國立台灣師範大學
資訊工程研究所碩士論文

指導教授：林順喜 博士

深度學習用於愛因斯坦棋研發之初步探討

The Initial Research of EinStein würfelt nicht! with Deep Learning

研究生：曹少剛 撰

中華民國　一百零六　年　七　月

摘要

愛因斯坦棋，是於西元 2004 年由德國中部耶拿(Jena)鎮的一位數學教授—Ingo Althöfer 所發明的兩人骰棋類遊戲。

在 5x5 的棋盤中放入雙方各六個棋子，雙方必須利用擲骰子的方式，來決定當前回合可以移動的棋子編號，透過各種不同的策略，減少我方或敵方的棋子，使我方比敵方優先達成勝利條件，以獲取勝利。雖然此遊戲的遊戲盤面尺寸、棋子數目較其他棋盤遊戲小、少，但是由於融入了骰子這個不確定的要素，大大地增加此遊戲的複雜度，同時也增加了耐玩性與挑戰性。

本研究將嘗試利用蒙地卡羅演算法、卷積式類神經網路的方法，嘗試使用、尋找各種不同的特徵，將這些特徵互相搭配以形成不同的 feature map，藉此訓練類神經網路各個節點的參數(權重)，期望新的方法可以達到、擁有，甚至是超越目前其他強力的愛因斯坦棋下棋程式的棋力。

關鍵詞：電腦對局、愛因斯坦棋、蒙地卡羅法、類神經網路、深度學習。

Abstract

EinStein würfelt nicht! is a dice board game for two players which was invented by a professor of applied mathematics, Ingo Althöfer, who lives in Jena, Germany.

In this game, initially each player has six pieces, numbered 1 to 6, on a board with size 5x5. Each player needs to roll a dice in turns to move one of his/her pieces forward to the goal. Each player also needs to use different policies to reduce his/her own pieces or enemy's pieces in order to win the game. Compared to other games, EinStein würfelt nicht! has smaller board size and fewer number of pieces. But it has a very important element, dice, that makes the game more complex, more fun and full of challenges and amazements.

This research will try to apply Monte Carlo method and convolutional neural network to explore different features and use different methods to combine these features to form different feature maps. Then base on these feature maps to train the weights of the neural network and expect that the new method can make the new EinStein würfelt nicht! program more powerful than the traditional approach.

Key words: computer games, EinStein würfelt nicht!, Monte Carlo method, neural network, deep learning

致謝

在這兩年中的碩士求學時期中，首先我要感謝林順喜教授的不辭辛勞、耐心地的指導與關懷，讓我這個從未踏入人工智慧領域什麼都不懂卻想踏入這個領域的學生能夠在這兩年間有所收穫。教授憑藉著自身多年豐富的經驗，常常能夠看出我沒有發現的問題核心，並且提出一些解決的可能方向給我去思考、嘗試，使我在許多方面的能力有所提升。

再來，我要感謝同個試驗室的夥伴們，讓我在沒有跟我同一個年級的研究生的情況下，不至於孤軍奮戰。這些夥伴中，第一個要感謝的是博士班的志宏學長，他在很多方面都有給予我有效的建議和幫助，這些建議和幫助在很多事情上對我的幫助非常大。第二個要感謝的是昱廷和俊緯學長，他們能夠友善地和剛進入這個實驗室的我對話，並給予建議，同樣地幫助我很多。第三個要感謝的是同個實驗室的學弟們，雖然我在年級上是你們的學長，但是在這個領域你們都是我的前輩，當我向你們詢問的時候，都會友善、耐心地向我解釋，並給予解決方向。

最後，我要感謝我的家人，能夠讓我心無旁騖地完成學業，因為家人們的包容與支持，我才能走到今天這一步。

目錄

圖目錄

表目錄

第一章　緒論

1.1 愛因斯坦棋與棋規

愛因斯坦棋[7]，是一個雙人骰棋對戰遊戲，棋盤的尺寸為 5x5，對弈的雙方分別各持藍、紅色棋子，雙方可以用任何方式決定哪一方做先、後手，每一盤棋結束後交換先後手順序。如圖 1-1 之棋局開始後，雙方輪流擲骰子且依骰子點數決定移動子，再從最多三個方向決定一個移動方向並且不可超出棋盤外(紅：右、下、右下，藍：左、上、左上)；若所擲到的點數的棋子已經被吃掉的時候，則從最接近所擲點數的棋子選其一，再選擇移動方向，例如：藍方擲骰子擲到點數 5，若點數為 5 的棋子還在棋盤上，則必須選擇點數 5 的棋子，並且從最多三個走步方向之內選擇其一移動；若點數 5 的棋子已經不在棋盤上，但是點數 4、6 的棋子還在棋盤上，則藍方可以從點數 4 或 6 的棋子做選擇移動。若移動的目的地上有棋子的話，不管敵我都會將位於目的地的棋子吃掉。

愛因斯坦棋的獲勝條件有二：

1. 殲滅對方所有棋子。
2. 己方任一個棋子比敵方早到達敵方陣地的角落(紅：棋盤右下角，藍：棋盤左上角)。

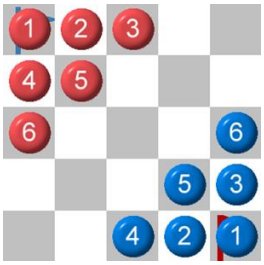

圖 1-1 愛因斯坦棋的棋盤

愛因斯坦棋相較於其他棋類遊戲較小的 5x5 的棋盤、較少的最大棋子數，表面上看起來會讓人以為是個複雜度沒有那麼高的棋類遊戲。但是，有一條特別的遊戲規則：擲骰子決定當回合可移動的棋子。

如何讓己方的走步選擇受骰子影響度降低、提高己方棋子整體走步靈活度，同時防止敵方走步受骰子影響度降低、限制敵方棋子整體走步靈活度提升幅度，將會是在棋局獲勝的重要關鍵。不過，此種戰術是一把雙面刃，有可能會發生敵眾我寡的盤面，這時若敵方採取殲滅我方棋子、一換一的策略，我方將陷於不利。因此，必須適當地、謹慎地控制敵我的棋子數量比、對盤面情勢的掌控、敵我分佈和敵我棋子間的間距，才能有效地發揮此戰術的優勢。

因為其特殊的骰子決定移動子的規則以及吃子、缺子情況對盤面情勢的影響，遊戲樹搜尋法若是以當前盤面情況規則評分和各個棋子走步的期望值作為判斷依據的話，我們很難窮舉出所有可能，並且給予最佳的評分組合。

此外，骰子點數對愛因斯坦棋盤面走勢影響甚鉅， 這也導致程式常常會選擇勝率較大的走步，而忽略了勝率小卻能扳回劣勢的險棋。

1.2 類神經網路的發展

類神經網路(neural network，簡稱 NN)，此名詞在數十年前就已經出現，其結構上，模仿了生物神經網路，利用了各個人工神經元間的互相連接，以傳遞演算法(propagation)和反傳遞演算法(back propagation)，調整儲存於每個人工神經元內的參數(權重)，此動作被稱之為「訓練」類神經網路，訓練完成的類神經網路，可以解決舊有的機器學習無法解決的問題，例如：異同問題(XOR)。

雖然因為類神經網路的出現，解決了一些難題，但是它有著嚴重的缺點，就是當問題較為複雜的時候，需要的參數(權重)也就越多，訓練類神經網路的過程所耗費的時間將會大大地增加，以當時的硬體技術，是無法克服的，因此，類神經網路這個技術，就如曇花一現般，很快地被打入冷宮。不過就在近十年，由於硬體技術的快速進步，改善了訓練類神經網路耗費時間過長的缺點，尤其是 GPU(Graphics Processing Unit)的出現，其在平行運算和單精度浮點數的運算上，較 CPU 具優勢，更加速了類神經網路的訓練效率。

近十年來，類神經網路逐漸在高科技產業中的人工智慧領域受到了重視，不過還尚未能夠吸引大眾的目光。直到 2016 年，人工智慧在許多領域達到了非常優異的成果，才受到眾人的矚目，如：IBM 的 AI Watson 正確診斷出「繼發性白血病」而救人一命、LipNet 唇語辨識率高達 93.4%已比人類一般專家的 52.3%高出許多、AlphaGO 以五戰四勝的成績贏過圍棋九段職業棋士李世石且於網路圍棋平台上完勝了眾多人類九段圍棋高手共 60 局、Libratus 與四位德州撲克世界頂尖高手對戰獲勝等。

因為人工智慧已經在許多領域有了非常好的成果，在解決一些問題的用途上異常強大，世界各國、許多人紛紛投入人工智慧的研究，而在人工智慧這個領域上扮演重

要角色之一的類神經網路更是如此。

1.3 TensorFlow

TensorFlow[8]是由其前身 DistBelief 演變而來。2011 年，Google Brain 建立 DistBelief 作為他們專有的第一代機器學習系統，其被用於建構深度類神經網路，被廣泛地應用於 Google 的各種服務中，例如：Google 搜索引擎、Google 語音搜索、推薦廣告等。

後來，DistBelief 由數學家 Geoffrey Hinton、Jeff Dean 簡化並重構，形成了一個更迅速、更強力的函式庫，即為 TensorFlow。

TensorFlow 作為一種開源的類神經網路開發模組，高階的部分以 python 構成，API 內的函式名稱較接近自然語言，方便使用者使用且增加開發效率。不過，在較底層的部分則是以 C 語言構成，以提高類神經網路的訓練效率。

TensorFlow 的優點是它是以「堆積木」的形式，讓使用者可以自由尋找所需要的零件，以自己資料的輸入輸出型態、輸入資料的特徵形式來設定欲構成的類神經網路形狀，再透過方便的 run 函式，即可開始訓練設計好的類神經網路，讓使用者跳過較底層的神經元鏈結的構成部份，增進開發效率。

此外，TensorFlow 亦支援 GPU，在訓練深度類神經網路的過程中，優化了大量調整參數(權重)的部分，其訓練效率是 CPU 的數倍。

1.4 監督式學習與增強式學習

在機器學習的多年演進下來，前人們發明了許多種方法，每種方法的使用條件、環境都不盡相同，在本研究中將會使用監督式學習與增強式學習兩種方法。

監督式學習，可視為數學上的迴歸分析或者是統計分析，簡而言之，對輸入的事物以給定的標準做分類。在學習、訓練的時候，我們必須給予欲訓練的對象多組輸入資料和相對應的標準答案。

增強式學習，輸入的所有資料都會對輸出結果產生影響，與監督式學習不同的是在此不給予欲訓練的對象標準答案，而是對其定義什麼是好的影響、擁有較高的效益。此方法被視為一種習慣性、適應性方法。此方法在很多地方被應用，被用來解決需要適應性的問題，以最為人所知的應用之一的掃地機器人為例，此種機器人最初只被賦予基本知識，如遇到障礙物要避開、牆壁要轉向等。由於每個使用者的住屋內配置都不盡相同，因此無法事先訓練如何以最短路徑打掃地板、計算回去充電的時間等，必須利用增強式學習的方式來「適應」使用者住屋的環境，以達到較大的效益。

第二章　文獻探討

由於愛因斯坦棋相較於其他棋類遊戲，是非常年輕的、相關的研究是較少的，還有很多方面、問題尚未被研究透徹。以下幾項文獻，是對本研究有所幫助的。

2.1 愛因斯坦棋算法設計與分析

"愛恩斯坦棋算法設計與分析"[1]這篇論文是由中國北京信息科技大學生李占宇於 2014 年所發表，該論文有提到關於作者對於戰略性讓己方、敵方缺子的看法：讓己方缺子讓己方整體移動選擇自由度增加、控制吃敵方子的時機以避免使敵方的移動選擇自由度增加等，還有對以移動子為主的 2x2、3x3 盤面評分判斷與相應的處理，如圖 2-1-1 所示，分析目前盤面情勢與移動子周圍的敵我分布狀況，並計算敵我的棋子分布間距，使程式能夠對不同狀況計算各別走步分數，再以貪婪演算法做決策。

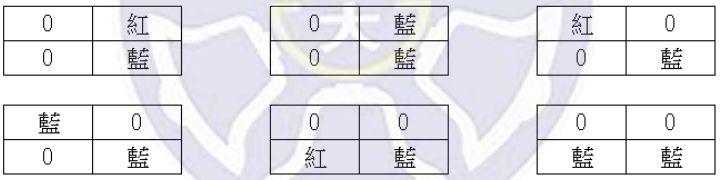

圖 2-1-1 2x2 盤面評分判斷依據(右下為移動子)

該論文在針對盤面情勢策略方面有獨特的見解，例如圖 2-1-2：

若現在是藍方的回合，可動子為 1，要往最接近終點的方向前進，有兩個候選方向：上、左上。該論文認為左上會是較好的走步方向，因為往上的方向會有在下一回合被紅方吃掉的風險，所以選擇「繞路」避開危險。

若現在是藍方的回合，可動子為 6，離終點較近的方向：左、左上。如上一個例子，

該論文會認為左上是較好的走步方向，不過這時會多了一個判斷依據：己方棋子數量與敵方棋子數量造成的整體靈活度差距。若 6 往左走，將紅方的 3 吃掉，這時雙方棋子數量比為藍：紅=3:1，紅方的 4 會處於非常優勢的狀態：不管骰子點數為何都能夠任意走動；相對地，藍方場上卻還有 3 個棋子，整體靈活度處於劣勢。

若現在是紅方的回合，可動子為 4，該論文會認為右下會是較好的走步，除了距離終點較近之外，往右吃掉 3 的話，由於目前藍方處於數量優勢，紅方很有可能被藍方殲滅。

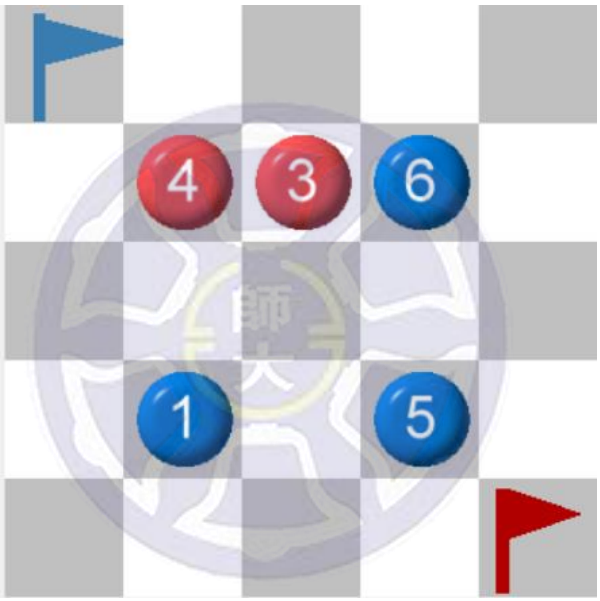

圖 2-1-2 愛因斯坦棋殘局盤面

2.2 手寫辨識作法與 TensorFlow

TensorFlow MNIST For ML Beginners[9]為 TensorFlow 官方網站的新手入門教學範例，以複雜度較低的數字手寫辨識來介紹 TensorFlow 的基本功能、使用方式以及強大之處。

該範例使用的是較簡單的 0~9 數字的手寫辨識。文內提到將每張手寫數字的圖片轉換為 28x28 的矩陣，如圖 2-2-1 所示，並且將圖中的每個像素(pixel)值大於零的像素(pixel)轉換為 1，其餘則為零。

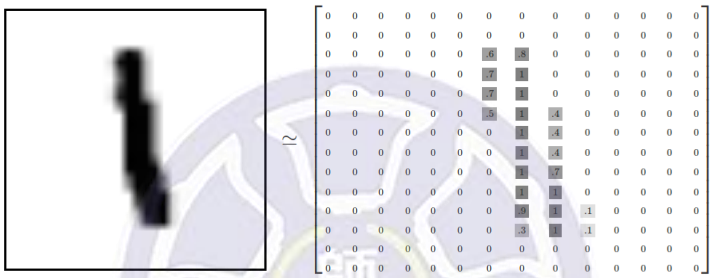

圖 2-2-1 手寫數字影像的資料轉換，取自[9]

範例文中的第一階段，利用 TensorFlow 建構出一個線性的類神經網路，再利用 gradient descent optimizer 對類神經網路內的參數(權重)做調整以減小誤差。此方式擁有 92%的辨識率。

第二階段則是改進第一階段的類神經網路，分成為五層的類神經網路的架構：兩層 convolutional layer、densely connected layer、dropout layer、readout layer。其中兩層 convolutional layer 和 densely connected layer 包含了 pooling layer 和 ReLU layer。整個網路的設計核心集中在了兩層 convolutional layer，本文中利用了 5x5 的權重矩陣對輸入的圖片做掃描，掃描方式為每次掃描只向右或向下移動一個 pixel，將 5x5 權重矩陣內的所有元素與目前掃描的圖片區塊內對應位置的 pixel 值

相乘之後再全部相加可得到一個值(若目前掃描的 5x5 區域超出圖片範圍，則將那些位置補零)，如此掃描完整張圖片之後，對每一個得到的值加上一個偏差值(bias)，利用 ReLU 函數，如圖 2-2-2，將所有小於零的元素令為零，即可以得到一張特徵圖(feature map)。不同的 5x5 的權重矩陣依照上述的步驟，都可以得到不同的特徵圖。接下來，將這些特徵圖經過 pooling layer，在 pooling layer 中，會使用 2x2 的矩陣，每次掃描只向右或向下移動兩個 pixel，並從 4 個元素值中，挑選最大之值並保留，掃描完整張特徵圖後，保留下來的值即會形成新的圖片，此時這些新的圖片的尺寸長寬為輸入圖片的二分之一。

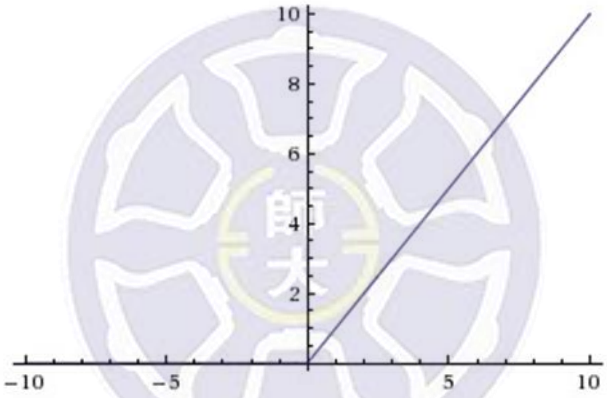

圖 2-2-2 Relu function: h = max(0, y), where y = Wx + b

例如，若初始輸入的圖片為 28x28，經過一層的 convolutional layer 之後，將會變為 14x14、輸出 32 張 feature map，經過第二層 convolutional layer 之後，會變為 7x7、輸出 64 張 feature map。雖然圖片尺寸逐層變小，但是相對地節點個數會增加。

經過了兩層 convolutional layer 後，接下來會進入 densely connected layer，又可稱為 completely connected layer，在此階段，輸入的圖片已經變為尺寸 7x7，由於上一層 convolutional layer 總共輸出了 64 張圖片，這 64 張圖片會被變形

(reshape)為 7x7x64 的矩陣向量，與相同尺寸的權重矩陣(feature)作相乘，並加上偏差值，產生出 1024 個輸出。

Dropout layer 只有在 training 的時候才會使用，為了預防 overfitting，又稱為過度學習，避免較不好的 training data 影響類神經網路內的參數調整。可以自由調整 keep_prob 參數使在訓練的過程中，無視誤差過大的 training data 的訓練結果，例如，設定為 0.5 的時候，若每次讀取 10 筆 training data 作訓練，會捨棄 5 筆產生最大誤差的訓練結果。

Readout layer 為輸出層，輸入數量為 1024 個，輸出為 10 個。利用尺寸為 1024 的矩陣向量對輸入作矩陣相乘，並加上偏差值(bias)，產生 10 個輸出。這 10 個分別代表著最後辨識結果每個數字的機率。

此階段改為使用 Adam optimizer 對類神經網路內的參數(權重)做調整以減小誤差。在此階段，辨識率大幅提升至 99.2%。

2.3 愛因斯坦棋平台

愛因斯坦棋平台[2]由國立台灣師範大學數學系謝昌龍製作，如圖 2-3 所示。其提供一個便利的圖形操作介面，且具備常用的功能，例如各種條件的愛因斯坦棋下棋機能、記譜、讀譜功能，而使用者只須撰寫 AI 引擎程式，透過標準輸入輸出一些簡單的文字，就能與該平台溝通，且可輕易抽換自己所撰寫的 AI 引擎程式。

透過此平台，可以讓兩個程式大量互相對弈，以之產生大量可供類神經網路訓練的棋譜。

兩個有近似或相同棋力的類神經網路程式的大量對弈，可以有效率地進行 training data 的提取，保留較好的盤面及其走步，藉以提升類神經網路的精準度。

圖 2-3　愛因斯坦棋介面

2.4 蒙地卡羅演算法

"An MCTS program to Play EinStein Würfelt!"[5]此論文使用蒙地卡羅演算法（Monte Carlo method），示意圖如圖 2-4。它是一種數值方法，利用隨機大量模擬的方式，可以得到一個盤面情況的近似解。

不過，此方法當遇到搜尋廣度、深度過大的問題的時候，能夠計算出精準的結果的機率就越低。因此本研究將訓練出一個對策類神經網路，輔助蒙地卡羅演算法，在每個模擬節點皆會給出較好的走步方向，藉此提高蒙地卡羅演算法模擬的精準度。

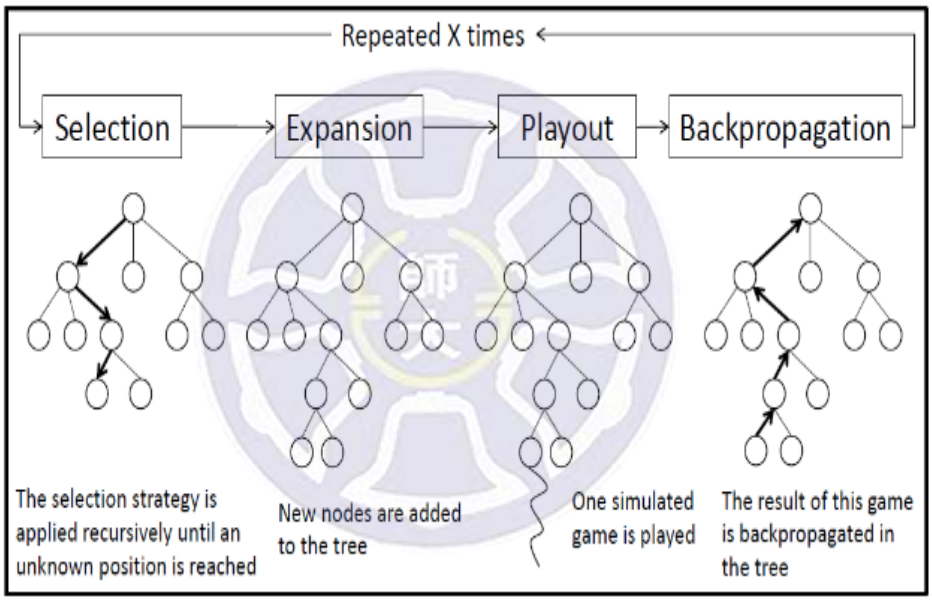

圖 2-4 MCTS 示意圖，取自[5]

2.5 AlphaGo 的網路結構與特徵擷取

“Mastering the game of Go with deep neural network and tree search”[6]該篇論文於 2016 年發表於 Nature 論文期刊上，論文內說明了 AlphaGo 圍棋下棋程式部分實作方式。

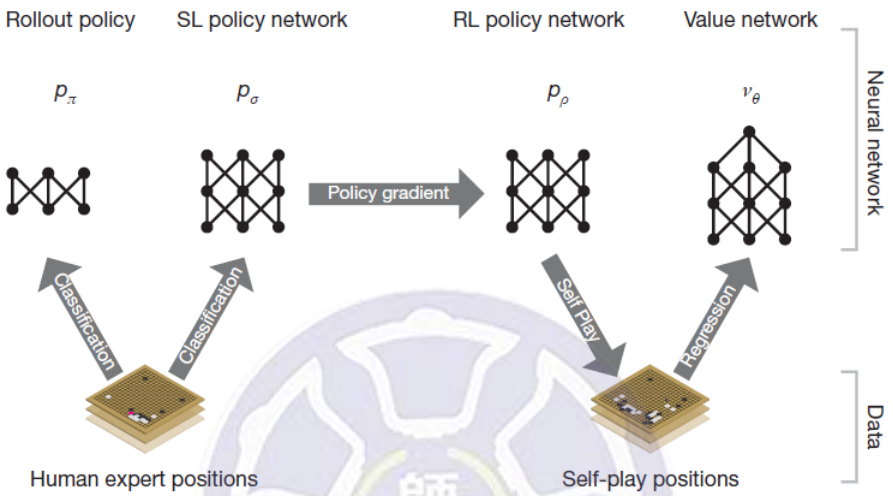

圖 2-5-1 AlphaGo 類神經網路訓練結構，取自[6]

如圖 2-5-1 所示，蒐集大量的人類圍棋專家的棋譜資料，將這些資料初步分類優劣、經過前處理，以取得數個特徵(Feature)，如圖 2-5-2 所示。

以圖 2-5-2 的特徵將大量的棋譜盤面作前處理(pre-processing)，使一個棋譜盤面產生 12 個結果盤面。再利用圖 2-5-2 的特徵(除了最後一項 Player color)，訓練出監督式學習的對策類神經網路(SL『Supervised Learning』policy network)。再以其作為核心的兩個圍棋程式互相對弈，利用增強式學習(Reinforcement Learning)的方式，汰弱留強，可以得到增強式學習的對策類神經網路(RL『Reinforcement Learning』policy network)。再使增強式對策類神經網路的程式互相對弈，利用圖 2-5-2 的訓練

特徵，最後得出對整個圍棋盤面的估值類神經網路(Value network)。

程式在對弈的時候，對策類神經網路所擔任的任務為「減少搜尋的廣度」，改善了當蒙地卡羅演算法應用在圍棋上時，模擬廣度過大的問題。將一個目前盤面輸入對策類神經網路後，它會回傳數個推薦落子位置，而這些推薦落子位置即是由大量的人類圍棋專家的棋譜中所可能出現的走步。此方法可以篩選掉大部分的落子選擇，只留下較好的。

再來，由於圍棋的搜尋廣度、深度非常深，如果只使用蒙地卡羅演算法的話，亂下模擬的準確度會非常低，故利用 rollout 對策網路快速地篩選較好的走步，減少模擬廣度，再利用估值類神經網路，對這些較好的走步作評分，如此可以提高蒙地卡羅演算法的模擬準確率。

AlphaGo 透過此方法，征服了被視為人類棋類遊戲最後的壁壘之一的圍棋，擁有了相當於人類職業九段棋士的棋力。

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

Feature planes used by the policy network (all but last feature) and value network (all features).

圖 2-5-2 AlphaGo 類神經網路的訓練特徵，取自[6]

2.6 蒙地卡羅樹搜索演算法和 Upper Confidence Bound

“愛因斯坦棋的電腦棋類程式設計”[3]論文發表於 2017 年 Taiwan Computer Game Association(TCGA)研討會，作者為交通大學資訊工程學系楊君亮、許庭嫣、林立秦。

首先，此論文中提到利用蒙地卡羅樹搜索演算法(MCTS)搭配上 upper confidence bound(UCB)來使 MCTS 在隨機模擬的時候，能夠增加雖然目前勝率不高，但有潛力是好的走步的樹節點被模擬到的機會。

再來，此論文中提到了五種 MCTS 模擬的方法：

1. 利用規則導向、經驗法則，對棋盤各個位置、敵我雙方的位置分以及是否採取防守策略作評分，並找出一個公式將這些方法所得出的分數作總合。
2. 隨機模擬方法(一)：先找出所有當前節點的盤面狀態的合法步，再從這些合法步中隨機取出一步進行模擬。其中在隨機挑選的過程中，每一個合法步被選到的機率皆相等。
3. 隨機模擬方法(二)：先模擬擲出骰子隨機得到的骰子點數，由可移動的棋子提出所有的合法步，再從中隨機取出一步進行模擬。其中在同一骰子點數的情況下，每一個可移動的合法步被選到的機率皆相等；而每一個骰子點數被隨機選到的機率也相等。
4. 隨機模擬方法(三)：先隨機取一顆棋子，若該棋子不在棋盤上，則再次隨機選取、直到被選到的棋子是存活的；確定要移動的棋子後，再隨機選取要移動的方向，直到確定該步棋為一合法步後進行模擬。
5. 隨機模擬方法(四)：先模擬擲出骰子隨機得到的骰子點數，由可移動的棋子隨

機選出一顆棋子，將該棋子所有的合法步提出，並從中隨機選一步進行模擬。

該論文分別使用 min-max 樹搜索演算法和 MCTS 來測試這五種模擬方法的優劣。其中，該論文發現在 min-max 樹搜索演算法中，四種隨機模擬方法的勝率差異不大，隨機模擬方法(三)在先手的時候較有優勢；而隨機模擬方法(四)在後手的時候較強勢。在 MCTS 中，四種模擬方法勝率差異不大，不過隨機模擬方法(四)相較其他方法勝率較高。

2.7 矩形特徵擷取和時序差異學習

“愛因斯坦棋人工智慧”[4]論文發表於 2017 年 Taiwan Computer Game Association(TCGA) 研討會，作者為交通大學資訊工程學系朱詠嘉、陳源灝。

首先，該論文中提到從棋盤盤面取出矩形特徵，如圖 2-7-1 所示，並將特徵中的每一個棋子編號的位置並且將之轉換成下一回合可以移動的機率。每一組特徵有兩個 label 值，該特徵出現的總次數(total)和該特徵出現且在該局棋勝利的次數(win)。

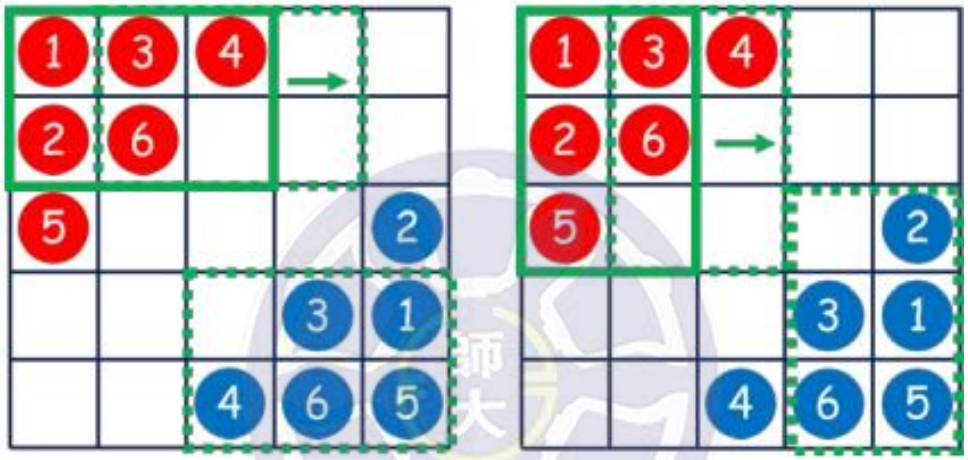

圖 2-7-1 2x3 及 3x2 特徵示意圖，取自[4]

再來，利用時序差異學習，如圖 2-7-2 所示，一局棋從 S_0 開始，經過許多個回合 S_t，到達最終分出勝負 S_T。記錄整局棋所有出現過的特徵，贏方所有在這局棋出現過的特徵的 total 值加一、win 值加一；反之，total 值加一，win 值保持不變。

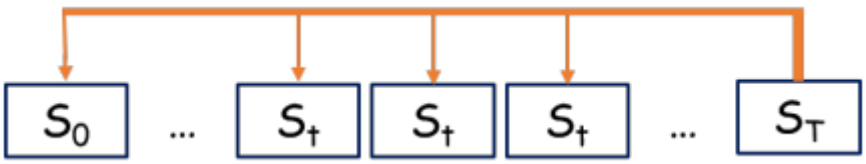

圖 2-7-2 時序差異學習示意圖，取自[4]

利用以上的方法可以得到每一種特徵的勝率，以之對 UCB 做變形，如以下公式：

$$UCB_i + C_p \times \frac{H_b}{n_i}$$

其中，Cp 為常數、Hb 為時序差異學習所得之結果、ni 為 i 節點被訪問的次數。

當開始做蒙地卡羅樹搜索模擬的時候，利用時序差異學習所得之結果來初始化欲模擬之節點的勝率，並利用新的 UCB 公式作為選取模擬分枝的依據，避免模擬初期的盲目選枝。

以此方法實作的下棋程式在跟模擬次數較少的 MCTS 下棋程式對弈的時候，能夠佔很大的優勢；不過，在跟模擬次數較多的 MCTS 下棋程式對弈的時候，就顯示出兩者已棋力相當。此結果代表著，「預訓練」能夠彌補模擬次數所造成的模擬準確度的差距。

第三章　程式設計方向

以蒙地卡羅演算法構成的程式，透過隨機大量亂下的方式產生走步，此程式與目前最頂尖的程式 meodero 對戰，擁有約 38%的勝率。

由於舊有的下棋程式常用的演算法大多為利用對敵我雙方的每個棋子的移動期望值做精算，搭配上規則導向對盤面棋型分布局勢做評分，來決定走步的選擇。此種方式若能找到一個完美的規則評分組合，會是非常強力的方法。但是，大部分的評分規則無法達到如此完美的程度，因為當一個問題的複雜度、維度很高的時候，我們很難窮舉出所有可能發生的狀況，難免會有所缺漏。

因此，我們從知名線上遊戲平台 Littlegolem 上抓取排名(Rank)前十名玩家的棋譜，再將這些棋譜做前處理產生特徵盤面，以作為對策類神經網路訓練之用。嘗試以機器學習的大量運算、不眠不休的優勢，訓練出能夠對愛因斯坦棋盤面做精準走勢評分的對策類神經網路以改善舊有演算法的不足之處。

在對策類神經網路的訓練上，我們建構出兩個結構不相同的 convolutional neural network(CNN)，並以特徵相同但表示方式不同的 training data 來當作 CNN 的輸入特徵盤面，互相變換搭配的組合，以觀察類神經網路結構的複雜性與特徵表示方式對訓練準確率的影響。

最後再以此對策類神經網路輔助蒙地卡羅演算法，讓此類神經網路作為在遊戲樹淺層展枝中選擇分枝的依據，減少模擬廣度，增加模擬深度節點時的準確率，期望改善蒙地卡羅演算法的缺點。

第四章　程式實作

4.1 棋譜的盤面提取

參考自 AlphaGo 實作方法的圍棋特徵擷取，愛因斯坦棋的特徵可分為骰子點數、顏色、敵我分佈、棋子前方的棋子數量。本研究將這些特徵轉換為尺寸為 25 的一維陣列作為類神經的網路的輸入。

首先，為了從大量的棋譜中提取較好的走步，我們將一場棋局的贏家的所有走步假設為較好的走步，並且只提取贏家的走步及其當前的盤面。

再來，我們將提取下來的走步及盤面以文字檔儲存下來。再從中提取並分類出各種資訊，以作為 training data 原始檔案，分類出的資訊共分為四類：當前回合走子方的顏色、當前回合的骰子點數、棋盤上走子方走步前的藍色棋子位置及紅色棋子位置、走子方選定欲走棋移動前的位置及移動的目的地。其儲存格式如下：

(顏色)(骰子)(藍色位置)(紅色位置)(欲走棋位置)(目的地)

除此之外，我們將棋盤上的每個位置都給予一個編號，如圖 4-1-2 所示，以便紀錄所有棋盤上的資訊。

如圖 4-1-1 所示的棋盤盤面(紅方的回合、骰子點數 5)，儲存成文字檔後顯示如下：

(R)(5)(25 24 20 23 19 15)(1 2 3 6 7 11)(7)(13)

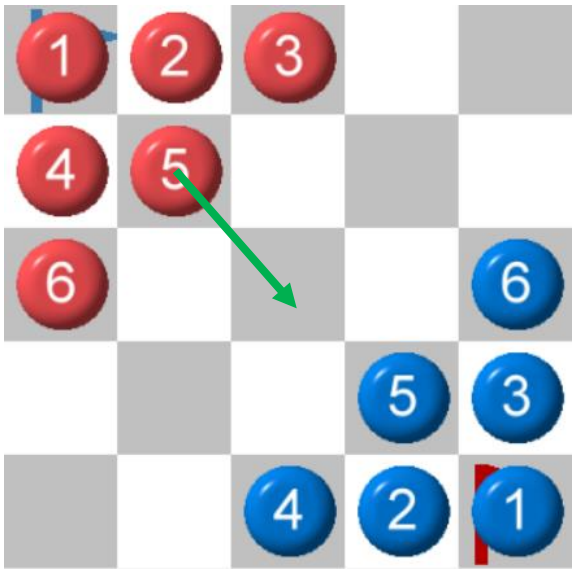

圖 4-1-1 愛因斯坦棋初始盤面

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

圖 4-1-2 愛因斯坦棋棋盤位置編號

4.2 原始盤面的特徵擷取

首先，參考手寫影像辨識的做法，將整個棋盤視為一張圖片，棋盤中的每一個位置視為圖片中的 pixel。再以此為原則，將原始盤面轉換為不同的特徵盤面以作為 training data 之用。為了讓骰子點數這個對棋局影響甚大的因素能夠融入棋盤中，我們將骰子點數以「可動子的概念」代替，並且以「負號」表示。然後，雖然藍紅雙方各有編號 1 至 6 的棋子，但是同樣編號的棋子必須視為不同的棋子，因此我們將藍色棋子以 1, 2, 3, 4, 5, 6 表示、紅色以 7, 8, 9, 10, 11, 12 表示。

再來，我們在 2.1 節中有提到，棋盤上及每個棋子前方的敵我分佈的狀況將會大大地影響棋局的勝負。我們將此特徵分離成兩個不同的特徵，其中整個棋盤的敵我分佈的特徵是以藍色為 1、紅色為-1、空格為 0 表示。

最後， 我們將棋子前方敵我分佈的狀況定義為「每個棋子前方九宮格內的包括自己的棋子數量」，以藍色方的棋子為觀點時，棋子位於九宮格的右下角，如圖 4-2-1 所示；以紅色方的棋子為觀點時，棋子位於九宮格的左上角，如圖 4-2-2 所示；若九宮格範圍超出棋盤外時，則忽略超出的範圍，如圖 4-2-3 所示。

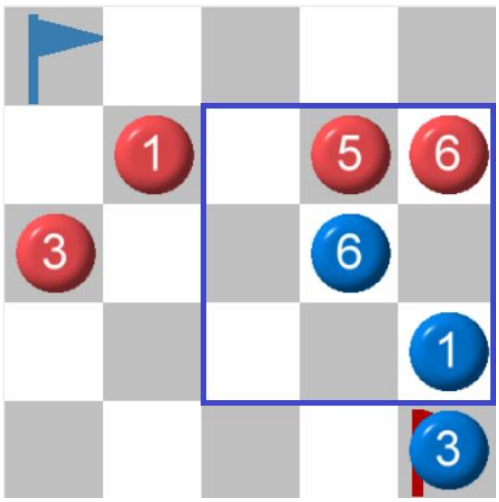

圖 4-2-1 棋子前方的棋子數量定義示意圖(藍色 1 號棋觀點)

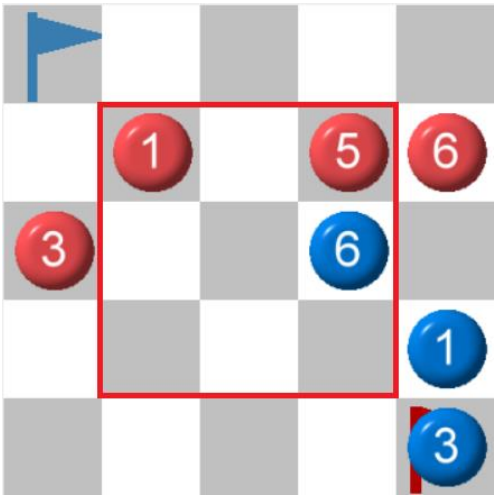

圖 4-2-2 棋子前方的棋子數量定義示意圖(紅色 1 號棋觀點)

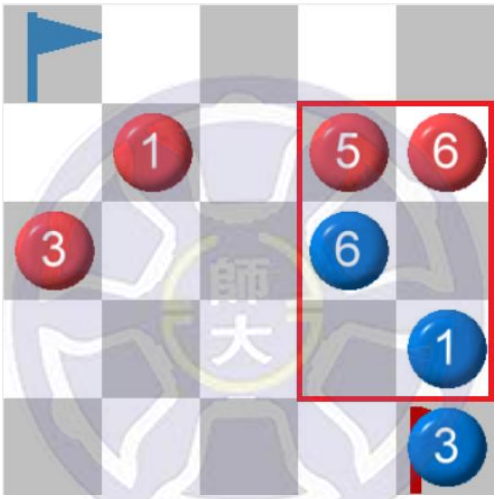

圖 4-2-3 九宮格超出棋盤的處理方式(紅色 5 號棋觀點)

以圖 4-2-1 的盤面、當前為藍方回合及骰子點數為 2 的狀態，做特徵擷取並轉換後，將會產生出三種特徵盤面，我們以三個大小為 25 的陣列表示，見圖 4-2-4：

```
0 0 0 0 0 0  7 0 11 12  9 0 0 6 0 0 0 0 0 -1 0 0 0 0 -3
0 0 0 0 0 0 -1 0 -1 -1 -1 0 0 1 0 0 0 0 0  1 0 0 0 0  1
0 0 0 0 0 0  2 0  4  2  1 0 0 3 0 0 0 0 0  4 0 0 0 0  3
```

圖 4-2-4 轉換後的特徵盤面

4.3 擷取特徵簡化轉換

我們會將在 4-2 節從原始盤面提取出來的三種特徵盤面，在此做簡化的的轉換，將所有特徵資訊以 0 或 1 表示。

圖 4-2-4 第一列的特徵盤面經過簡化轉換後，會被分為兩個部分共 13 種大小為 25 的特徵盤面。其中 12 種分別表示藍紅雙方棋子是否還在棋盤上且於盤面上的位置，若棋子已經不在棋盤上則此特徵盤面全為 0；反之，則在其對應的位置上標示為 1。另 1 種則表示當前盤面的可動子，將可動子的位置標示為 1。

圖 4-2-4 第二列的特徵盤面經過簡化轉換後，分別由 2 種大小為 25 的特徵盤面，分別將藍、紅雙方棋子的對應位置標示為 1。

為了將圖 4-2-4 第三列的特徵盤面簡化，我們利用圖 4-3-1 的棋子前方九宮格內敵我分佈狀態關係、4-3-2 的棋子各方向敵我狀態關係，若棋盤上的棋子符合此關係，則在其對應位置標示為 1。依此方式分別會將其分為 22 個特徵盤面、6 個特徵盤面。

0	0	紅
0	0	紅
0	0	藍

紅	0	0
0	紅	0
0	0	藍

0	0	0
0	0	0
紅	紅	藍

0	藍	0
0	0	0
0	0	藍

0	0	藍
0	0	紅
0	0	藍

藍	0	0
0	紅	0
0	0	藍

0	0	0
0	0	0
藍	紅	藍

0	紅	0
0	0	0
0	0	藍

0	0	紅
0	0	藍
0	0	藍

紅	0	0
0	藍	0
0	0	藍

0	0	0
0	0	0
紅	藍	藍

0	0	0
藍	0	0
0	0	藍

0	0	藍
0	0	藍
0	0	藍

藍	0	0
0	藍	0
0	0	藍

0	0	0
0	0	0
藍	藍	藍

0	0	0
紅	0	0
0	0	藍

0	0	藍
0	0	0
0	0	藍

藍	0	0
0	0	0
0	0	藍

0	0	0
0	0	0
藍	0	藍

0	0	紅
0	0	0
0	0	藍

紅	0	0
0	0	0
0	0	藍

0	0	0
0	0	0
紅	0	藍

圖 4-3-1 九宮格內棋子各方向敵我分佈狀態關係(藍方觀點)

0	0	0
0	0	藍
0	0	藍

0	0	0
0	藍	0
0	0	藍

0	0	0
0	0	0
0	藍	藍

0	0	0
0	0	紅
0	0	藍

0	0	0
0	紅	0
0	0	藍

0	0	0
0	0	0
0	紅	藍

圖 4-3-2 棋子前方各方向敵我狀態關係(藍方觀點)

4.4 類神經網路的輸出

由於愛因斯坦棋每一回合的走步選擇數量最多為 6，因此在對應輸出方面以每一列 6 個 0 或 1 的元素表示，1 代表使用蒙地卡羅演算法的程式所選擇的走步，表示的順序為水平、垂直、斜向。

若當前盤面只有一個可動子的時候，則只會使用每一列的前三行的元素，後三行則必為零。這個資訊代表的意義為在當前盤面以及可動子的狀況，程式選擇走步的勝率，可視為監督式機器學習的標準答案。為當我們將盤面狀況、可動子狀況輸入類神經網路之後，我們期望得到的理想結果。

圖 4-4 為圖 4-2-1，藍方回合，骰子點數為 2，蒙地卡羅模擬出的結果為藍色 3 往上移動一格的結果：

0 0 0 0 1 0

圖 4-4: 類神經網路輸出範例

4.5 卷積式類神經網路(convolutional neural network, CNN)

利用 TensorFlow 建構出一個擁有 25 個輸入，6 個輸出值的類神經網路，其中 25 個輸入值分別代表特徵盤面棋盤上的 25 個位置情況，6 個輸出值代表在此特徵盤面的情況下，每個走步的獲勝機率。

本研究分別設計了兩個在 convolutional layer 層數不同、每一層輸出 feature map 數量不同的類神經網路，如圖 4-5-1、4-5-2 所示，來對經過前處理過後的 training data 做訓練，以觀察準確率分布的結果。

此兩種架構的類神經網路分別共有兩層和三層 convolutional layer，其中分別以 2x2、3x3 以及 2x2、3x3、4x4 的權重矩陣(weight matrix)作為 filter，對已經過前處理的特徵盤面或 feature map 做掃描，以之作為下一層類神經網路的輸入。

第一種共有兩層 convolutional layer。首先，參考自圖 4-3-1，第一層 convolutional layer 中作為 convolutional filter 的 weight matrix 的大小為 2x2、每一個 input feature 會使用 6 個 filter 做掃描，產生 6 個大小同為 5x5 的 feature map 作為下一層 convolutional layer 的 input。再來，參考自圖 4-3-2，第二層 convolutional layer 中作為 convolutional filter 的 weight matrix 的大小為 3x3，每一個 input feature map 會使用 22 個 filter 做掃描，產生 22 個大小同為 5x5 的 feature map 作為下一層的 completely connected layer 的輸入。

第二種共有三層 convolutional layer。參考自手寫辨識的範例，我們將每一層 convolutional layer 的 filter 數量分別設定為 40、80、160。第一層每一個 input feature 會產生 40 個 feature map，第二、三層分別會產生 80、160 個 feature map。

掃描的方式為 weight matrix 每次移動只棋盤一格並對目前掃描的區域做內積

(multiply-summation)，再以 Rectified Linear Unit(ReLU) function 為激勵函數(activation function)，將小於 0 的值變更為 0，大於等於 0 的值則保持不變。以此方式掃描完整張棋盤即可產生新的 feature map，而此 feature map 將會是下一層類神經網路的輸入。

經過了所有的 convolutional layer 之後，連接上 completely connected layer，亦可稱為 densely connected layer，在此會將所有 feature map 轉換成一個一維矩陣，且為一個線性的類神經網路結構。

completely connected layer 之後，是 drop layer ，此層只會在訓練的時候使用，功能是用來防止 overfitting 的發生，在訓練的時候會以設定好的比率捨棄誤差過大的結果。在絕大部分的 training data 為良好的情況下，此方法可以避免在訓練的時候，遇到那極少數不良的 training data 而影響類神經網路內的權重參數的情況。

最後的是 output layer，此層是一個 completely connected 的線性類神經網路，在這裡會將 completely connected layer 的輸出轉換為我們想要的輸出型式，在此會有 6 個輸出值，分別代表著最多六個走步中，是較好走步的機率。

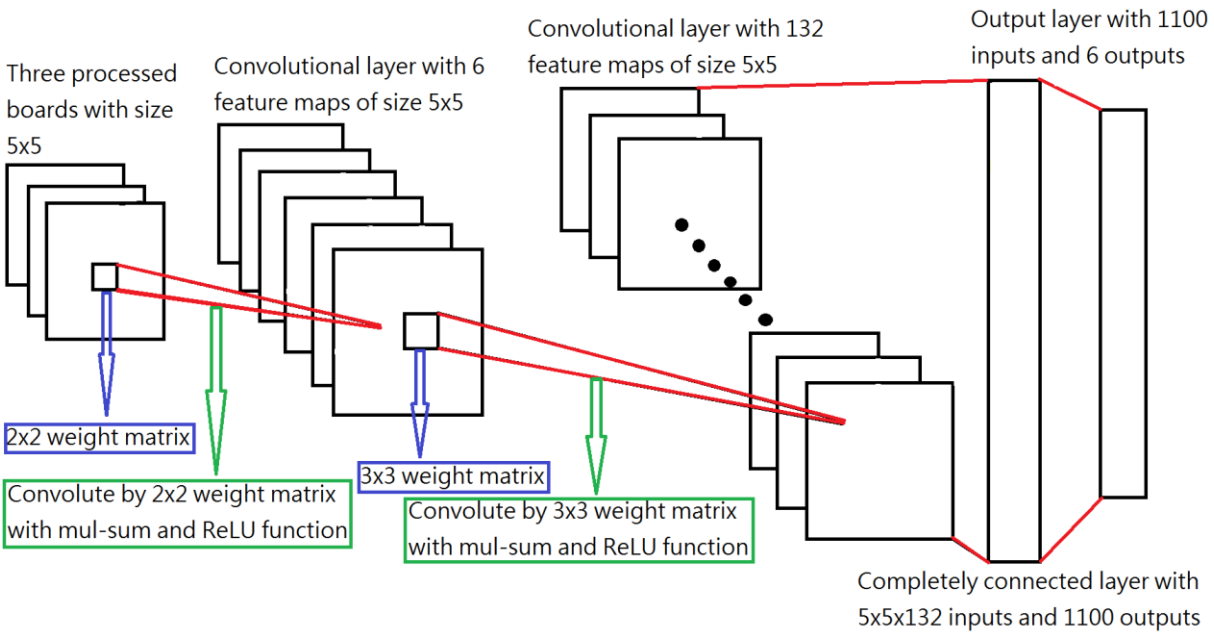

圖 4-5-1 類神經網路架構一

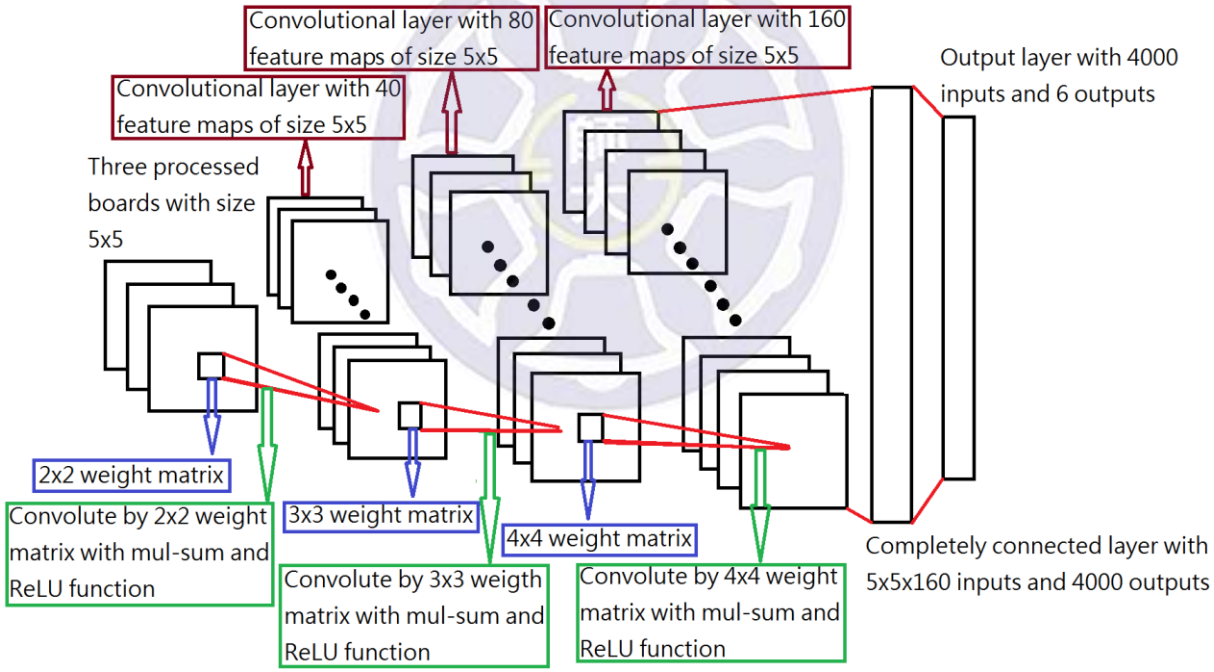

圖 4-5-2 類神經網路架構二

4.6 愛因斯坦棋下棋程式的架構

首先，在展枝的過程中，展開第一層時，程式是已知當前盤面及骰子點數的狀況，而第二層以下的展枝時，則是當前盤面已知、骰子點數未知的狀況。因此，當程式要展開第一層的時候，只需要根據當前盤面及骰子點數決定的可動子窮舉出所有走步並展開第一層未來可能的盤面。

再來，展開第二層以下的時候，程式會找出欲做展枝的盤面在不同骰子點數的情況下的可動子組合以及其機率權重。

最後，將當前做欲展枝的盤面、骰子點數 1 至 6 產生的所有可動子組合的集合等資訊轉換成符合已訓練完成的類神經網路的輸入形式。轉換完成後，每一組可動子會產生一組特徵盤面，將每一組特徵盤面分別輸入進類神經網路中，會分別得到一組輸出值，程式會選取數值(勝率)最大者做向下展枝的動作。

依照以上的做法，如圖 4-6 所示，在奇數層以我方的觀點產生特徵盤面、偶數層以敵方觀點產生特徵盤面，到達指定的深度的時候，再利用蒙地卡羅演算法對其作隨機亂數模擬，再將所有模擬結果經過每個節點的可動子機率加權後，加權方法如表 4-6-1、4-6-2、4-6-3 的範例所示，整合至上層分支，以取得當前盤面各走步的勝率，作為走步的選擇依據。

棋子編號	1	2	3	4	5	6
是否存在	T	T	T	T	T	T
權重分配	1	1	1	1	1	1

表 4-6-1 未缺子的權重分配

棋子編號	1	2	3	4	5	6
是否存在	T	F	T	T	F	T
權重分配	2	0	2	2	0	2

表 4-6-2 缺子權重分配範例(缺 2、5)

棋子編號	1	2	3	4	5	6
是否存在	T	F	F	T	F	T
權重分配	3	0	0	4	0	2

表 4-6-3 缺子權重分配範例(缺 2、3、5)

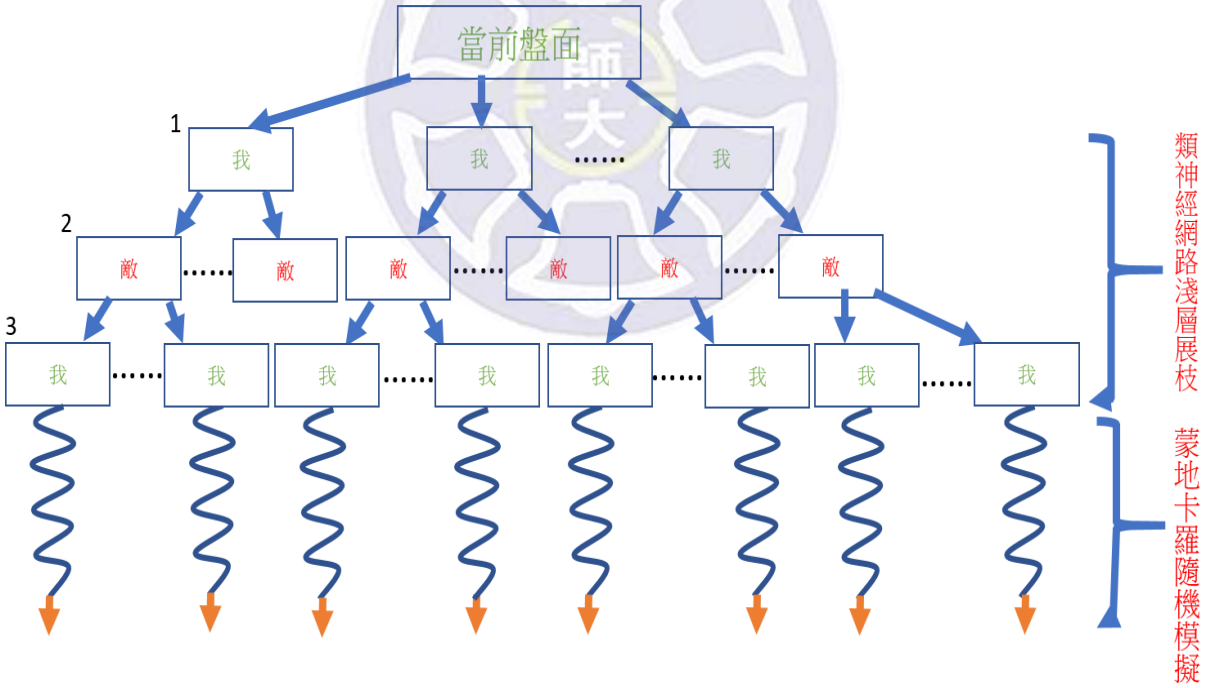

圖 4-6 愛因斯坦棋 AI 下棋程式架構圖

第五章 實驗與結果

本實驗共分為三個部分，分別就 feature 組合的方式、feature 表示方式以及 convolutional neural network 每層 filter 的數量這三方面來做實驗。實驗使用的類神經網路是使用圖 4-5-1、圖 4-5-2 的 CNN 架構，每一組實驗的設定為：training data 為 258042 筆、訓練次數為 200 萬次、每次 50 個盤面、最初 10 萬次訓練忽略不計。

5.1 input feature 組合方式

本實驗嘗試使用不同組合的 input feature，觀察在不同 input features 組合的情況下於圖 4-5-1 及圖 4-5-2 兩種 CNN 架構下對走步判斷的準確度。使用的特徵如 4.2 節所述，共分為三項：

1. 雙方棋子在棋盤上的位置及當回合的可動子，不可或缺的 input feature。
2. 雙方顏色於棋盤上的分布。
3. 棋子前方九宮格內棋子數量(若為簡化表示法，則為前方各方向敵我分布狀態)。

表 5-1-1、表 5-1-2 為在 training 過程中，對 training data 的準確率，由 training 的過程中，準確率有發生浮動的現象，因此紀錄其準確度的分布情形。

項目	<20%	20%<=acc<25%	25%<=acc<30%	30%<=acc<35%	35%<=acc<40%	>=40%	*<30%*	*>=30%*
ori#1#I	0.00%	1.32%	16.18%	33.03%	40.66%	8.82%	*17.50%*	*82.50%*
ori#1.2#I	0.00%	2.37%	19.74%	40.79%	32.11%	5.00%	*22.11%*	*77.89%*
ori#1.3#I	0.00%	4.21%	18.42%	35.79%	34.21%	7.37%	*22.63%*	*77.37%*
ori#1.2.3#I	0.00%	1.75%	18.16%	37.54%	33.68%	8.86%	*19.91%*	*80.09%*

表 5-1-1 I-CNN 架構、各 ori input feature 組合準確度分布表

項目	<20%	20%<=acc<25%	25%<=acc<30%	30%<=acc<35%	35%<=acc<40%	>=40%	*<30%*	*>=30%*
sim#1#II-40	0.00%	3.42%	18.68%	40.79%	30.79%	6.32%	*22.11%*	*77.89%*
sim#1.2#II-40	0.00%	3.68%	22.63%	43.95%	27.11%	2.63%	*26.32%*	*73.68%*
sim#1.3#II-40	0.27%	3.98%	20.16%	31.56%	35.81%	8.22%	*24.40%*	*75.60%*
sim#1.2.3#II-40	0.00%	4.21%	21.58%	33.42%	33.68%	7.11%	*25.79%*	*74.21%*

表 5-1-2 II-40-CNN 架構、各 sim input feature 組合準確度分布表

在以上兩表的項目欄位中，ori 和 sim 分別代表原始盤面特徵擷取及將其簡化成 0 或 1 的表示方式，1、2、3 為上述特徵盤面的代號，I、II-40 分別代表圖 4-5-1 的 CNN 架構和圖 4-5-2 的 CNN 第一層 convolutional layer 有 40 個 filter 的架構，acc 代表在 training 的時候，對 training data 的準確度，最右側粗斜體字的兩欄分別代表準確率小於 30%及大於等於 30%佔訓練總次數的百分比。

首先，由此兩組數據，可以看到在只有特徵盤面 1 的情況下，兩種 CNN 的架構的訓練準確率皆是最高的。分別加入特徵盤面 2、3 後，I 和 II-40 的訓練準確率的分布有些許降低。其中，相較於 sim#1.2#II-40、sim#1.3#II-40，ori#1.2#I、ori#1.3#I 的下降幅度較大，不過 ori#1.2#I 和 ori#1.3#I 下降幅度非常接近，sim#1.2#II-40 的下降幅度較 sim#1.3#II-40 大。

再來，由 ori#1.2.3#I 及 sim#1.2.3#II-40 的數據中，可以看見在 I 的 CNN 架構下，當 2、3 兩種特徵盤面和 1 特徵盤面搭配作為 input feature 的時候，準確率較高的次數百分比有上升，不過還是比原本只有 1 特徵盤面時所佔的次數百分比低；而在 II-40 的 CNN 架構下，sim#1.2.3#II-40 準確率較高的所佔百分比較 sim#1.3#II-40 低。

由以上結果可發現，在 I 的 CNN 架構下，2、3 兩種特徵盤面以 ori 的方式表示的情況下，單獨和 1 特徵盤面使用的時候會造成訓練的效果降低，兩者搭配使用的時候，效果會上升。由此我們推斷 I 的 CNN 架構無法完全學習到 2、3 兩種特徵盤面所給予的資訊；不過，由表 5-1-1 的 ori#1.2#I、ori#1.3#I 及 ori#1.2.3#I 三組數據可以得知

2、3 特徵盤面有相輔相成的效果。在 II-40 的 CNN 架構下，2、3 兩種特徵盤面以 sim 的方式表示的情況下，跟在 I 的 CNN 架構的結果一樣，單獨 1 特徵盤面使用的時候，訓練效果會降低；不同的是，在 2、3 兩種特徵盤面同時與 1 特徵盤面搭配時，效果較 1、3 兩種特徵盤面搭配的效果差。由此我們推斷 2 特徵盤面的表示方式有不適合之處，代表藍、紅兩色棋子於棋盤上位置的特徵盤面在 sim 的表示方式下，被分解為兩個分別表示藍色和紅色的特徵盤面，且皆以 1 表示，這可能導致 CNN 在訓練的過程產生對此特徵盤面的誤解，誤認為藍、紅兩方是等價的、沒有敵我關係而產生錯誤的判斷。

5.2 feature 表示方法

本實驗將利用 II 的 CNN 架構中，調整每一層 convolutional layer 的 filter 數量，藉以觀察在不同的 input feature 表示方法的情況下，其準確率的分布情形。II 後面的數字代表著第一層 convolutional layer 的 filter 的數量，其他 convolutional layer 的 filter 數量為上一層的兩倍。

項目	<20%	20%<=acc<25%	25%<=acc<30%	30%<=acc<35%	35%<=acc<40%	>=40%	*<30%*	*>=30%*
ori#1.2.3#II-10	0.26%	3.16%	22.37%	35.26%	34.21%	4.74%	*25.79%*	*74.21%*
ori#1.2.3#II-20	0.26%	2.89%	22.37%	33.16%	32.37%	8.95%	*25.53%*	*74.47%*
ori#1.2.3#II-30	0.00%	2.63%	18.42%	41.05%	30.79%	7.11%	*21.05%*	*78.95%*
ori#1.2.3#II-40	0.00%	2.63%	18.16%	39.21%	33.16%	6.84%	*20.79%*	*79.21%*

表 5-2-1 II-CNN 架構、1.2.3 ori input feature 各 filter 數準確度分布表

項目	<20%	20%<=acc<25%	25%<=acc<30%	30%<=acc<35%	35%<=acc<40%	>=40%	*<30%*	*>=30%*
sim#1.2.3#II-10	0.00%	3.95%	20.00%	34.21%	33.16%	8.68%	*23.95%*	*76.05%*
sim#1.2.3#II-20	0.00%	5.00%	18.95%	35.26%	31.32%	9.47%	*23.95%*	*76.05%*
sim#1.2.3#II-30	0.26%	2.89%	20.79%	36.05%	30.26%	9.74%	*23.95%*	*76.05%*
sim#1.2.3#II-40	0.00%	4.21%	21.58%	33.42%	33.68%	7.11%	*25.79%*	*74.21%*

表 5-2-2 II-CNN 架構、1.2.3 sim input feature 各 filter 數準確度分布表

由表 5-2-1 及表 5-2-2 的數據顯示，在 ori 的表示方式、1.2.3 特徵盤面組合的情況下，當每一層 convolutional layer 的 filter 數量增多時，準確率大於等於 30%所佔的百分比有逐漸上升的趨勢；而在 sim 的表示方式、1.2.3 特徵盤面組合的情況下，II-10、II-20、II-30 的準確度分布百分比接近，但是在 II-40 的狀況下，準確的分布開始往較低的方向下降。

由此實驗可以看到，ori 較複雜的表示方式需要使用較多的 filter 去抽取 input feature 或上一層 convolutional layer 的特徵，隨著 filter 數量的增加，準確度的分布有往高百分比移動的趨勢，逼近效果較好的 I CNN 架構的準確率分布。

相較於 ori 的表示方式，sim 在 II-10、II-20、II-30 的情況下，準確度的分布沒

有太大的變化；不過，在 II-40 的時候，準確度的分布往低百分比移動。我們認為 sim 的表示方式，較易使 II CNN 架構學習到我們想要的特徵，使用較少數量的 filter 就可以達到一定的效果；當使用過多數量的 filter，將會開始出現 overfitting(或稱為過度學習、過度訓練)的現象，也就是說，filter 數量過多將會使 CNN 失去 weight sharing 所帶來的容忍 noise 或不良 training data 的優勢，學習到錯誤的特徵、降低訓練的效果。

第六章　結論與未來工作

6.1 結論

本研究使用的方法，效果比原本只使用蒙地卡羅演算法的程式差，主要的原因出在對策類神經網路訓練出來的對棋盤盤面走步判斷的準確度不夠高，無法被用來代替規則導向對棋盤盤面做出正確的淺層展枝的動作，連帶使得下棋程式整體的棋力較弱。

第一，在實驗中觀察到的類神經網路的訓練準確度頻繁地浮動，有可能是起因於 Littlegolem 網站前 10 名玩家的棋譜差異性過大所致；換句話說，對稱的棋盤盤面、骰子點數、顏色，可能因為不同的玩家走法不同，而有所差異。以下有個較為極端的例子，如圖 6-1-1 所示：

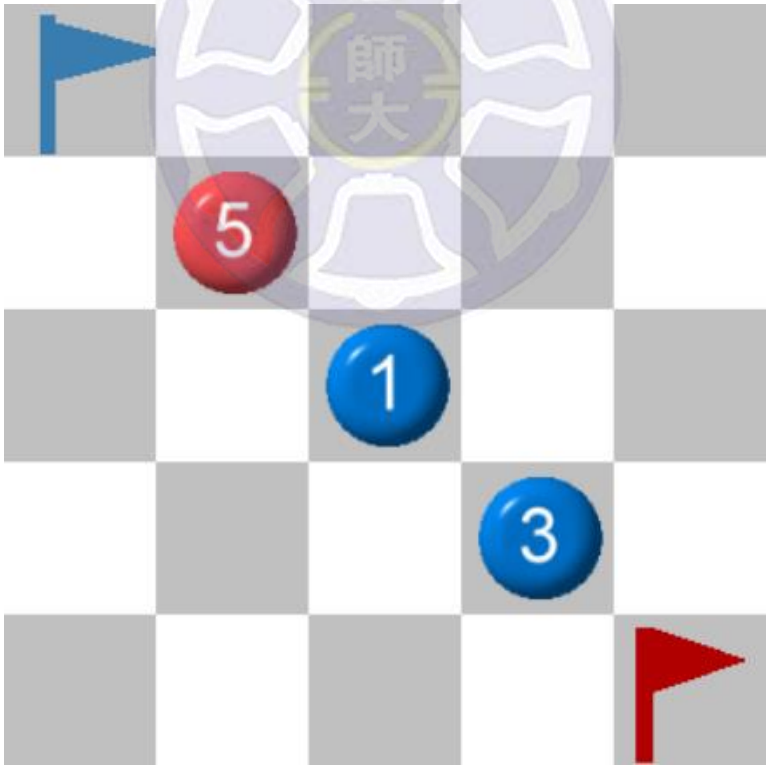

圖 6-1-1 棋盤盤面棋型對稱範例

若現在為紅方的回合，移動子為紅色 5，在這種情況下，正常的下棋程式判斷絕對

不會走右下這一個必輸的壞棋，因此下棋程式會從右或下擇其一，且不管選擇哪一者，直觀來看是等價的。我們在做 training data 及特徵盤面的擷取的時候，並沒有做這一部份的處理，將類似這種對稱的棋盤盤面棋型視為同一種，是影響類神經網路訓練準確率的重要因素之一。

第二，由於本研究的 training data 數量只有約 25 萬筆，撇除其中可能重複的數量，以愛因斯坦棋的遊戲複雜度約 10^{15} 來看，是非常不足的；再加上 training data 沒有做敵我鏡像處理，如圖 6-1-2 所示，使 training data 的數量更加不足。

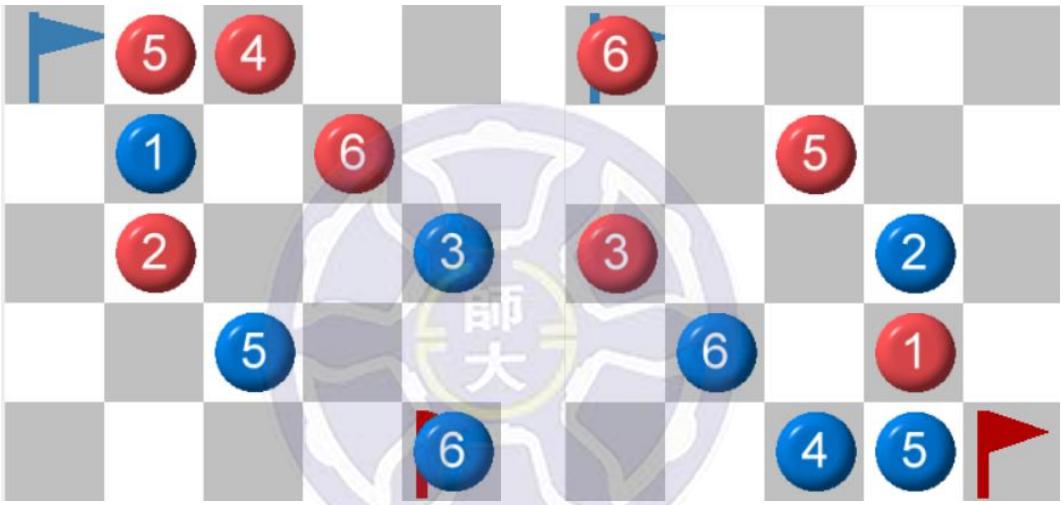

圖 6-1-2 棋盤盤面敵我鏡像

第三，特徵擷取、其表示方法和類神經網路的結構不適合，由實驗的結果可以發現：當特徵盤面只有 1 特徵盤面(紀錄棋盤狀態及當回合可動子)的時候，效果是最佳的，此現象造成的原因很多，我們推測是 sim 的 2 特徵盤面設計不佳及類神經網路無法學到我們想要的知識。

最後，較底層蒙地卡羅演算法的模擬效果不佳，以圖 4-6 的程式架構來看，此作法無法有效地減少模擬廣度，再加上淺層類神經網路展枝的精準度不佳，導致程式的整體棋力較原本弱。

6.2 TAAI 2016 比賽概況

Rank	Program name	operator	Organization
1	Meowdero	謝昌龍	臺灣師範大學
2	VS_WTN	Yunpeng Zhang	北京科技大學
3	PointPlaneLine	曹少剛	臺灣師範大學
4	ImFay5	林立秦	交通大學
5	Katze Ohren	林品儒	臺灣師範大學

表 6-2 TAAI 2016 愛因斯坦棋比賽結果

表 6-2 為 2016 年 TAAI 的比賽成績表，第三名為本研究將會用來產生 training data 的程式 PointPlaneLine，由於此程式只使用了最陽春的蒙地卡羅演算法，因此不敵第一名的蒙地卡羅樹搜索演算法及第二名的期望值搜索演算法，皆以 5 比 3 的成績作負。

首先，由於比賽採用人工操作，並非如實驗時使用愛因斯坦棋平台大量對弈的情況，比數跟第一、二名非常接近，若是在平台上大量對弈情況的話，差距將會非常明顯，因此程式的改良是必要的。

再來，比賽採用的是積分制，如何把握在跟較弱的程式對弈時拿下較多勝場數是拿下好名次的重要關鍵，第一名的程式在跟四、五名的程式對弈時皆拿下全勝的成績，其在對盤面走步判斷的精準度，遙遙領先於其他程式。因此，如何增強程式的穩定度也是重要的關鍵，尤其是比賽局數不多且勝敗的一部份被機率影響的情況，必須增進計算的精準度，把握所有的勝機，避免走出較壞的棋步。

6.3 ICGA 2017 比賽概況

	PointLinePlane	Meowdero	Emc2/7025	TBD	VSWTN	Score
PointLinePlane		1:7	2:6	4:4	2:6	9
Meowdero	7:1		4:4	1:7	4:4	16
Emc2/7025	6:2	4:4		4:4	1:7	15
TBD	4:4	7:1	4:4		7:1	22
VSWTN	6:2	4:4	7:1	1:7		18

表 6-3 2017 ICGA 比賽成績表

表 6-3 為 2017 年 ICGA 的比賽成績表，此次比賽參加的程式為新的版本，由於類神經網路的對盤面走步的判斷準確度只有三至四成、程式架構無法有效地減少模擬廣度及程式速度較慢、蒙地卡羅模擬次數不足等原因，影響程式整體的棋力表現，因此此次比賽排名第五。

6.4 未來工作

未來可繼續努力的方向如下：繼續蒐集更多的 training data，並對其做對稱性等價處理及敵我鏡像處理，增加 training data 的變化性、減少重複的機率。改善 training data 前處理的表示方法、調整類神經網路的架構，找出提升訓練準確率的方法。使用類神經網路作為 MCTS 模擬時 prior knowledge，以彌補模擬次數不足的問題，加入 UCB 作為選擇模擬分枝的依據，讓有潛力的節點能夠得到模擬的機會。

參考文獻

[1] 李占宇、李淑琴、顧磊、史玉峰、周文敏，"愛恩斯坦棋演算法設計與分析"，信息技術與信息化，第 1 期，2014。

[2] 謝昌龍、林順喜，"電腦愛因斯坦棋自動對弈平台的設計與開發"，Proceedings of TCGA 2016，pages 21–27，2016。

[3] 楊君亮、許庭嫣、林立秦，"愛因斯坦棋的電腦棋類程式設計"，Proceedings of TCGA 2017，pages 96-106，2017。

[4] 朱詠嘉、陳源灝，"愛因斯坦棋人工智慧"，Proceedings of TCGA 2017，pages 85-95，2017。

[5] R. J. Lorentz. ,"An MCTS program to Play EinStein Würfelt!", In Proceedings of the 12th International Conference on Advances in Computer Games, pages 52–59, 2011.

[6] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis, “Mastering the game of Go with deep neural network and tree search”, NATURE Vol.529, JAN 2016, pp. 484-503.

[7] 愛因斯坦棋介紹，http://www.3-hirn-verlag.de/MasterGame/regel.html。

[8] 維基百科：蒙地卡羅法，https://zh.wikipedia.org/wiki/%E8%92%99%E5%9C%B0%E5%8D%A1%E7%BE%85%E6%96%B9%E6%B3%95。

[9] TensorFlow MNIST For ML Beginners，

https://www.tensorflow.org/get_started/mnist/beginners。

[10] 維基百科：TensorFlow，https://zh.wikipedia.org/wiki/TensorFlow。

[11] TAAI 2016 官方網站比賽成績，http://www.cs.nthu.edu.tw/~taai2016/。