國立台北大學資訊工程學系專題報告

研究 AlphaZero 強化式學習流程並且應用在 Ultimate Tic Tac Toe

專題組員:駱昱均、蔡昀辰、楊浩、黃冠瑋

專題編號:PRJ-NTPUCSIE-110-009
執行期間:2021 年 7 月 至 2022 年 5 月

1. 摘要

此專題研究如何以人工智能研究公司 DeepMind 開發的程式 AlphaZero 掌握棋類遊戲。為此，我們選擇了 Ultimate Tic Tac Toe 作為中高難度遊戲作為本研究的主題。該程式基於深度學習、強化式學習和賽局樹演算法。要獲得表現良好的 AI，必須進行訓練，並且需要相對較長的時間。在有限的硬體設備下，我們訓練了一個模型來與人類玩Ultimate Tic Tac Toe 遊戲。在整個過程中，我們記錄了 AI 的表現結果，並見證了它隨著時間的推移而改進。

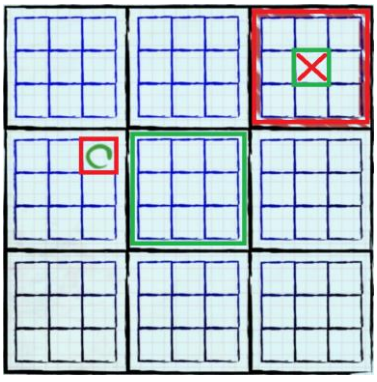
2. 簡介

提到 AlphaZero，就必須先說到 AlphaGo 以及 AlphaGo Zero。AlphaGo 作為 AlphaZero 的前身，最廣為人知的就是曾在無需讓子的情況下於 2017 年擊敗了被稱為人類最強棋士的柯潔九段。AlphaGo 的演算法是以蒙地卡羅樹搜尋法為基礎，結合了深度學習、強化式學習以及賽局樹演算法，以此形成了戰勝人類的 AI。

而 AlphaGo Zero 顧名思義，它從未使用棋譜，也就是從零開始，靠自我對弈進行訓練，這是它和 AlphaGo 最大的分別，也代表 AI 不局限於人類的知識極限，能往更深的地方發展。AlphaZero 則是 AlphaGo Zero 的改良版，它不再受限於圍棋，而是通用於各種棋類。

圈圈叉叉，又稱井字遊戲、井字棋，英文則稱做 Tic Tac Toe。而在 Ultimate Tic Tac Toe 中，和普通井字遊戲不同，在原本的 9 個大方格內有更小的 9 個方格，因此整個棋盤共有 81 個小方格，而規則大致可以分成下列三點：

1. 比賽開始時，第一位玩家在 81 個空格中任意位置落子，而他的對手則須在大棋盤中對應上名玩家所下的位置。例如，O 在任一小棋盤的右上角開始遊戲，那麼 X 需要在大棋盤中右上角的小棋盤中下一個棋子，如圖一中大紅色方框，然後 X 可以在該小棋盤的可用位置中任何一個進行遊戲，而 O 則依此類推在對應位置下棋。

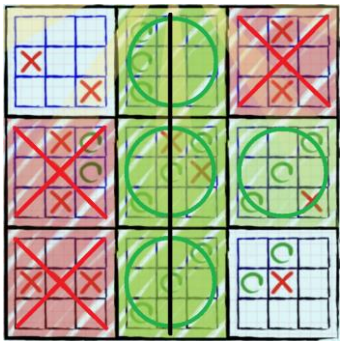


圖一(手機程式:終極井字棋)

2. 一旦有玩家贏得該小棋盤或將其完全填滿，則任何玩家不能在該小棋

盤上進行動作，轉而可以在任何其他方格上下棋。

3. 如果任何玩家按照正常井字遊戲的規則下贏得小棋盤的勝利，則將大棋盤中該位置標記為該玩家的勝利，而只須將大棋盤當作一般的井字遊戲即可分辨最後的贏家，如圖二由 O 獲勝。



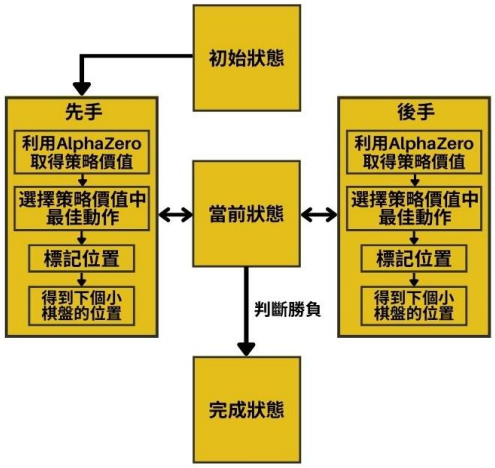
圖二(O 獲勝)

而我們目標是讓 AI 在經過訓練後，能達到與人類對戰不會輸的程度，以證明此演算法的運用可以進一步擴展。

3. 專題進行方式

此專題製作完全使用 Python，使用套件為 tensorflow, numpy, random, math, os, pickle 及 tkinter。

3.1 Ultimate Tic Tac Toe 遊戲系統設計



圖三（遊戲系統）

初始化狀態: 初始化兩個數組，表示兩個玩家的棋子，大小為 81 (9x9)。初始化三個數組，表示小棋盤是贏、輸還是平局，大小為 9。初始化變數 board 以告訴遊戲玩家當前在哪個棋盤上玩。

下棋過程:

- 取得合法棋步，在以下情況下，動作是合法的：
  1. 小棋盤還沒有勝負或平手
  2. 這一步還沒有被自己或對手下過
- 利用 AlphaZero 強化式學習：AlphaZero 中程式會輸出每個合法棋步的策略價值。根據策略價值選擇最佳的棋步。
- 標記位置: 使用已選擇的動作，將玩家的棋子數陣列標記為 1。
- 得到下個小棋盤的位置: 根據選擇的動作，小棋盤的變數必須更新為對手回合的該下的棋盤。

當前狀態：檢查當前狀態的每個小棋盤有沒有贏、輸或平手。 如果有的話，則將其標記在適當的陣列上。

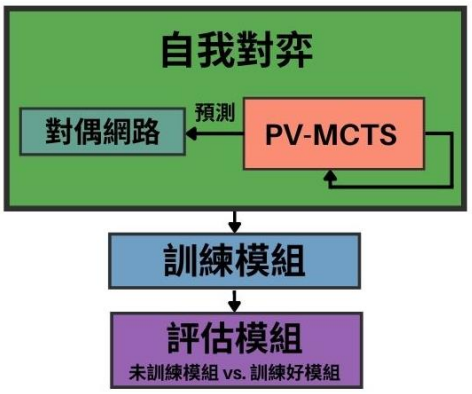
如何判斷小棋盤勝負:

- 當小棋盤 3 個棋子能夠連成一（直線、橫線、斜線）
- 當小棋盤滿格卻無法連線則平手

完成狀態：如果能夠將 3 個小棋盤連成一線(直線、橫線、斜線)，則該玩家獲勝，若是 9 個小棋盤都已分出勝負，卻沒有玩家可以連成一線，則此局平手。

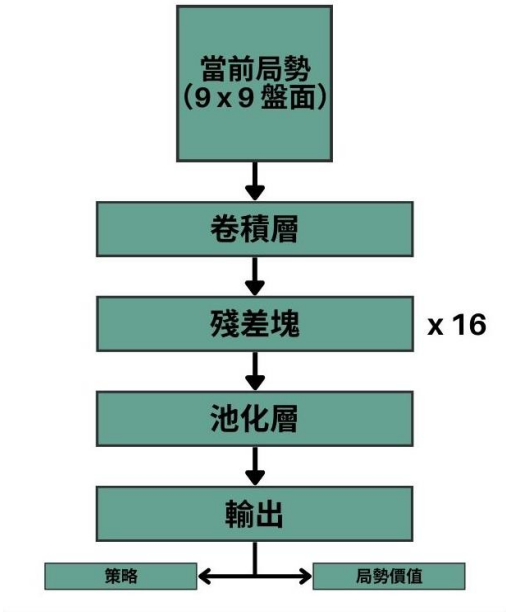
3.2 AlphaZero 機制

3.2.1 強化式學習流程



圖四（強化式學習流程）

3.2.2 對偶網路



圖五（對偶網路架構）

這是一個二合一的網路模型，也就是可以同時預測策略和局勢價值的神經網路，對偶網路　是將當前局勢(盤面)作為輸入，然後同時輸出策略(下一步)與局勢價值(預測勝負)。

實際上的架構是利用殘差網路，殘差網路中，有卷積層、殘差塊、池化層。架構是利用卷積核去掃描棋盤，最終將影像轉換為表現其特徵的特徵圖。再來會經過 Batch Normalization(批次正規化)，這裡主要會對每一層的輸出都進行正規化處理，使其成為(平均值為 0，標準差為 1)的分布，此方法會讓神經網路更容易被訓練。接下來使用激活函數 ReLU，最後得到的輸出與一開始的輸出相加再送入激活函數，避免神經網路退化。

最後建構對偶網路

1. 檢查模型是否已建構完成: 如果已經建立好最佳模型 (./model/best.h5)，便不執行。
2. 建構最佳玩家模型: 按照輸入層、卷積層、殘差塊*16、池化層、策略輸出、局勢價值輸出的順序建構模型
3. 儲存模型: 若沒有 model 資料夾則先建構一個，再將最佳玩家的模型儲存在內

3.2.3 PV MCTS

PV MCTS 或策略價值蒙地卡羅樹搜尋法使用神經網路來取每個狀態的策略價值。他的概念與一般的 MCTS 差不多一樣，有 4 個步驟，選擇，評估，擴充，更新，只是 PV MCTS 每個步驟使用不同的方法。

1. 選擇：由根節點開始，一路選擇 PUCT 最大的根節點，直到抵達葉節點為止

$$PUCT = \frac{w}{n} + Cpuct \times p \times \frac{\sqrt{t}}{1+n}$$

w = 此節點的累計價值
n = 此節點的試驗次數
$Cpuct$ = 用於調整「勝率」與「棋步預測機率 X 偏值」之平衡常數
p = 棋步的機率分布
t = 累計的試驗次數

圖六（PUCT 公式）

2. 評估: 利用對偶網路的輸出取得策略及價值。首先按照對偶網路輸入的 shape（9，9，2），設置單筆資料的 shape。做法是在單筆資料的 shape 前面加入

資料的筆數(資料筆數，9，9，2)。設置好 shape 之後，我們使用 model.predict()取得策略及局勢價值。策略會用來計算 PUCT 中棋步的機率分布，局勢價值則用來更新 PUCT 中的累計價值。

3. 擴充 : 獲得策略和價值後，我們根據狀態和策略插入對應的節點來展開子節點。 我們在每一次實驗中會擴充子節點。
4. 更新：更新該節點的策略及價值。

在獲得所有選定的子節點的試驗次數後，我們將這些節點轉換為分數列表 (0 和 1)。 這些分數代表要選擇哪些合法棋步作為下一步。 但是，由於分數是 0 和 1，我們利用波茲曼分佈增加變化也就是將概率分佈到每個合法棋步中。 既然列表已經變成了每一個合法棋步的概率，我們選擇概率最大的合法棋步作為下一個動作。

為了使模擬更快，我們使用了動態模擬(Dynamic simulation)的方法。算法很簡單，我們設置模擬檢查點來檢查合法棋步的概率，如果最大概率高於某個閾值，我們結束模擬並選擇該合法棋步作為下一步。 因此，我們設置了 3 個參數，即最大模擬次數、檢查點和分數閾值。

3.2.4 自我對弈

Self Play 是 AlphaZero 的重點，就是 AI 與自己對戰多次。 與其他 AI 方法不同，AlphaZero 使用自己通過自我對弈創建的資料集。 目標是玩遊戲直到自己學習到不同變化的棋路。 自我對弈中必須獲取的資訊有 3 種，第一種是玩家和對手的棋子，第二種是 PV MCTS 生成的策略，第三種是每個完成狀態的值。

【【我方棋子配置，對方棋子配置】，策略，局勢價值】

一個遊戲狀態有 3 種局勢價值:

贏 : 1，輸：-1，平：0。

在遊戲的每個狀態中，我們將這 3 種資訊放入 history 中，當遊戲結束時，我們將所有 history 保存在一個文件中。

3.2.5 訓練模組

1.載入訓練資料:首先建立一個函式 load_data()用來載入自我對弈模組所儲存的訓練 資料。接下來利用 zip()將上述資料中的【【【我方棋子配置，對方棋子配置】，策略，局勢價值】轉換為 3 個獨立的串列像是

(1)【【我方棋子配置，對方棋子配置】,【我方棋子配置，對方棋子配置】,...】

(2)【策略，策略，...】

(3)【局勢價值，局勢價值，...】

2.重塑訓練資料的 shape:因為要一次輸入多筆的訓練資料，所以多一軸(訓練資料筆數，9，9，2)

3.載入最佳玩家模型

4.編譯模型: 需指定損失函數、優化器

5.調整學習率:初始設置為 0.001，經過 50 步設為 0.0005，80 步後設為 0.00025

6.印出訓練次數

7.進行訓練

8.儲存最新玩家的模型

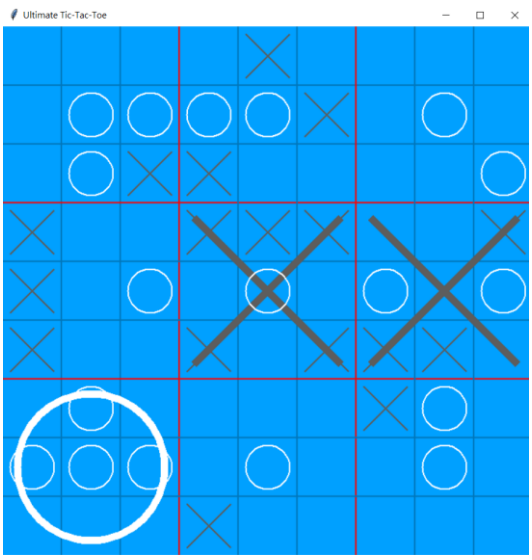
3.2.6 評估模組

模型訓練好後，進行評估，讓我們知道 AI 的學習進度。 首先是通過與未訓練模型對戰 10 次來評估新訓

練的模型。 如果訓練的模型在 10 場比賽中得分超過 50%，那麼我們將最好的模型更新為新訓練的模型。

3.3 人類與 AI 對戰

將之前訓練好的 best 模型放入 UI 中，用 Tkinter 實現人類與 AI 對戰。利用事件處理，將滑鼠左鍵作為的事件常數、下棋作為事件處理函式。我們設定人類一定是先手，當輪到人類下子時，首先須先判斷遊戲是否結束，再來須要判斷是否為人類的回合，將滑鼠點擊的位置(X,Y)轉換為格子編號，再來須判斷下的位置是否為合法棋步，點選那個位置之後，就需要更新畫面並且取得新的盤面，再來就是換 AI 下棋，同樣要取得 AI 下的編號轉換成 X,Y 座標，這樣才能更新畫布。畫圈及畫叉的做法則是，先獲得 X,Y 座標後，畫圈是要利用左上角到右下角的座標，並且利用 Canvas 裡面的 create_oval 函式來實現畫圈。畫叉式利用 2 條直線一條是從左上角到右下角的座標，另一條則是從右上到左下的座標，並且利用 Canvas 裡面的 create_line 函式來實現畫叉。最後更新畫布的方法則是，先將舊有畫布刪除，重新畫上遊戲的外框以及判斷所有以下子的位置，將圈跟叉都重新畫上。



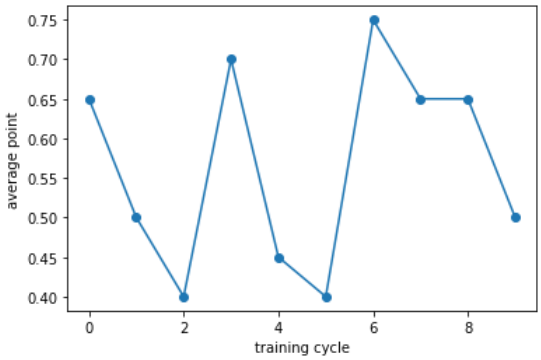
圖七（遊戲畫面）

4. 主要成果與評估

4.1 訓練配置

- 模擬次數 = 27 - 81
- 動態模擬閾值 = >0.8
- 自我對弈游戲局數 = 1000
- 訓練次數 = 100

4.2 訓練結果

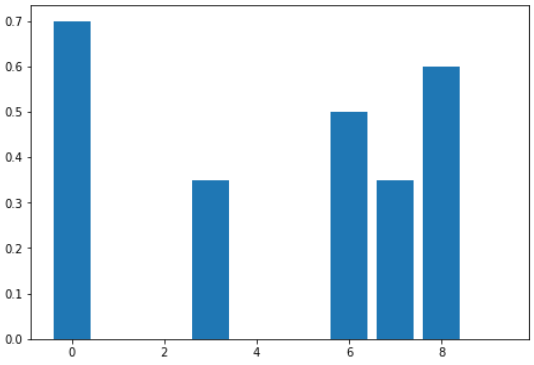


圖八（未訓練模組與訓練好模組對戰結果）

Average point 是未訓練的模型與自我對弈 1000 次後的模型對戰，若得分超過 0.5 則更新 Bestmodel，所以並不是每次對亦完都會更新。

4.3 對弈結果

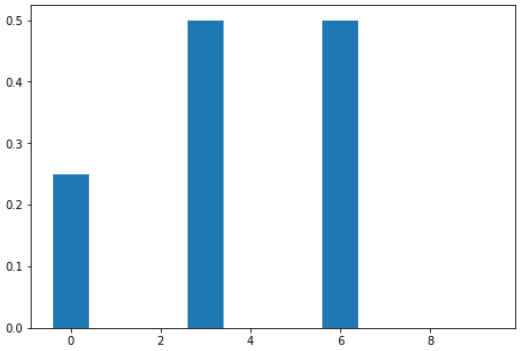
4.3.1 與隨機下法對戰結果



圖九（BestModel vs. Random）

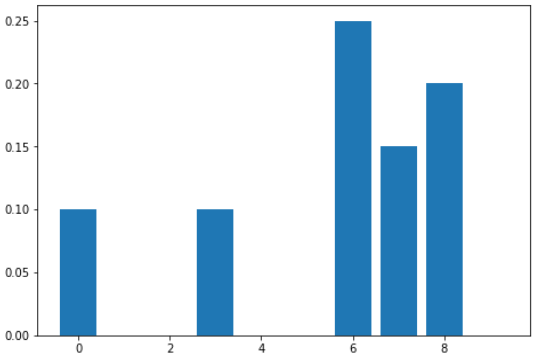
4.3.2 與其他演算法對戰結果

Alphabeta 演算法介紹: Alphabeta 是利用樹狀結構將賽局分為敵、我回合，並且為了評估每步棋的好壞，我們會加入一個稱為局勢價值的指標，將敵我兩方的優劣做量化。這個演算法會從當前局勢往後推估幾個回合，每一個節點代表那個棋步的局勢價值，且每個節點都有 α 與 β 兩個值，用來判斷哪些節點的後續子節點可以減掉不處理， α 代表該節點的局勢價值至少為 α ， β 代表該節點的局勢價值至多為 β ，依序由下面往上推，敵方下子的盤面取局勢價值的最小值(對敵方最不利)，而我方下子時的盤面則取局勢價值的最大值(對我方最有利)，在過程中搜索子節點時，都會對此節點的 α 與 β 進行修正，當修正後出現了 α 值 $>$ β 值的情況，就代表後續的子節點可以剪掉不處理了，總而言之，Alphabeta 就是判斷即使不計算也不會影響到整體結果的部分來進行剪枝，進而加快程式執行速度。而依序往上推就可得知我們當前的局勢要往哪個位置落子。



圖十（BestModel vs. AlphaBeta）

MCTS 演算法介紹: MCTS 是先根據隨機模擬來計算節點的局勢價值，做法是先將當前的合法棋步先模擬過一輪，並找出模擬結果最好的當作下一個棋步。而 MCTS 演算法分成四個部分，分別是選擇(Selection)、評估(Evaluation)、擴充(Expansion)、更新(update)，選擇(Selection)的部分是從根節點開始，連續向下選擇子節點至葉節點為止。擴充(Expansion)是當選擇完子節點時執行 playout，根據每一次 playout 的結果來更新局勢價值。擴充(Expansion)則是因為任何子節點僅試驗一次還不足以判斷此節點是否值得探索下去，而當某節點的試驗次數超過自行設定的次數，則將該節點擁有的合法棋步增加為子節點。更新(update)是在 playout 完，需重複將 playout 算出該節點的局勢價值加入根節點，而用意是可以從根節點清楚知道當前各局勢的總試驗次數。不斷的重複這四個步驟，直到達到自己設定的次數。



圖十一（BestModel vs. MCTS）

4.3.3 分析結果

模型只要經過訓練並且更新，對戰 random 下法幾乎都可以達到超過50%的勝率，但是因為對戰場數不多，有時候不能到達一定的勝率。在這個遊戲中，AlphaBeta 及 MCTS 還是比 BestModel 強，使得模型還不能夠達到50%的勝率。經過圖十及圖十一的比較，MCTS 可能優於 AlphaBeta。

5. 結語與展望

這次的專題我們成功的應用 AlphaZero 演算法。與人類的對戰過程中，可以發現它有在進步，從一開始未經訓練的模型，到最後，可以發現它與我們的思考越來越相近。在專題的最後，我們見證了一個 AI 的成長，如何學到一個遊戲的策略。AlphaZero 是一個偉大的人工智慧，但是它的缺點是需要很強的設備去訓練。未來若是有機會，我們可能會開發更高效的強化式學習演算法，讓它訓練效果提升。

6. 銘謝

感謝指導教授一步步的帶領，讓我們從一次次的從報告中找到自己專題的方向，並在我們都是第一次做機器學習的情況下，給予我們許多關於模型上的建議，也在我們出差錯的時候適時的指正，以及提供實驗室的設備讓我們能順利的訓練模型，讓我們順利的完成此次專題。

7. 參考文獻

[1]布留川英一作；王心薇譯(2021)。強化式學習：打造最強 AlphaZero 通用演算法

[2]Li-Cheng Lan; Ti-Rong Wu; I-ChenWu; Cho-Jui Hsieh 2021. Learning to Stop: Dynamic Simulation Monte Carlo Tree Search arXiv preprint arXiv:2012.07910.

[3] David Silver；Thomas Hubert；Julian Schrittwieser；Ioannis Antonoglou；Matthew Lai；Arthur Guez；Marc Lanctot；Laurent Sifre；Dharshan Kumaran；Thore Graepel；Timothy Lillicrap；Karen Simonyan；Demis Hassabis 2018. A general reinforcement learning algorithm that masters chess, shogi and Go through self-play