第十九屆旺宏科學獎

創意說明書

參賽編號：

作品名稱：Algorithms to play UNO

姓名：吳邦寧

關鍵字：基因算法、強化學習、機器學習

摘要

電腦攻克了許多領域，包括西洋棋、圍棋以及五子棋，卻鮮少有人研究UNO—全世界最暢銷的桌遊。

因此，本研究設計了五種不同的演算法，Dummy algorithm、Naïve offensive algorithm、Naïve defensive algorithm、Genetic algorithm 以及 Deep-Q-Learning algorithm，並讓這五種演算法玩 UNO，相互對抗，看誰是最終贏家。

經過實驗得知，UNO 是一個「先手優勢」的遊戲，不論是哪種傳統演算法，先手的勝率都比較高，為了保證實驗結果正確性，其餘關於勝率的實驗都是做先後手各三十場，確保數據不受先後手影響。

每種演算法都有自己的特性，Dummy algorithm 是一個名副其實的「假人」，不論怎麼打都必輸，正如預料；Naïve offensive algorithm 如同只見近利之人，有好牌就出，被 Naïve defensive algorithm，也就是「永遠最後才出好牌」的人，給打敗了。

Genetic algorithm 好比細菌，使用基因記憶所學所能；Deep-Q-Learning 好比人類，使用大腦記憶所學所能，若是細菌對上人類，也就是 Genetic algorithm 對上 Deep-Q-Learning，究竟會如何呢？答案是人類獲勝，就像人類在瘟疫中戰勝細菌一樣，Deep-Q-Learning 也戰勝了 Genetic algorithm，不過偶爾也有 Genetic algorithm 戰勝 Deep-Q-Learning 的時候，就像是瘟疫爆發，細菌也有戰勝人類的時候。

本研究設計了諸多演算法，經過實驗後得知，這些參賽者中的黑馬是 Deep-Q-Learning，不僅勝率穩定，還能夠藉由 Behavior cloning 增加勝率，是目前最強的演算法。

繼 Deep Blue、Alpha Go 之後，電腦又將矛頭指向全世界最暢銷的桌遊—UNO，蓄勢待發，準備攻克這個領域！

壹、研究動機

電腦攻克了許多領域，包括西洋棋、圍棋以及五子棋，卻鮮少有人研究UNO—全世界最暢銷的桌遊。

經過網路文獻回顧，只有一篇「Notes on machine learning - Playing UNO」提出了相關的研究，該文作者採用 Genetic algorithm 進行實驗，然而成效不彰，且作者無意繼續研究。

細菌好比 Genetic algorithm，人類好比 Deep-Q-Learning；細菌只能藉由基因記憶所學所能，但是人類能夠藉由學習記憶所學所能，若是使用 Deep-Q-Learning 進行實驗，必能帶來斬獲。

Deep-Q-Learning 常常應用在 Unsupervised learning & Reinforcement learning 領域，Alpha Go 也是基於 Deep-Q-Learning 的產物，筆者認為若是 Deep-Q-Learning 與 Genetic algorithm 較勁，Deep-Q-Learning 將會勝出，好比人類在一次又一次的瘟疫中戰勝細菌。

貳、研究目的

一、設計各種不同的演算法來玩 UNO

二、探討先手與後手的利與弊

三、探討各演算法之間的勝率

四、探討 Deep-Q-Network 與 Genetic-Network 的利與弊

參、文獻回顧

一、Notes on machine learning - Playing UNO

https://john-hearn.info/articles/notes-on-machine-learning-playing-uno

The article used four different algorithms, dummy algo, defensive algo, offensive algo and genetic algo to conduct the research. The author also reveals that genetic optimization is not effective on deep neural networks due to the randomness of genetic fluctuation.

二、RL - Cards

https://arxiv.org/pdf/1910.04376.pdf

It's an open source library recently published in Feb 2020 for conducting reinforcement learning researches of card games. Despite its usefulness, the library can't meet my needs. Thus, I decided to use it as a blueprint to build my own.

三、Deep-Q-Learning

https://www.mlq.ai/deep-reinforcement-learning-q-learning/

The article is a tutorial of Deep-Q-Learning illustrated by vivid graphs and simple math which is much more unsophisticated than the original paper. The following discussion would assume readers understand the concepts of neural networks, Deep-Q-Learning and Markov decision process which is discussed in the article.

肆、研究器材

	器材	用途
研究器材	Tensorflow	Lessen the coding complexity for developing.
	Tkinter	Visualize the gaming process.
	NVIDIA 1660 Ti	Hardware for increase training performance.
	CUDA	Utilize GPU to increase training performance.
	Tensorboard	Plot the win rate.
	Python	Language used for conducting experiments.
	Github	Online web service for storaging codes and files.

伍、研究方法

一、遊戲規則

一張圖勝過千言萬語，如下簡易流程圖所示。

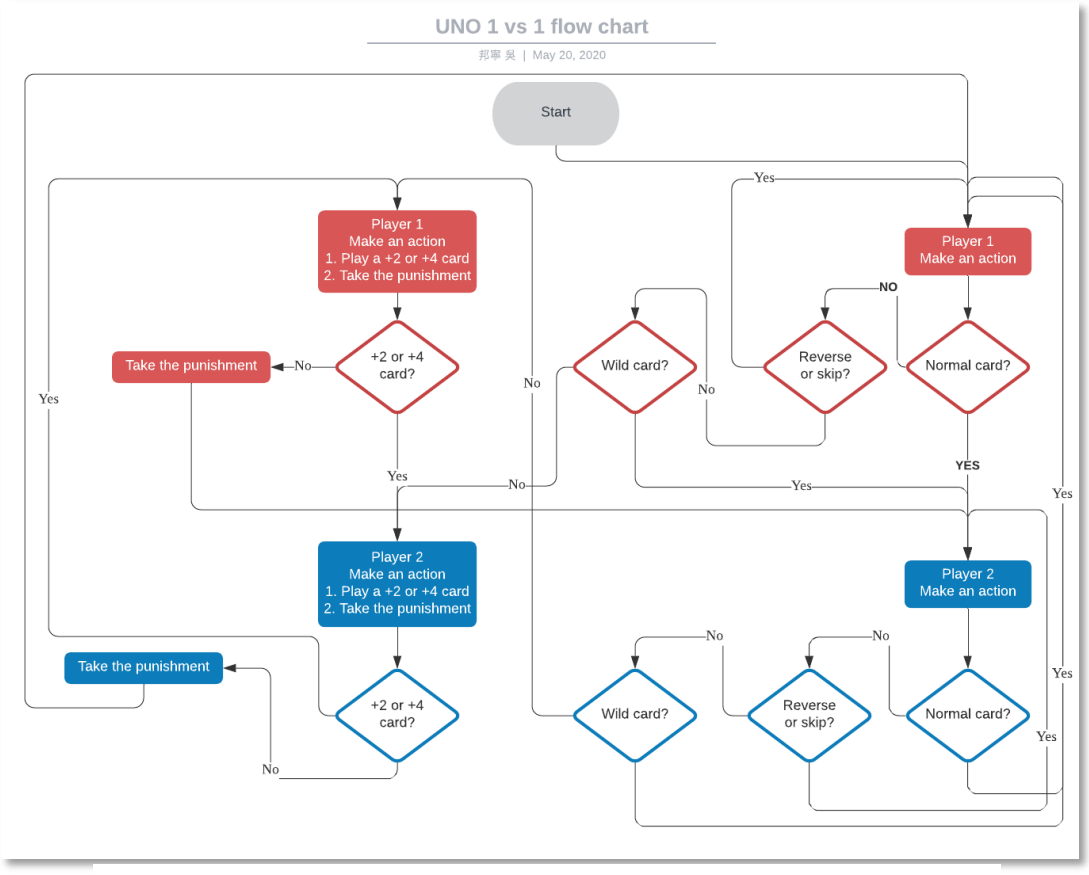

圖 1 ▲ 遊戲簡易流程圖

這並不是一張完整的流程圖，這張流程圖只是很簡明扼要的表達遊戲流程，像是「先出完牌就贏了」這種眾所皆知的規則沒有被記錄上去，值得注意的是，規則中不包含「Wild draw four challenge」。

二、實驗流程

(一)、傳統演算法對抗傳統演算法

1. 讓兩個演算法進行遊戲先後手至少各三十場
2. 統計結果，並回到 (1.)

(二)、傳統演算法對抗神經網路

1. 初始化若干個神經網路，若是 DQN 則執行 Behavior cloning
2. 讓神經網路與傳統演算法進行遊戲若干場
3. 使用勝場最多的神經網路與傳統演算法對抗先後手各三十場
4. 統計結果，並回到 (2.)

(三)、Deep-Q-Learning 對抗 Genetic algorithm

1. 初始化若干個 Deep-Q-Network，並執行 Behavior cloning
2. 讓 DQN 與假想敵(Defensive algorithm as default)進行遊戲若干場
3. 讓 Petri dish 上的 Cells 與勝場最多的 DQN 進行遊戲若干場
4. 使用勝場最多的 Cell 與勝場最多 DQN 對抗先後手各三十場
5. 統計結果，並回到 (3.)

(四)、勝率圖表判讀方法

下圖中，每一個點代表先後手各三十場，橫軸代表$\lfloor\frac{場數}{60}\rfloor$，縱軸代表勝率。

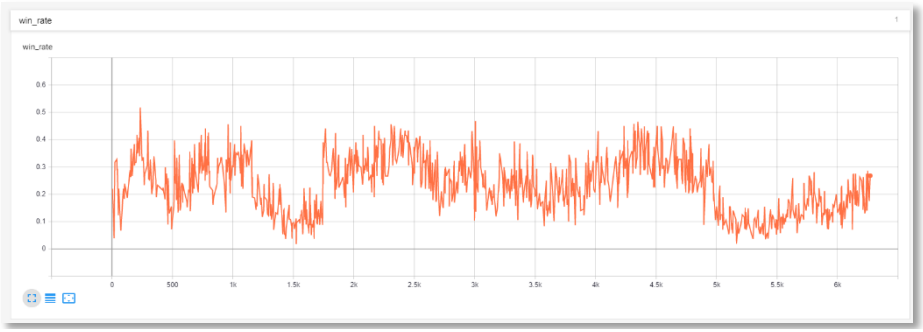

圖 2 ▲ 勝率圖範例

三、Dummy algorithm

一言以蔽之，這個演算法等同「找一張能丟的卡或抽牌」。

Algorithm:Dummy algorithm

```
function GET ACTION(S as state)
    Actions ← S.Playable ∪ Take card from deck
    return Random.GetOne(Actions)
```

四、Naïve Offensive/Defensive Algorithm

Offensive algorithm 等同「先丟最強的功能牌，再丟普通牌，沒牌再抽卡」，猶如忿狷之人，有好牌就出，不顧後果。

Algorithm:Naive Offensive algorithm

```
function GET ACTION(S as state)
    Functional Cards ← S.Playable ∩ Card.Functional
    Normal Cards ← S.Playable ∩ (Card \ Card.Functional)
    if Functional Cards is not ∅ then
        return Random.GetOne(Functional Cards)
    else if Normal Cards is not ∅ then
        return Random.GetOne(Normal Cards)
    else
        return Take card from deck
```

Defensive algorithm 等同「先丟普通牌，再丟功能牌，沒牌再抽卡」，猶如杞人憂天，永遠把好牌留在手上，以備不時之需。

Algorithm:Naive Defensive algorithm

```
function GET ACTION(S as state)
    Functional Cards ← S.Playable ∩ Card.Functional
    Normal Cards ← S.Playable ∩ (Card \ Card.Functional)
    if Normal Cards is not ∅ then
        return Random.GetOne(Normal Cards)
    else if Functional Cards is not ∅ then
        return Random.GetOne(Functional Cards)
    else
        return Take card from deck
```

五、Genetic algorithm

演算法如同在一個 Petri dish 上模擬許多個 Cells，細胞將與對手(Opponent)決鬥，根據決鬥結果計算細胞的最終積分(Rank)，再進行基因擴散(Genetic diffusion)，藉此交換基因。

Algorithm:Genetic algorithm

```
function GET ACTION(S as state)
    for 1 ≤ i ≤ 108 do                    ▷ There are 108 cards in a UNO deck
        Mask_i ← [Card_i ∈ S.Playable]     ▷ [ ] represents Iverson bracket
    Mask_0 ← 1                  ▷ Player may always take a card from the deck
    Result ← Network.Run(S)
    Result ← Sigmoid(Result) + 1
    return Argmax(Mask ⊙ Result)        ▷ ⊙ represents Hadamard product
function GENETIC DIFFUSION
    Alpha ← The cell achieved the highest rank
    for Cell in Petri \ Alpha do
        Cell.Weights ← (Cell.Weights + Alpha.Weights) / 2
function MUTATE
    for Cell in Petri do
        Cell.Weights ← Cell.Weights + Gaussian random()
function EVOLUTION
    Petri ← Cells initialized with zero weights
    for Cell in Petri do
        for 1 To N do                          ▷ N is an arbitrary number
            Result ← Cell v.s. Opponent
            if Result is Win then
                Cell.Rank ← Cell.Rank + 1
    Genetic Diffusion()
    Mutate()
```

不適應環境的基因會在基因擴散的過程中漸漸消失，適應環境的基因會在基因擴散的過程中壯大，如同達爾文的「適者生存」說，演化過程會淘汰掉不適應的個體，留下「贏家」，也就是適應環境的個體。

六、Deep-Q-Learning Algorithm

嚴格來說，玩 UNO 的過程不是 Markov decision process，任何情況下，玩家都不能任意出牌。因此，我對演算法進行了一些調整，如下偽代碼所示。

Algorithm:Deep Q Learning algorithm

```
Memory ← Initialize memory container with zeros
step ← 0
Network ← Initialize weights with zeros
γ ← Discount factor
ϵ ← ϵ-greedy algorithm factor
function GET RESULT(S as state)
    for 1 ≤ i ≤ 108 do                ▷ There are 108 cards in a UNO deck
        Mask_i ← [Card_i ∈ S.Playable]        ▷ [ ] represents Iverson bracket
    Mask_0 ← 1                        ▷ Player may always take a card from the deck
    Result ← Sigmoid(Network.Run(S)) + 1
    return (Mask ⊙ Result) - 1        ▷ ⊙ represents Hadamard product
function GET ACTION(S as state)
    if Random.Uniform(0, 1) > ϵ then
        return Dummy Algorithm.Get Action(S)
    else
        return Argmax(Get Result(S))
function OBSERVE(S as state, A as action, R as reward, S′ as state)
    Memory[step mod N] ← S, A, R, S′        ▷ N is an arbitrary number
    Values ← Network.Run(S)[A] + R + γ · max_A′ Get Result(S′)[A′]
    Network.Run(S)[A] ← Perform gradient descent on Values
    step ← step + 1
```

Behavior cloning 是一個常見的處理手法，如同徒弟模仿師傅的行為，能夠有效增加強化學習的成效。

Algorithm:Deep Q Learning algorithm

```
function BEHAVIOR CLONING(M as mentor, E as enviroment)
    S ← E.Get State()
    A ← M.Get Action()
    R ← E.Do Action(A)
    S′ ← E.Get State()
    Observe(S, A, R, S′)
```

陸、現階段研究成果

一、先手制霸

不論是使用哪一種演算法，先手的勝率都較高，如下表所示。

演算法	先手勝率
Dummy algorithm	53%
Naïve offensive algorithm	52%
Naïve defensive algorithm	56%

二、Genetic Algorithm 的特色 – 劇烈起伏

可以見到 Genetic algorithm 的勝率時高時低，如下圖所示。

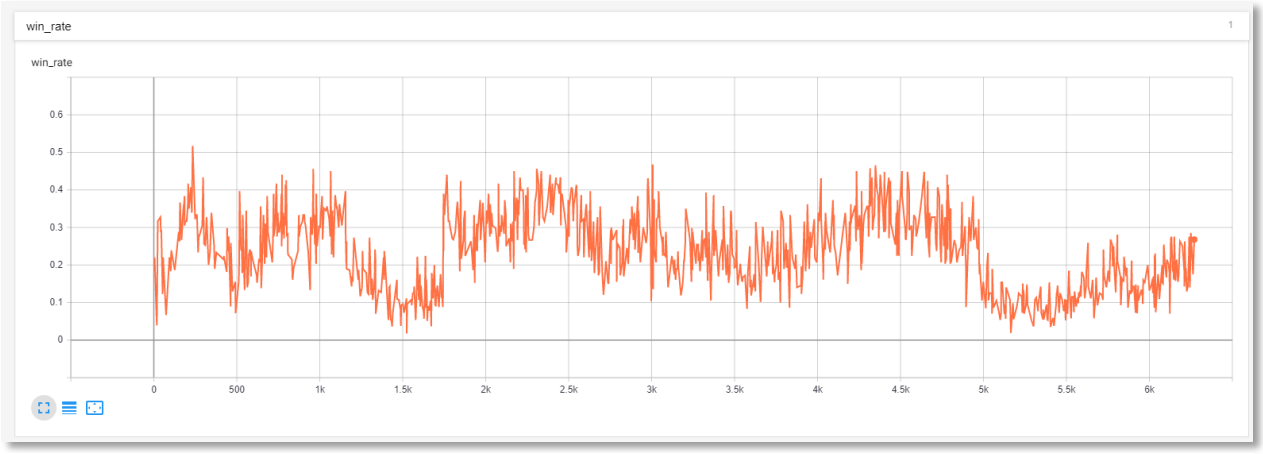

圖 3 ▲ Genetic algorithm 對抗 Defensive algorithm 的勝率圖

好的基因，適應力強的基因，也就是勝率高的基因，會隨著基因擴散(Genetic Diffusion)快速的散播到培養皿(Petri)的每個細胞內，使得勝率增加。在突變(Mutation)的影響下，好的基因也有可能變成壞的，就像惡性腫瘤一樣，使得勝率下降。

二、Deep-Q-Learning 的特色 – 恍然大悟或瞬間失憶

可以見到，Deep-Q-Learning 具有「恍然大悟」的潛力，神經網路彷彿突然觀察到某種重要線索，勝率瞬間飆升，如下圖所示。

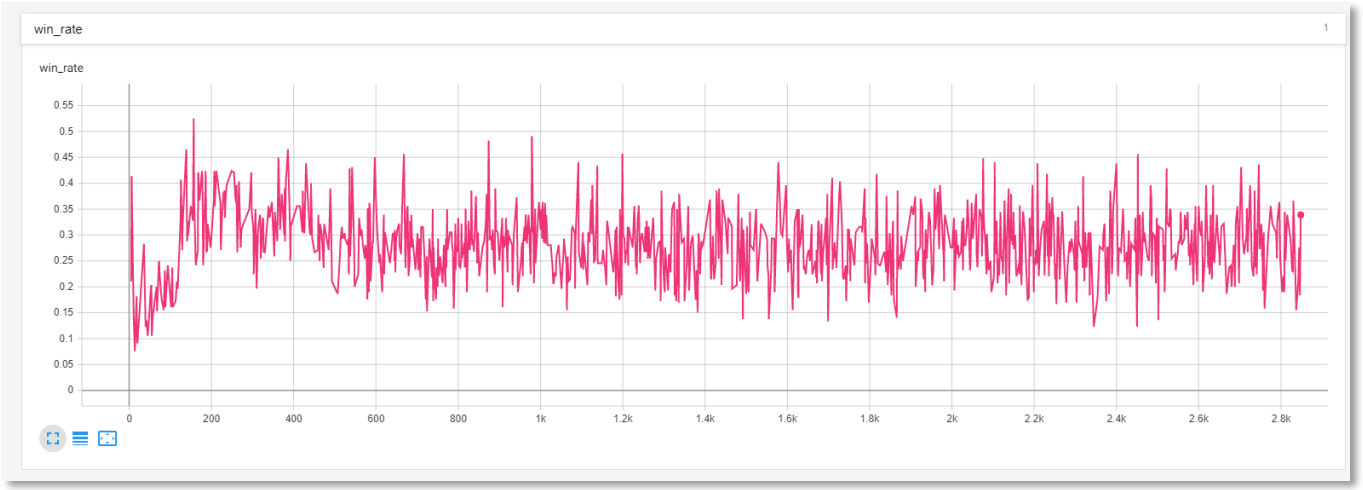

圖 4 ▲ 3-Hidden Deep-Q-Learning 搭配 Behavior cloning(Offensive algorithm) 對抗 Offensive algorithm 的勝率圖

相對的，Deep-Q-Learning 也具有「瞬間失憶」的潛力，神經網路彷彿突然失憶，把畢生所學給全忘了，勝率瞬間跌落谷底，如下圖所示。

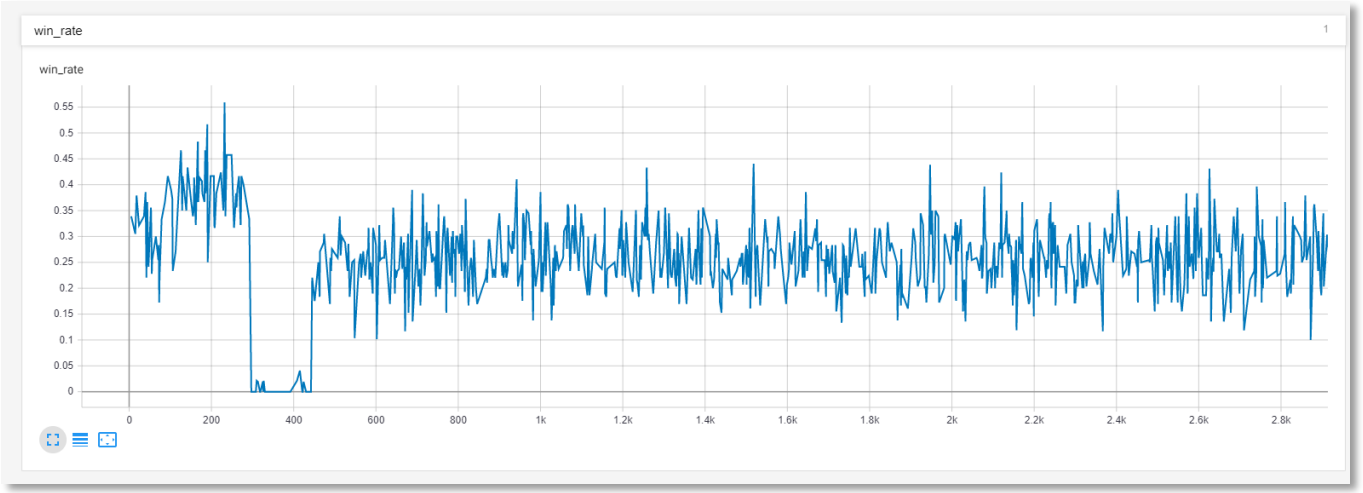

圖 5 ▲ 4-Hidden Deep-Q-Learning 搭配 Behavior cloning(Defensive algorithm) 對抗 Defensive algorithm 的勝率圖

由此可知，Deep-Q-Learning 有可能「恍然大悟」，也有可能「瞬間失憶」，不過與 Genetic-Algorithm 相較，Deep-Q-Learning 是較為穩定的演算法。

四、隱藏層深度

基因演算法對隱藏層深度不太敏感，如下圖所示。

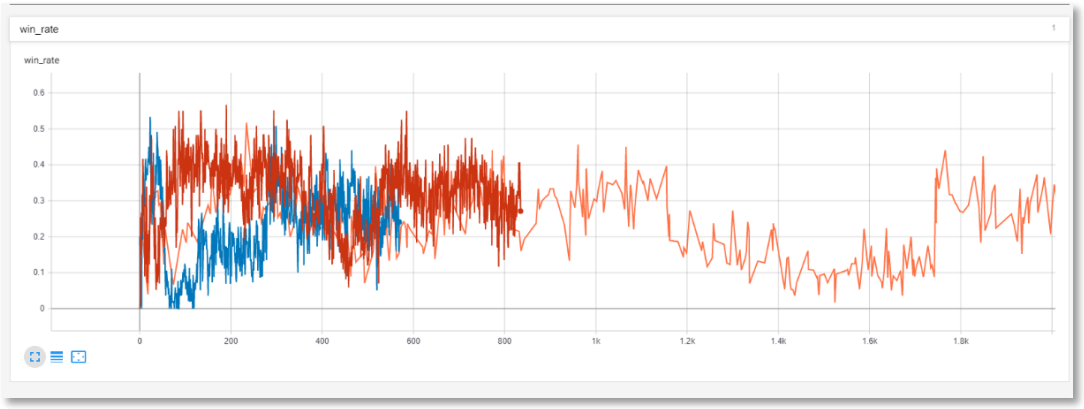

圖 6 ▲ Genetic algorithm 對抗 Defensive algorithm 的勝率圖

顏色	意義
藍	Genetic algorithm with 1-hidden layer
橘	Genetic algorithm with 2-hidden layer
紅	Genetic algorithm with 3-hidden layer

Deep-Q-Learning 對隱藏層較為敏感，隱藏層深度越深，反而會造成反效果，如下圖所示。

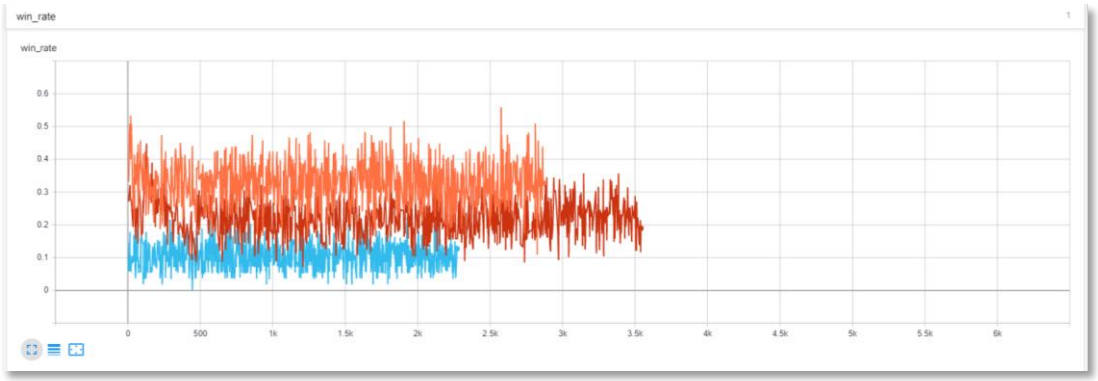

圖 7 ▲ Deep-Q-Learning 對抗 Offensive algorithm 的勝率圖

顏色	意義
橘	Deep-Q-Learning with 1-hidden layer
紅	Deep-Q-Learning with 2-hidden layer
藍	Deep-Q-Learning with 3-hidden layer

深層的神經網路如塊璞玉，未經雕琢不得其瑰。因為環境提供的樣本品質不夠好，使得神經網路無法好好的被「雕琢」，若是有一個師傅能夠提供高品質的樣本，去「雕琢」這塊璞玉，便能得到內部的瑰麗，便能得到更好的勝率。

Behavior cloning 猶如師徒相授，師傅做什麼，徒弟便跟著做；過於淺層的神經網路如朽木，再怎麼教也教不會，神經網路擬和能力過差；過於深層的神經網路如大材小用，耗費大量運算資源，卻無優異成效。

筆者認為四層的神經網路搭配 Behavior cloning，不僅能有效的增加勝率，也能妥善使用運算資源，最為恰當，因此，多數實驗都以此為標準。

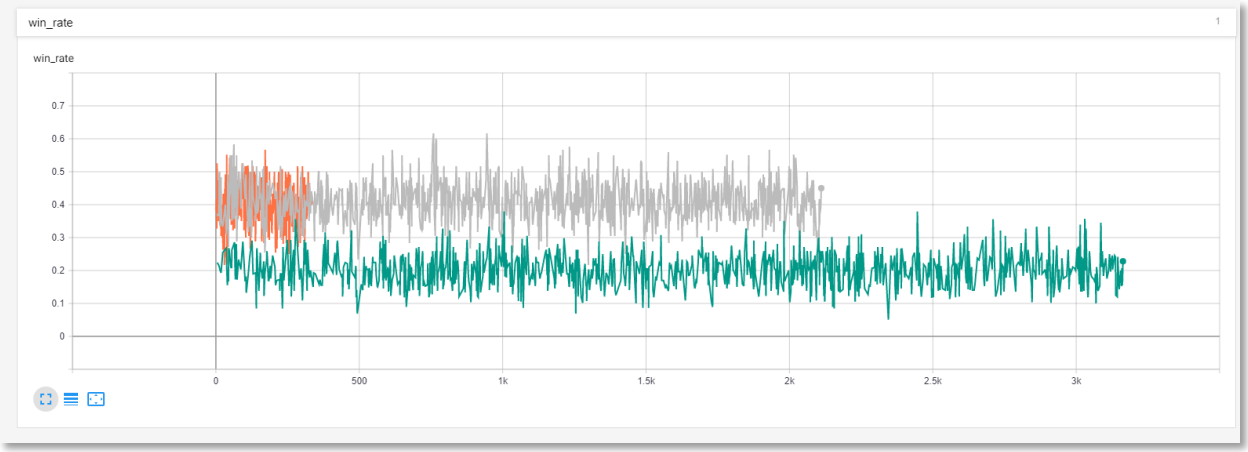

圖 8 ▲ Deep-Q-Learning 搭配 Behavior cloning 對抗 Defensive algorithm 的勝率圖

顏色	意義
橘	Deep-Q-Learning with 5-hidden layer and offensive algo as mentor
灰	Deep-Q-Learning with 4-hidden layer and offensive algo as mentor
綠	Deep-Q-Learning with 3-hidden layer and offensive algo as mentor

五、Genetic-algorithm 對決 Deep-Q-Learning

Genetic algorithm 好比細菌，Deep-Q-Learning 好比人腦，兩者都是基於神經網路的演算法，而這場神經網路大戰，結果將會如何呢？下圖展示了 Genetic algorithm 的勝率。

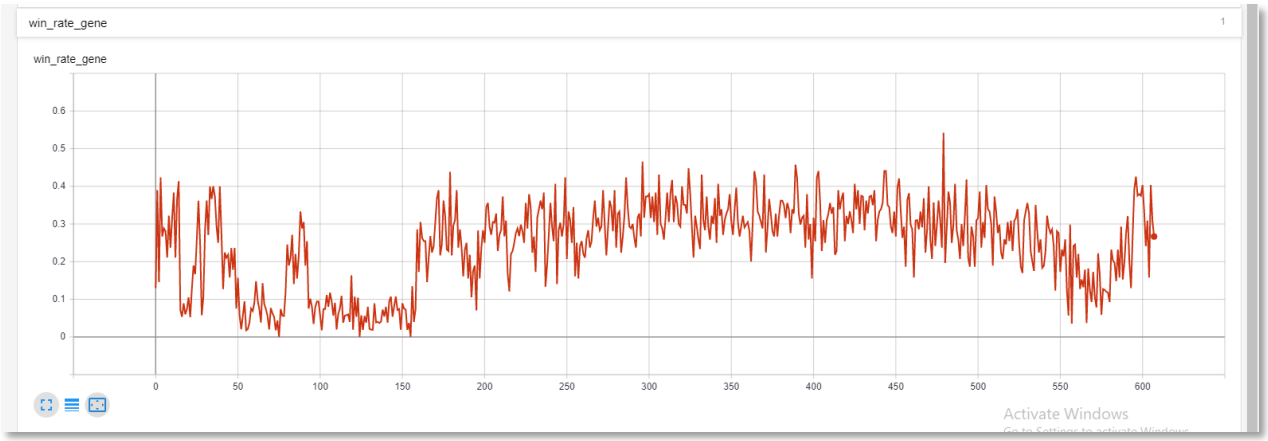

圖 9 ▲ 4-Hidden Deep-Q-Learning 搭配 Behavior cloning(Defensive algorithm) 對抗 Genetic algorithm 的勝率圖

可以見到，多數時候 Deep-Q-Learning 佔上風，藉由 Genetic fluctuation，Genetic algorithm 有時能夠贏過 Deep-Q-Learning，不過多數時候還是敗給 Deep-Q-Learning。

Gene algorithm 勝率最低點	Gene algorithm 勝率最高點
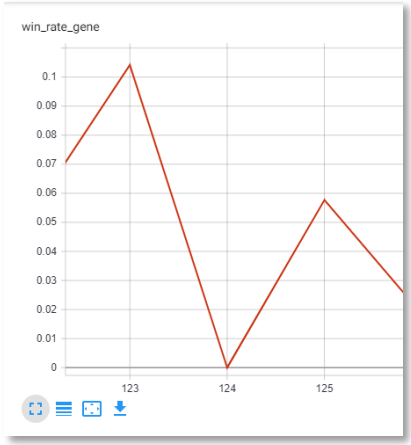	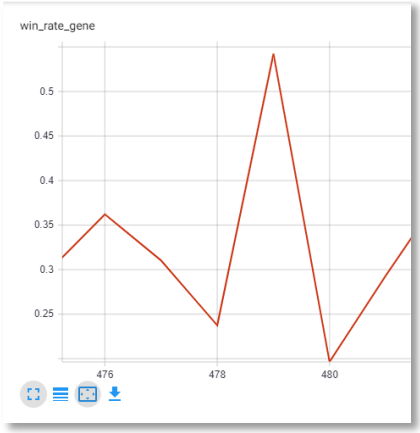
圖 10 ▲	*圖 11* ▲

在 Deep-Q-Learning 的全盛時期，勝率將近 100%，可謂戰無不勝；在 Genetic algorithm 的全盛時期，勝率高達 54%，可謂略勝 Deep-Q-Learning 一籌。

六、各演算法之間的勝率

下表展示了各演算法之間的勝率，並列出神經網路的重要規格、參數。

(一)、Genetic algorithm 採用一層 Hidden layer。

(二)、DQN 採用四層 Hidden layer，搭配 Defensive algorithm 作為師傅。

Subject / Object		Naïve Offensive	Naïve Defensive	Genetic Algorithm	DQN
Dummy	最高	93.8%	93.1%	58%	96%
Naïve Offensive	最高		54.7%	56%	53%
Naïve Defensive	最高			53%	57%
Genetic algorithm	最高				99%
	最低				46%

可以觀察到幾個重點：

(一)、Dummy algorithm 穩輸

(二)、Defensive algorithm 比 Offensive algorithm 強

(三)、Deep-Q-Learning 多數時候比 Genetic algorithm 好

(四)、不論是 DQN 或是 Genetic algorithm 都能與 Naïve algorithm 相抗衡

(五)、Genetic algorithm 都只是略微勝出

柒、結論

一、探討先手與後手的利與弊

在特定條件下，先手能夠「完全制霸」對手，也就是存在一種出牌順序，使得先手必勝，如下表所示。

先手牌組	紅色停止	紅色迴轉	紅 5
後手牌組	黑色+4	藍色+2	Wild card

先手只需要出「停止」、「迴轉」再出「紅 5」，先手就贏了，若存在先手必勝策略，後手將無機會逆轉，造成先手勝率較高。

二、探討各演算法之間的勝率

Naïve offensive algorithm 對上 Naïve defensive algorithm 時，defensive 有較高的機率勝出，這是因為 offensive algorithm 常常把功能牌先丟完了，使得最後沒有功能牌可以丟。根據筆者的自身經歷，多數人喜歡把功能牌留到最後一刻再丟，通常大家會傾向於「存功能牌」，並在遊戲結束前盡速丟出，就像 Naïve defensive algorithm 一樣。

因為 Naïve algorithm 是「死」的，而 Deep-Q-Learning 與 Genetic-Network 是「活」的，經過訓練後，神經網路能夠佔 Naïve algorithm 的小便宜。美中不足的是，UNO 是個「資訊不對稱」的賽局，就算能夠掌握對手的行為模式，也沒辦法掌握對手的下一步棋，因為神經網路沒辦法看穿對手的牌，沒辦法像 AlphaGo 或是 Deep Blue 那樣「輾壓」對手。

Dummy algorithm 遇到所有演算法幾乎都被「輾壓」，正如其名，他是設計來「被輾壓」的演算法，不論遇到誰都輸，名副其實的「對戰假人」。

三、探討 Deep-Q-Network 與 Genetic-Network 的利與弊

如果把 Deep-Q-Network 譬喻成人類，而 Genetic-Network 譬喻成細菌的話，UNO 譬喻成大自然的話，我們可以用相當有趣的角度來看待這場遊戲。

細菌使用基因作為記憶，酵素作為武器，向人類宣戰；原始的人類也使用基因作為記憶，免疫系統作為武器，向細菌宣戰；神農氏跨出了偉大的一步，使用大腦作為記憶，草藥作為武器，向細菌宣戰；微生物學家使用文字作為記憶，抗體、抗生素以及噬菌體作為武器，向細菌宣戰。

在這數千年的歷史中，人類不斷進步，發明新的武器與細菌對戰；在這數千年的歷史中，細菌也不斷進步，演化出新的基因與人類對戰，這不就跟電腦裡發生的事情一模一樣嗎？Deep-Q-Learning 不斷想出新主意去對付 Genetic algorithm，Genetic algorithm 不斷演化出新策略去對付 Deep-Q-Learning，好比人類研發出新疫苗對抗細菌，細菌演化出新偽裝去對抗人類！

Genetic algorithm 時常劇烈起伏，好比細菌的攻擊能力時強時弱；Deep-Q-Learning 比 Genetic algorithm 穩定許多，好比人類懂得使用語言、文字以及記憶來儲存所學所能，凡是學會的，就會永遠保存在在腦中。

下表簡潔有力的表達了兩種演算法之間的利弊得失，以及之間的關係。

	Genetic algorithm	Deep-Q-Learning
優勢	神經網路構造簡單	勝率較為穩定 搭配 Behavior cloning 效果顯著
劣勢	增加隱藏層深度成效不彰 勝率十分不穩定	神經網路構造複雜
好比	細菌	人類

四、總結

昔日，Deep Blue 使用自己強大的運算效能，窮舉每一種狀況，最終攻克了西洋棋領域；近年，Alpha Go 使用卷積神經網路以及 Q-Learning，演算哪個落點勝率最高，終於攻克了圍棋領域。

本研究設計了諸多演算法，經過實驗後得知，這些參賽者中的黑馬是 Deep-Q-Learning，不僅勝率穩定，還能夠藉由 Behavior cloning 增加勝率，是目前最強的演算法。

繼 Deep Blue、Alpha Go 之後，電腦又將矛頭指向全世界最暢銷的桌遊—UNO，蓄勢待發，準備攻克這個領域！

捌、參考資料

參考資料	連結
UNO on wiki	https://zh.wikipedia.org/wiki/UNO#%E5%85%A9%E4%BA%BA%E5%B0%8D%E7%8E%A9%E7%9A%84%E8%A6%8F%E5%89%87
Deep-Q-Learning paper	https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf
Deep-Q-Learning tutorial	https://www.mlq.ai/deep-reinforcement-learning-q-learning/
Genetic algorithm on wiki	https://en.wikipedia.org/wiki/Genetic_algorithm
RL cards	https://arxiv.org/pdf/1910.04376.pdf
Q-Learning on wiki	https://en.wikipedia.org/wiki/Q-learning
Tensorflow	https://www.tensorflow.org/api_docs/python