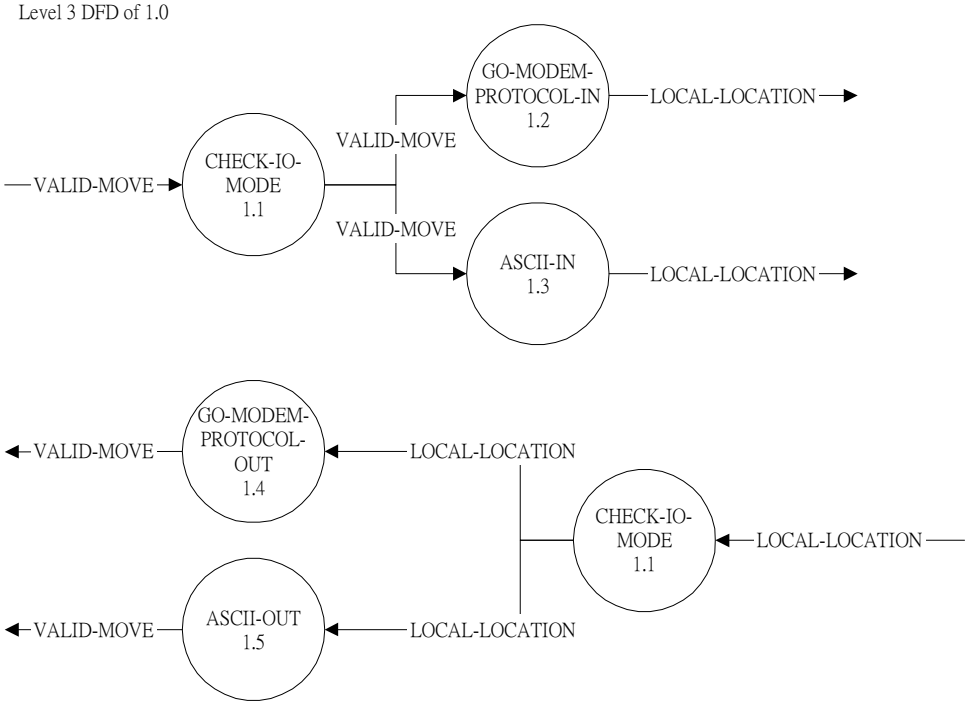


Interface are functions that handles I/O

Board maintain the board information such as strings, string liberties, legal moves

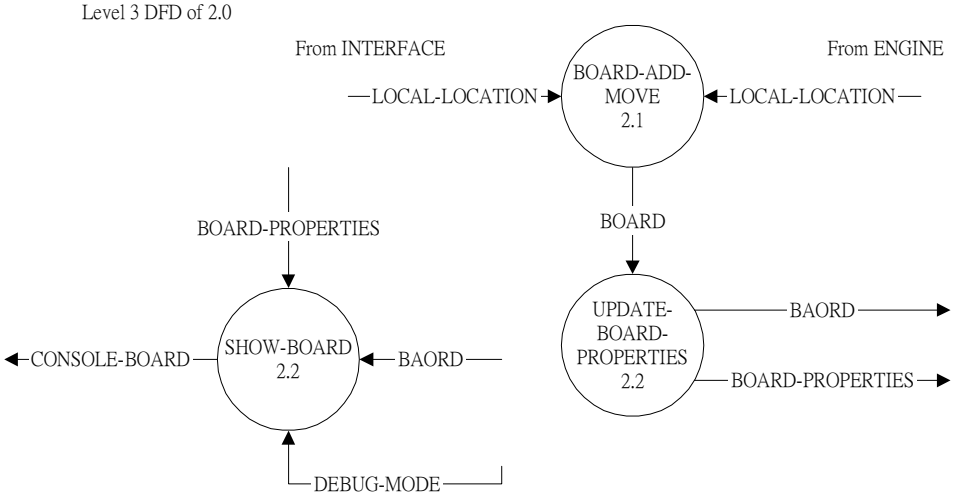
Engine are functions that generate moves and return moves to the board and to the interface for output

Level 3 DFD of 1.0



Details of interface are just coverting the Move location accroding the to play mode

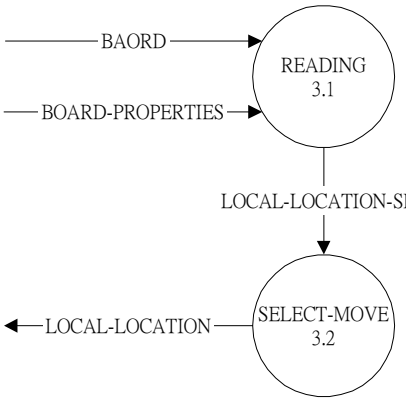
Level 3 DFD of 2.0



SHOW-BOARD will output the board to the console with varise information for debug

After moves added onto the board, UPDATE-BOARD-PROPERTIES will update the information of strings

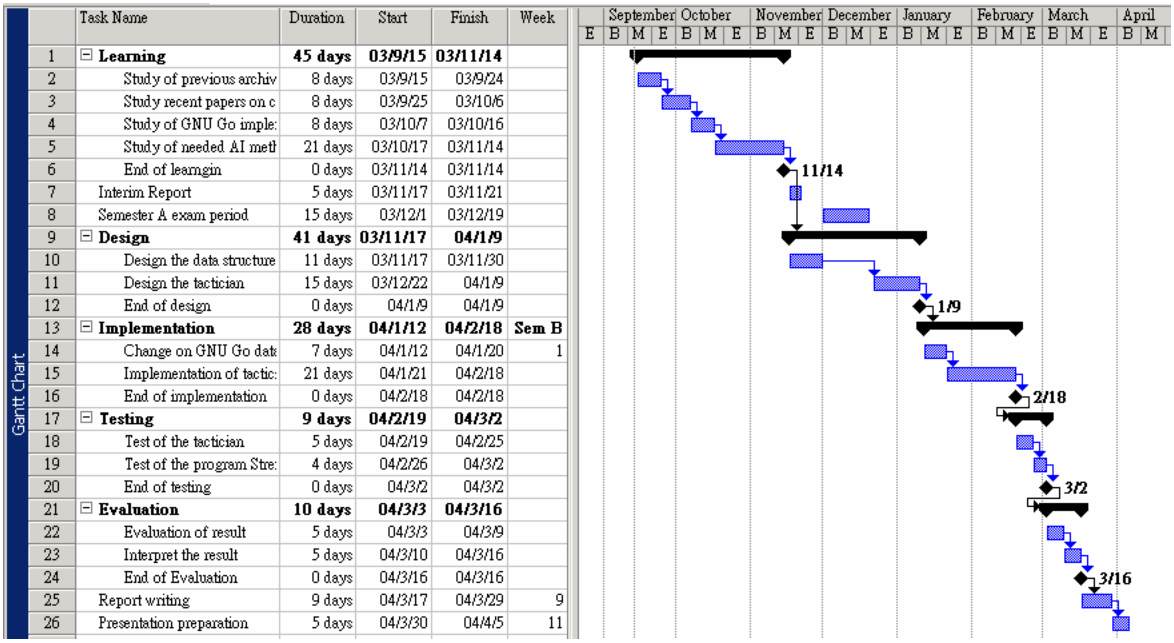
Level 3 DFD of 3.0



READING will propose move after the baord added the move and updated the properties

SELECT-MOVE have the final power to pick a move to paly

6.0 Schedule



	Task Name	Duration	Start	Finish	Week
1	**Learning**	**45 days**	**03/9/15**	**03/11/14**	
2	Study of previous archiv	8 days	03/9/15	03/9/24	
3	Study recent papers on c	8 days	03/9/25	03/10/6	
4	Study of GNU Go imple:	8 days	03/10/7	03/10/16	
5	Study of needed AI metl	21 days	03/10/17	03/11/14	
6	End of learngin	0 days	03/11/14	03/11/14	
7	Interim Report	5 days	03/11/17	03/11/21	
8	Semester A exam period	15 days	03/12/1	03/12/19	
9	**Design**	**41 days**	**03/11/17**	**04/1/9**	
10	Design the data structure	11 days	03/11/17	03/11/30	
11	Design the tactician	15 days	03/12/22	04/1/9	
12	End of design	0 days	04/1/9	04/1/9	
13	**Implementation**	**28 days**	**04/1/12**	**04/2/18**	**Sem B**
14	Change on GNU Go dat	7 days	04/1/12	04/1/20	1
15	Implementation of tactic:	21 days	04/1/21	04/2/18	
16	End of implementation	0 days	04/2/18	04/2/18	
17	**Testing**	**9 days**	**04/2/19**	**04/3/2**	
18	Test of the tactician	5 days	04/2/19	04/2/25	
19	Test of the program Stre:	4 days	04/2/26	04/3/2	
20	End of testing	0 days	04/3/2	04/3/2	
21	**Evaluation**	**10 days**	**04/3/3**	**04/3/16**	
22	Evaluation of result	5 days	04/3/3	04/3/9	
23	Interpret the result	5 days	04/3/10	04/3/16	
24	End of Evaluation	0 days	04/3/16	04/3/16	
25	Report writing	9 days	04/3/17	04/3/29	9
26	Presentation preparation	5 days	04/3/30	04/4/5	11

My Schedule contains 5 parts; include learning, design, implementation, testing and evaluation. Within the design phase, I have reserved 3 weeks for semester A examination so the total number of days spend on design appears as 41. By semester B all the learning and design will have to be finished and implement will start.

7.0 Reference

[1] Jay Burmeister and Janet Wiles, “The Challenge of Go as a Domain for AI Research: A Comparison Between Go and Chess”

[2] Sensei’s Library, with go terms explanation

http://senseis.xmp.net/?header=keywords&term=Go%20term

[3] Markus Enzenberger, September 1996, “The Integration of A Priori Knowledge into a Go Playing Neural Network”

[4] Bruce Wilcox, “Computer Go”, Computer Games II, pp 94-135

[5] Shi-Jim Yen, Weh-Jyh Chen, Shun-Chin Hsu, “Design and Implement of a Heuristic beginning Game System for Computer Go”

[6] 顏士淨, 嚴祁麒, 許舜欽, “Move Strategies in Middle Game of Computer Go”

[7] Shi-Jim Yen, Shun-Chin Hsu, “A positional Judgement System for Computer Go”

[8] “GNU Go development document”

http://www.uni-bonn.de/~uzsxtn/gnugo.html

[9] Tanguy Urvoy, GNU Go team, “Pattern Matching in GO with DFA”

[10] Andrew Blais, David Mertz, “Introduction to Neural Network“

http://www-106.ibm.com/developerworks/linux/library/l-neural/

[11] Karsten Kutza , “Neural Networks at your Fingertips”,

http://www.geocities.com/CapeCanaveral/1624/

[12] Prof. Leslie Smith, ”An Introduction to Neural Networks”,

http://www.cs.stir.ac.uk/~lss/NNIntro/InvSlides.html

[13] Mark Boon, “Pattern Matcher for Goliath”