程式人

用十分鐘瞭解

《AlphaGo 的幾個可能弱點》

（研究過 AlphaGo 論文的程式人給李世石的幾點建議）

陳鍾誠

2016 年 3 月 13 日

現在是 2016/3/13 上午 10:30

- Google 的圍棋程式 AlphaGo 已經連贏了李世石三盤！

很多人都在研究 AlphaGo 的下法

但是這些對李世石而言

- 似乎沒有明顯直接的幫助

因為

- 李世石已經是圍棋界數一數二的高手
- 其他人又怎麼能指導他呢？

但是、這難道代表

- 沒有人能幫助《李世石》了嗎？

我想未必

我認為

- 李世石現在最需要的

不是圍棋專家

- 也不是柯潔的幫助！

而是

- 熟悉人工智慧技術的電腦專家！

這個專家

- 必須努力的研究 AlphaGo 的論文！

然後、從論文中找出

- AlphaGo 可能的缺陷

把這些缺陷列出來

- 清楚地給李世石看

然後

- 讓他在對局的時候
- 能夠利用這些可能缺陷，測試出 AlphaGo 真正的致命弱點！

只要找到一個弱點

- 就有機會破了 AlphaGo 的局

因為、在高手對戰的時候

- 任何一個弱點都足以致命！

既然現在

- 沒有真正的人工智慧專家跳出來

那就先讓我

- 這個半吊子的專家，

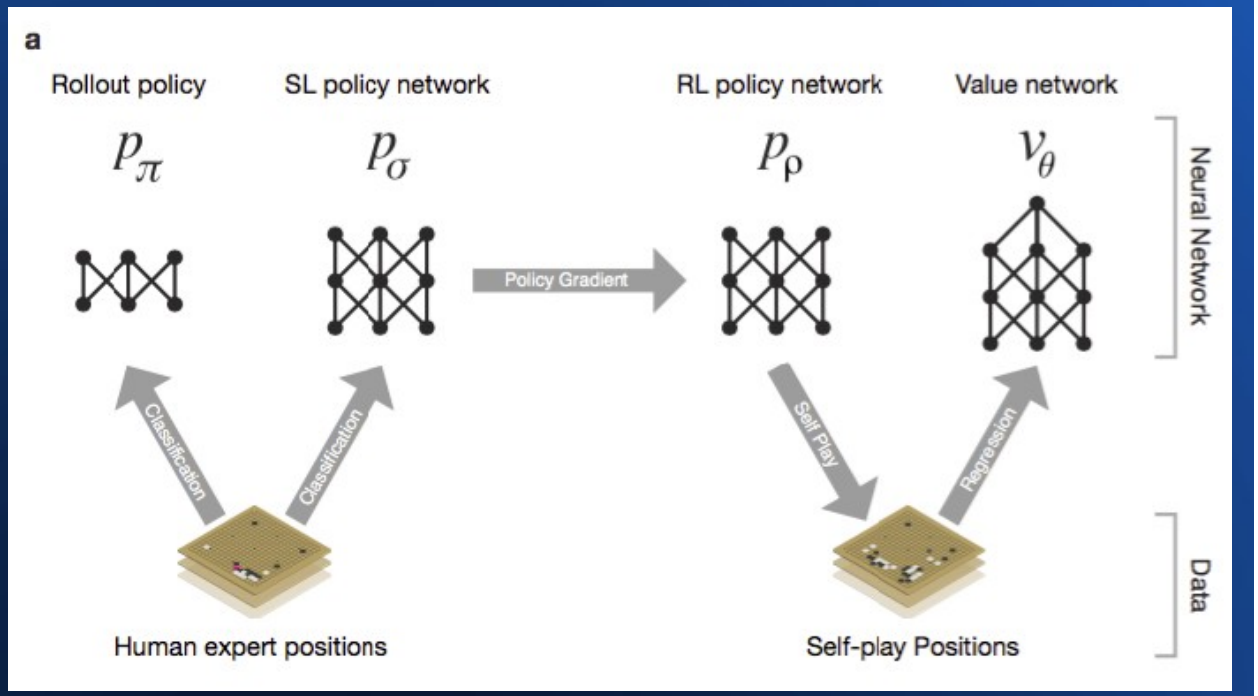
  先來研究一下論文。

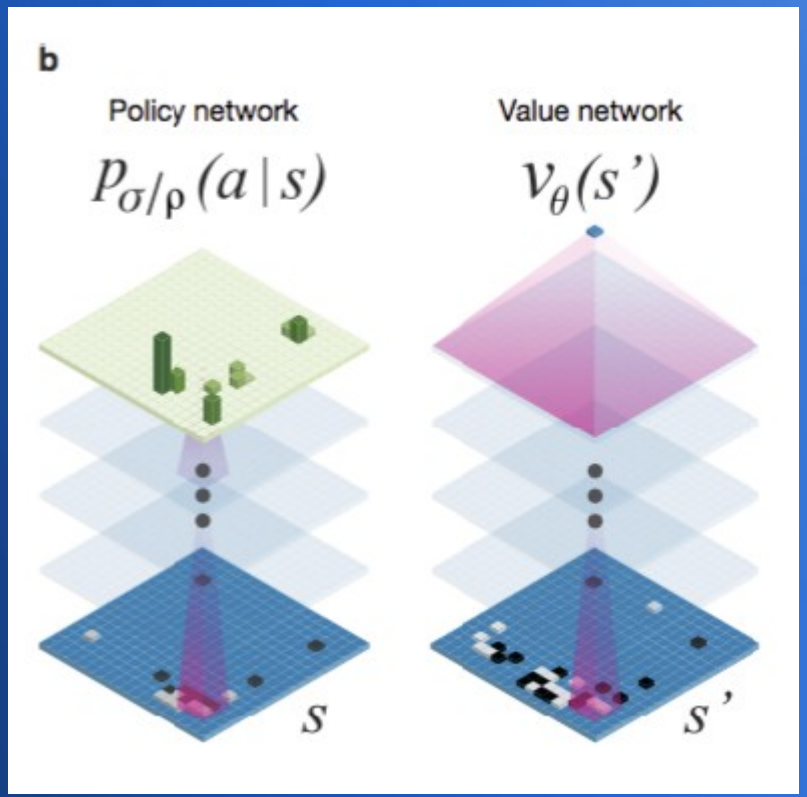
想辦法

- 找出 AlphaGo 的可能弱點吧！

在 AlphaGo 的那篇論文裡面

有幾個關鍵的圖形


a
Rollout policy
SL policy network
RL policy network
Value network
p_π
p_σ
p_ρ
v_θ
Policy Gradient
Classification
Classification
Self Play
Regression
Neural Network
Data
Human expert positions
Self-play Positions


b
Policy network
Value network
$p_{\sigma/\rho}(a \mid s)$
$v_\theta(s')$
s
s'

這些圖形

- 記載著 AlphaGo 的設計方法

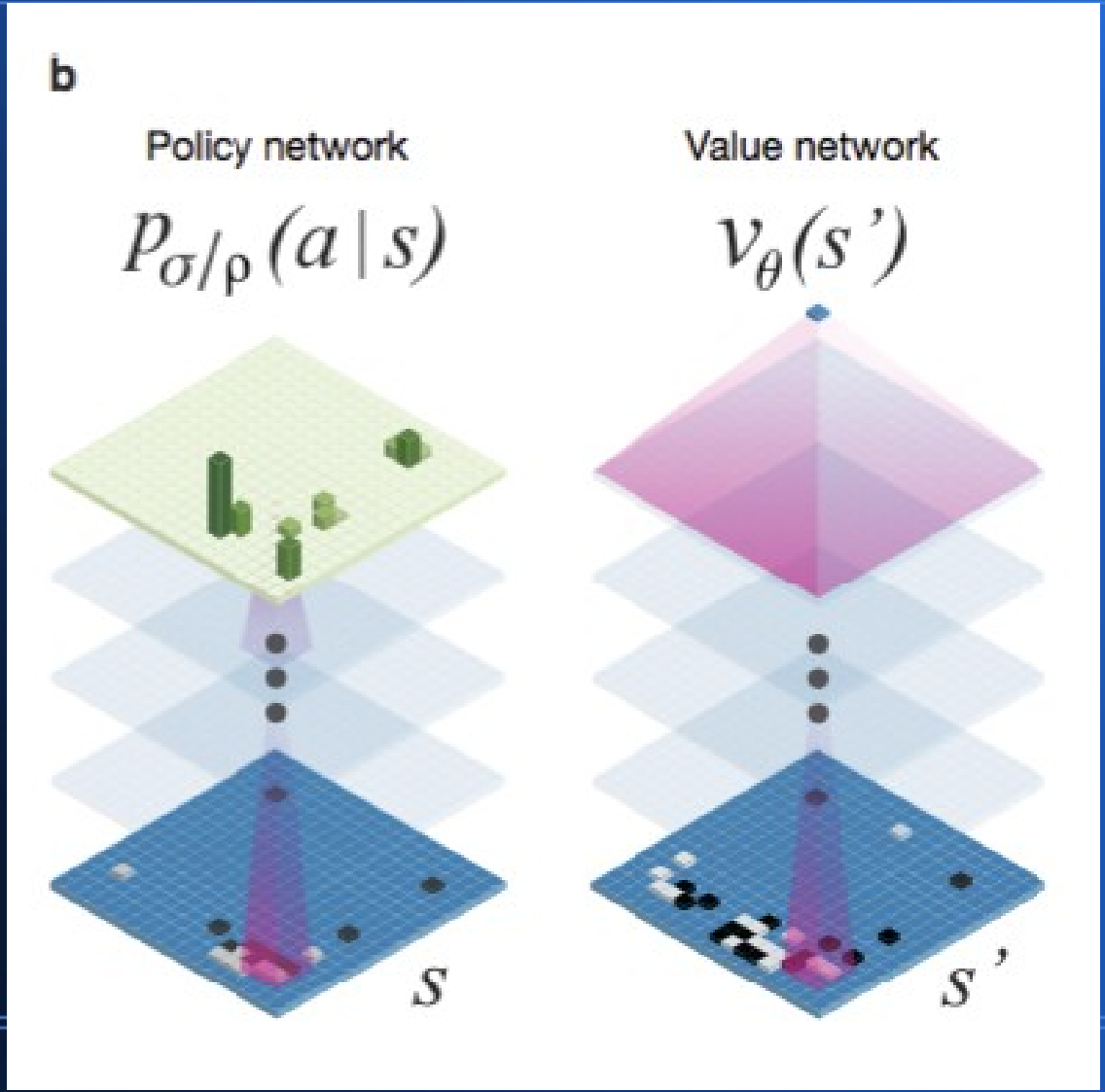
我相信

- 仔細研究這些圖形所展現的方法
- 應該就能看出弱點之所在！

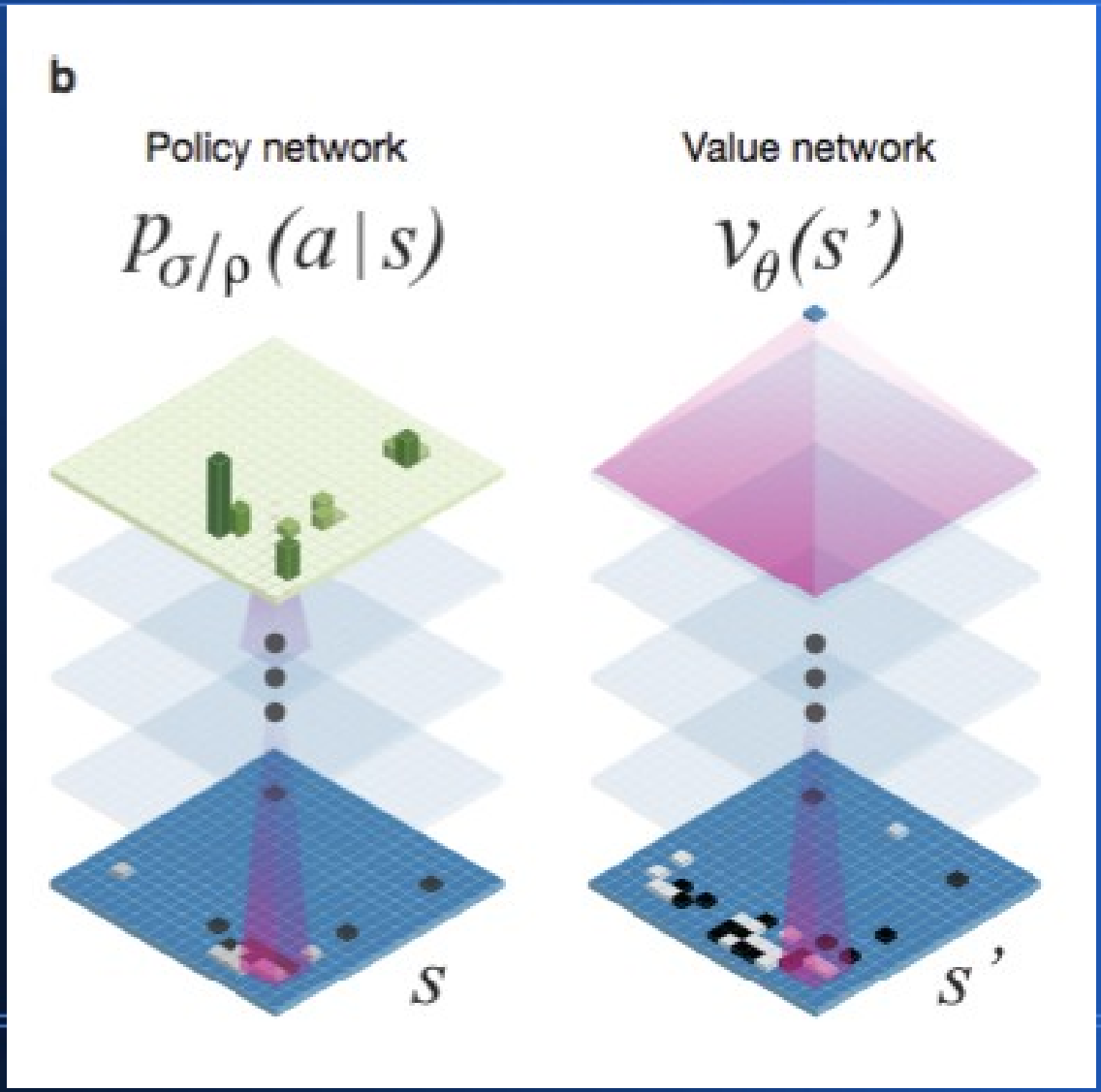
現在、請容我解說一下

- 這些圖的意義！

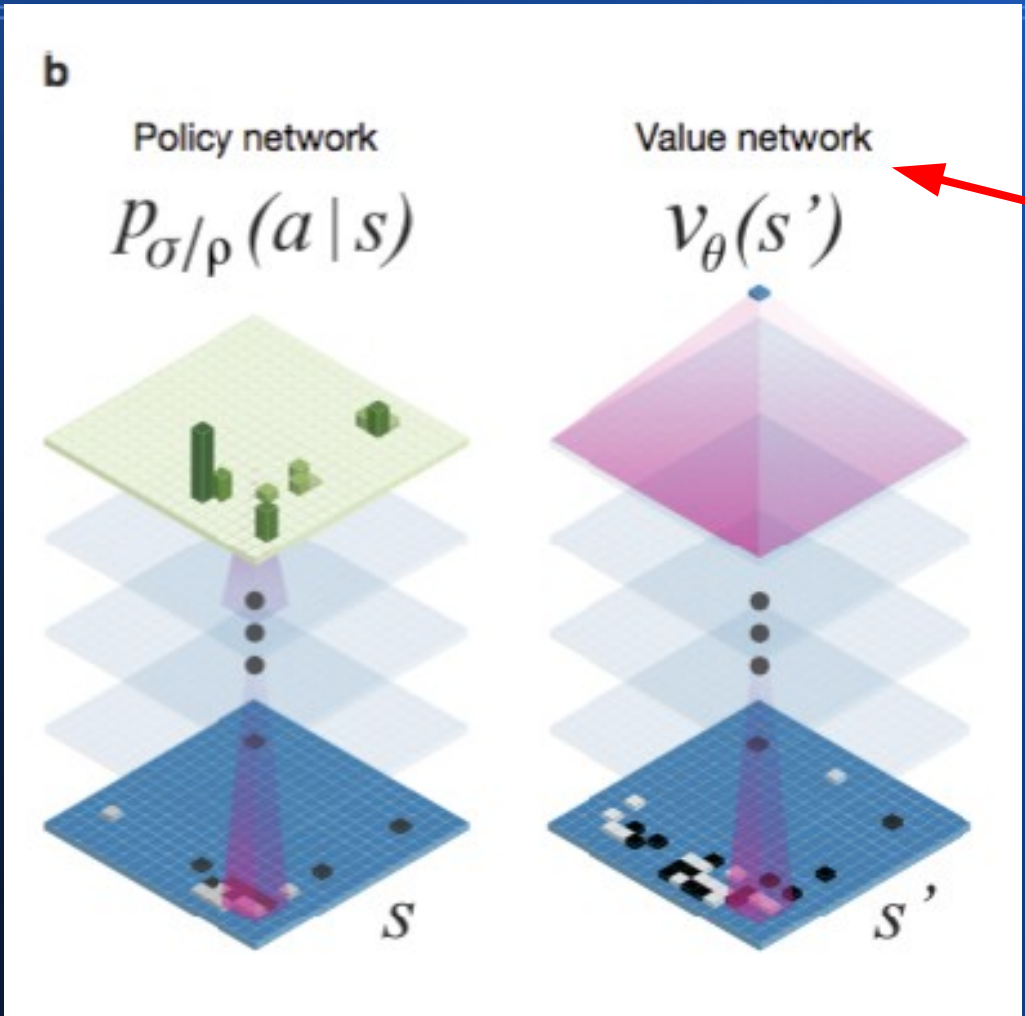
首先、我認為最上層的，是這張圖



圖中顯示兩個 AlphaGo 經由學習所得到的函數



第一個稱為 - 價值網路
Value Network



價值網路訓練出來的

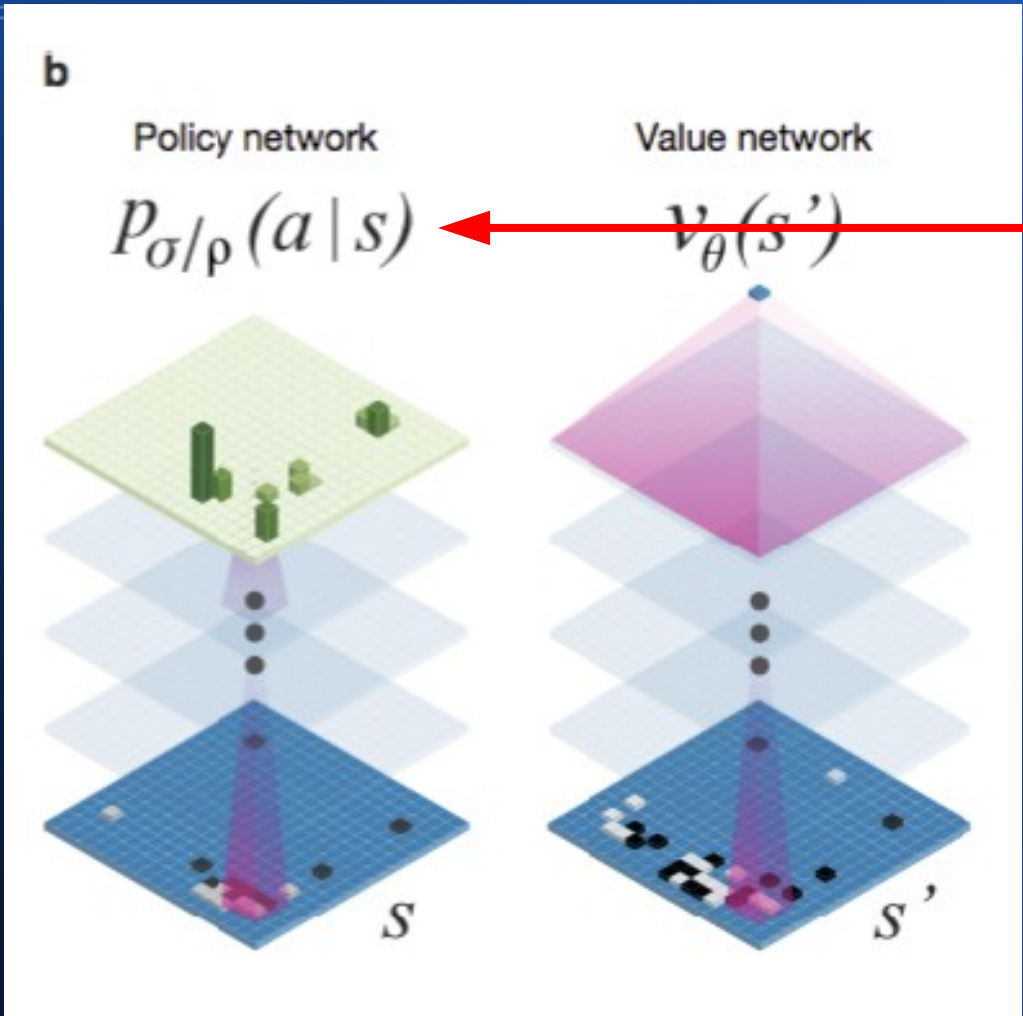
V(s’ ）函數，可以計算個盤面 s’ 的分數。

換句話說：

V(s’ ）就是《盤面評估函數》

對於不熟悉電腦下棋技術的人，可以參考《用十分鐘瞭解電腦到底是怎麼下棋的》這篇投影片

第二個稱為 - 策略網路
Policy Network



策略網路訓練出來的

P(a|s) 函數，可以
告訴 AlphaGo 對手
比較可能下哪一步？

於是 AlphaGo 就可以對
這些可能的位置進行
更深入的搜尋與評估

注意：這個對手也可能是
AlphaGo 自己，所以也能
用來提供進一步的搜尋點

對於不熟悉電腦下棋技術的人，可以參考《用十分鐘瞭解電腦到底是怎麼下棋的》這篇投影片

假如 AlphaGo 這兩個函數當中

- 有任何一個有缺陷

  人類就有可能找到突破點！

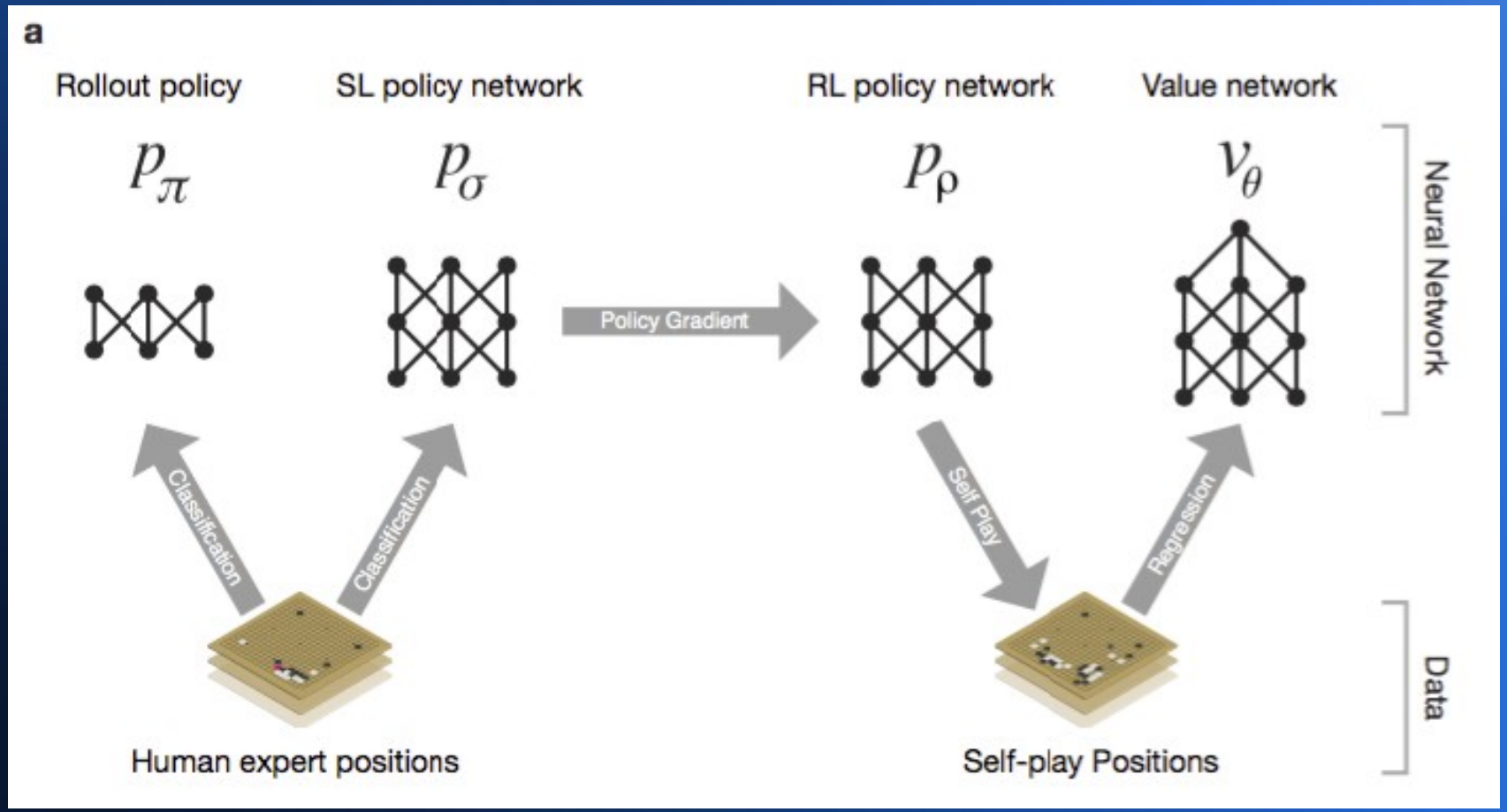
但是、真的有缺陷嗎？

- 請讓我們繼續研究下去！

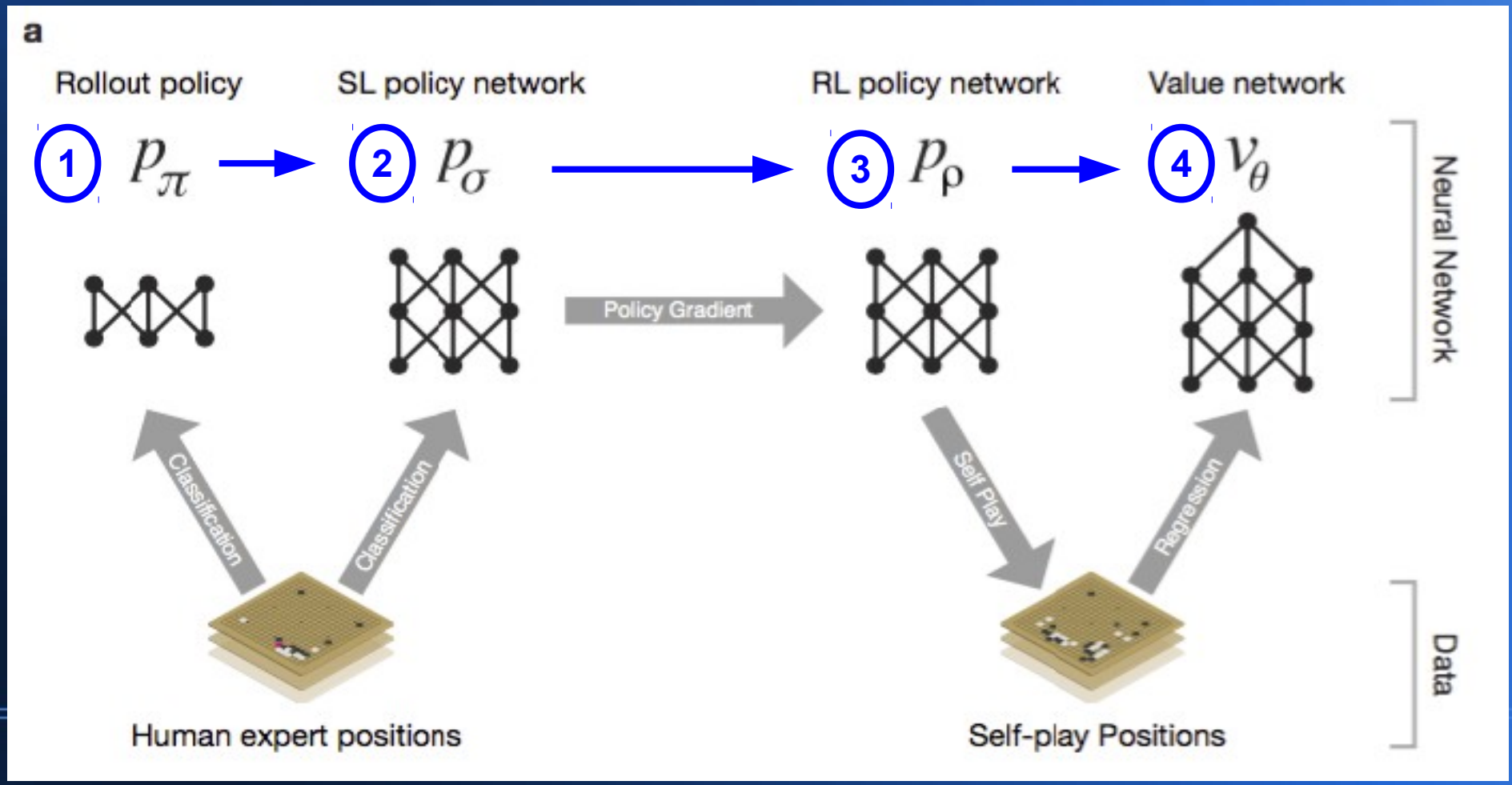
在該圖片的下方
有著 AlphaGo 運作原理的描述

Figure 1: **Neural network training pipeline and architecture. a** A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data-set of positions. A reinforcement learning (RL) policy network p_ρ is initialised to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (i.e. winning more games) against previous versions of the policy network. A new data-set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (i.e. whether the current player wins) in positions from the self-play data-set. **b** Schematic representation of the neural network architecture used in *AlphaGo*. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

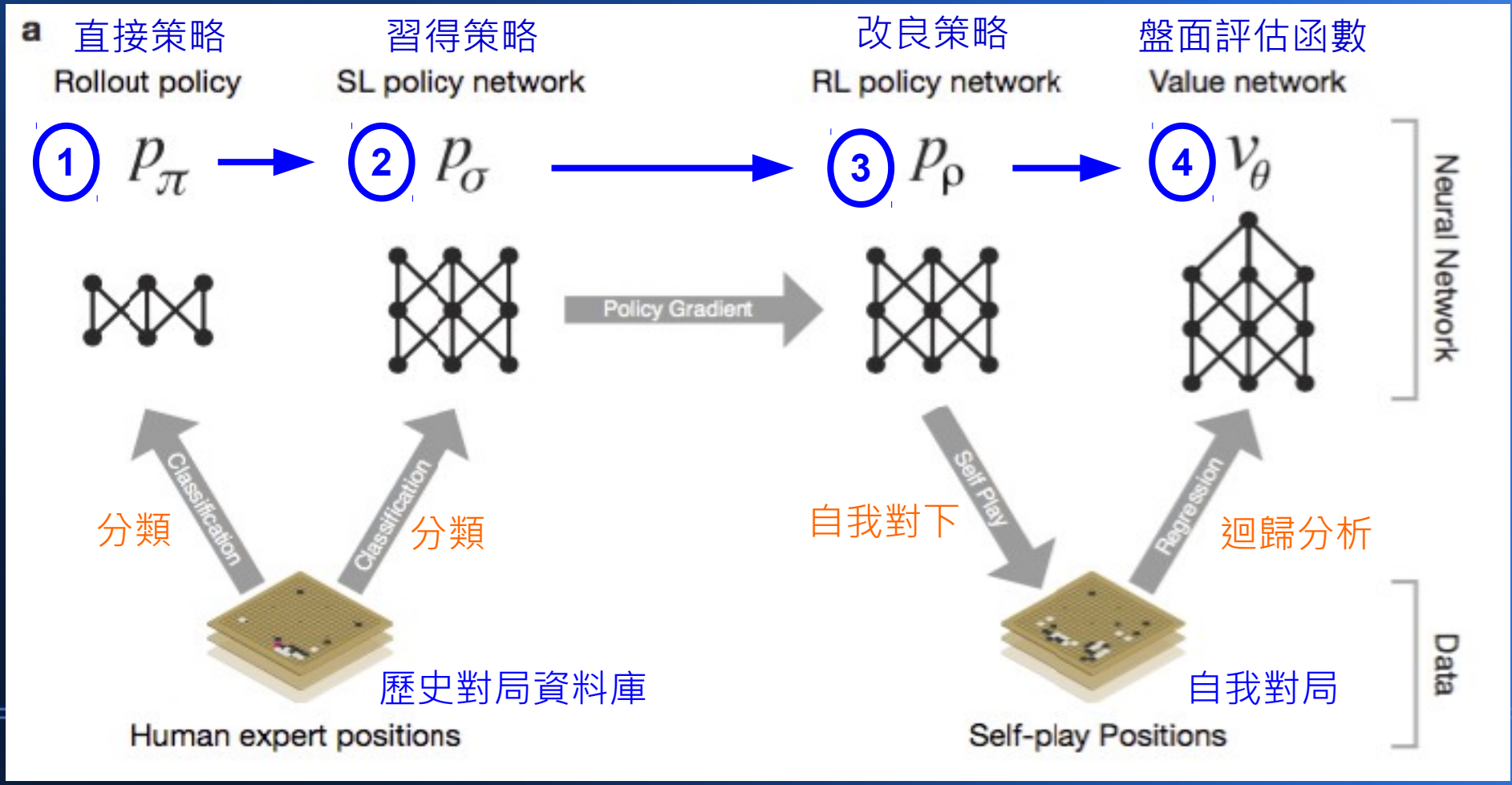
這個描述、其實主要牽涉到圖中的(a)部分



這個圖必須由左向右看

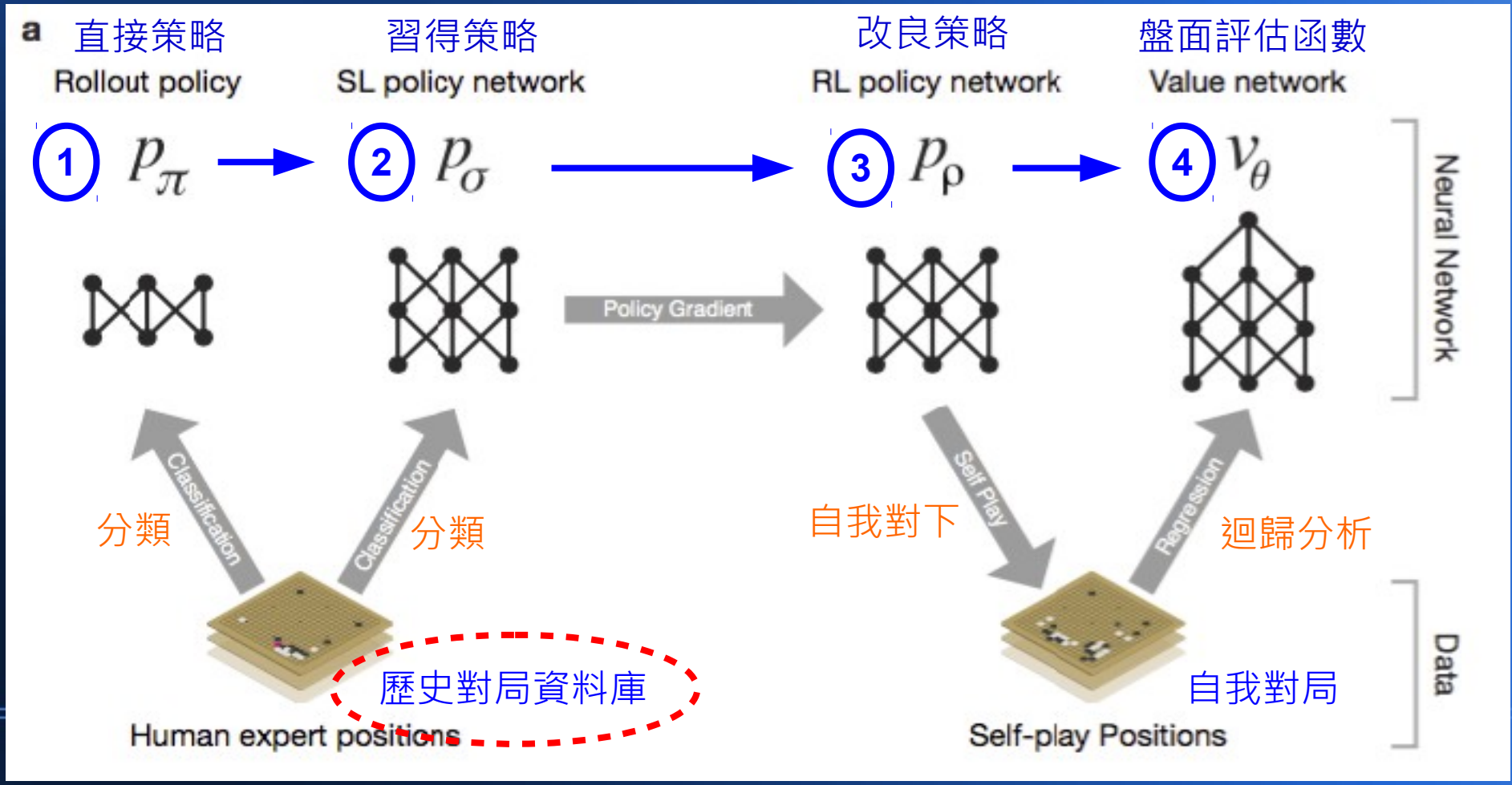

a
Rollout policy
SL policy network
RL policy network
Value network
1
p_π
2
p_σ
3
p_ρ
4
v_θ
Policy Gradient
Neural Network
Classification
Classification
Self Play
Regression
Data
Human expert positions
Self-play Positions

AlphaGo 的整體訓練架構如下



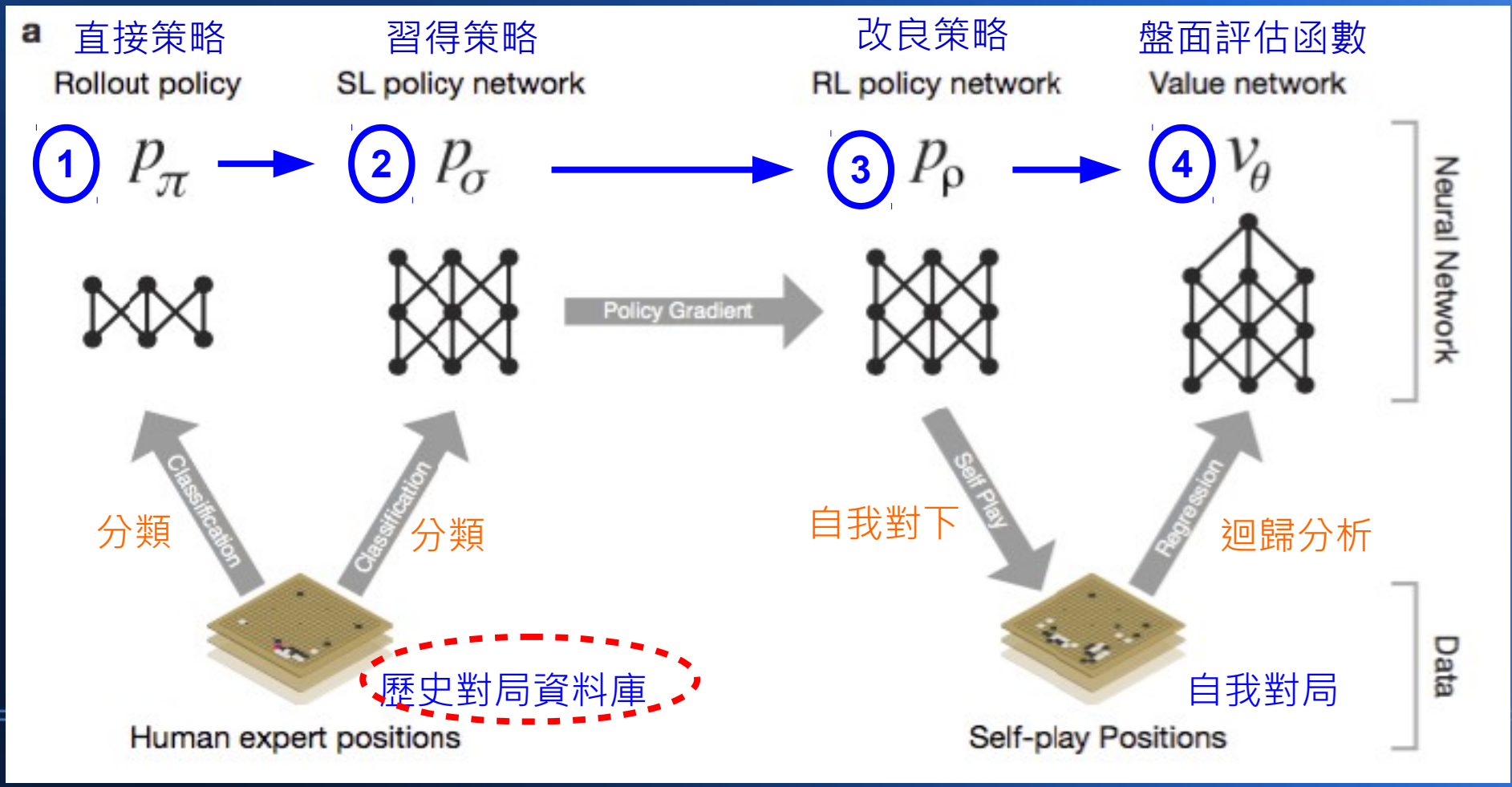
首先看歷史對局資料庫的部分

這就是 Google 蒐集的所有歷史對局的完整過程，應該是很大的對局資料庫，或許包含了幾百年或上千年的歷史對局



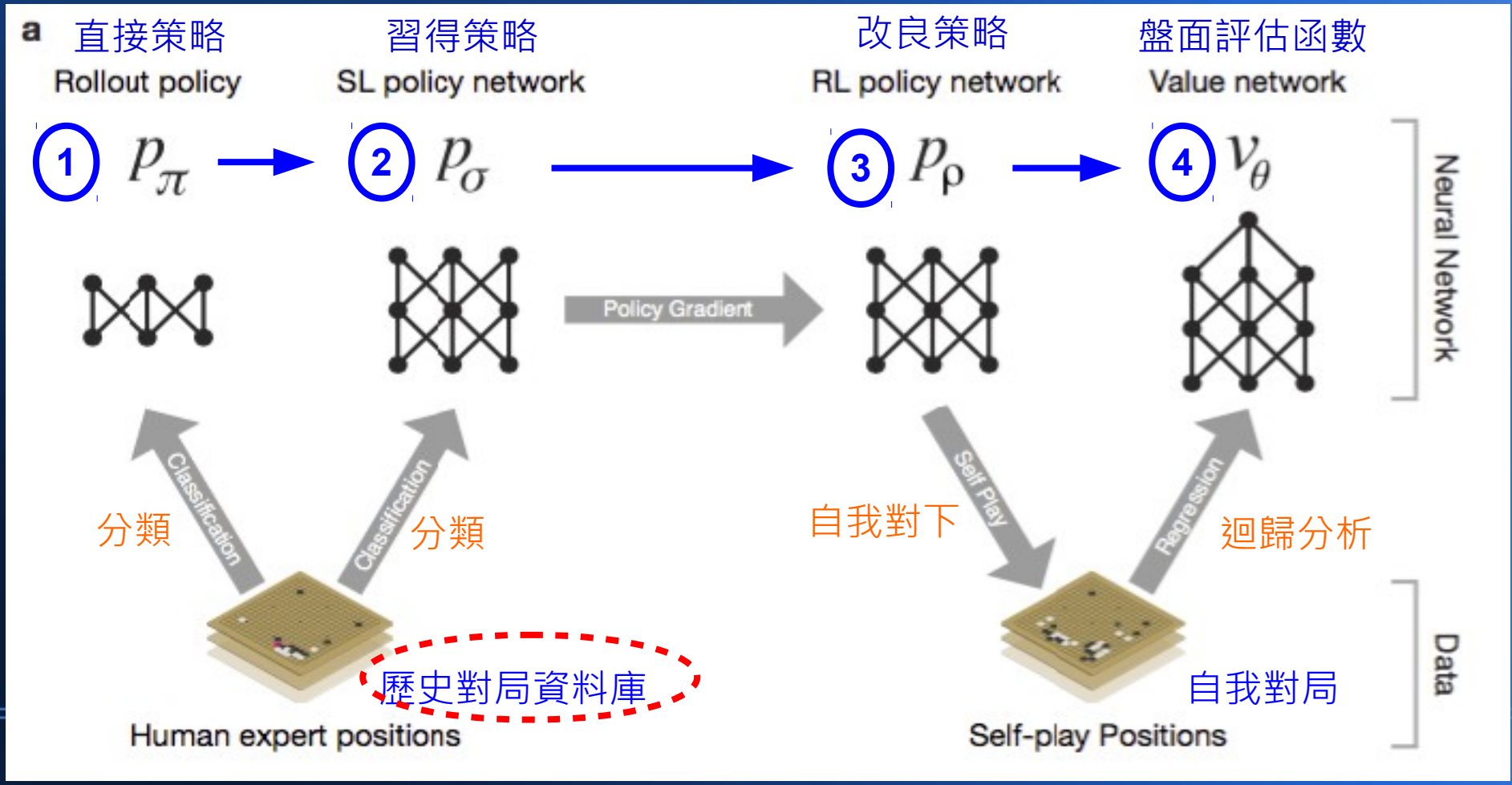
AlphaGo 利用這個對局資料庫

1. 進行分類 (Classification) 之後得到《直接策略》
2. 然後再用神經網路一般化之後得到《習得策略》
3. 接著用強化學習 RL 《自我對下》得到《改良策略》
4. 最後利用《迴歸》從中得到《盤面評估函數》



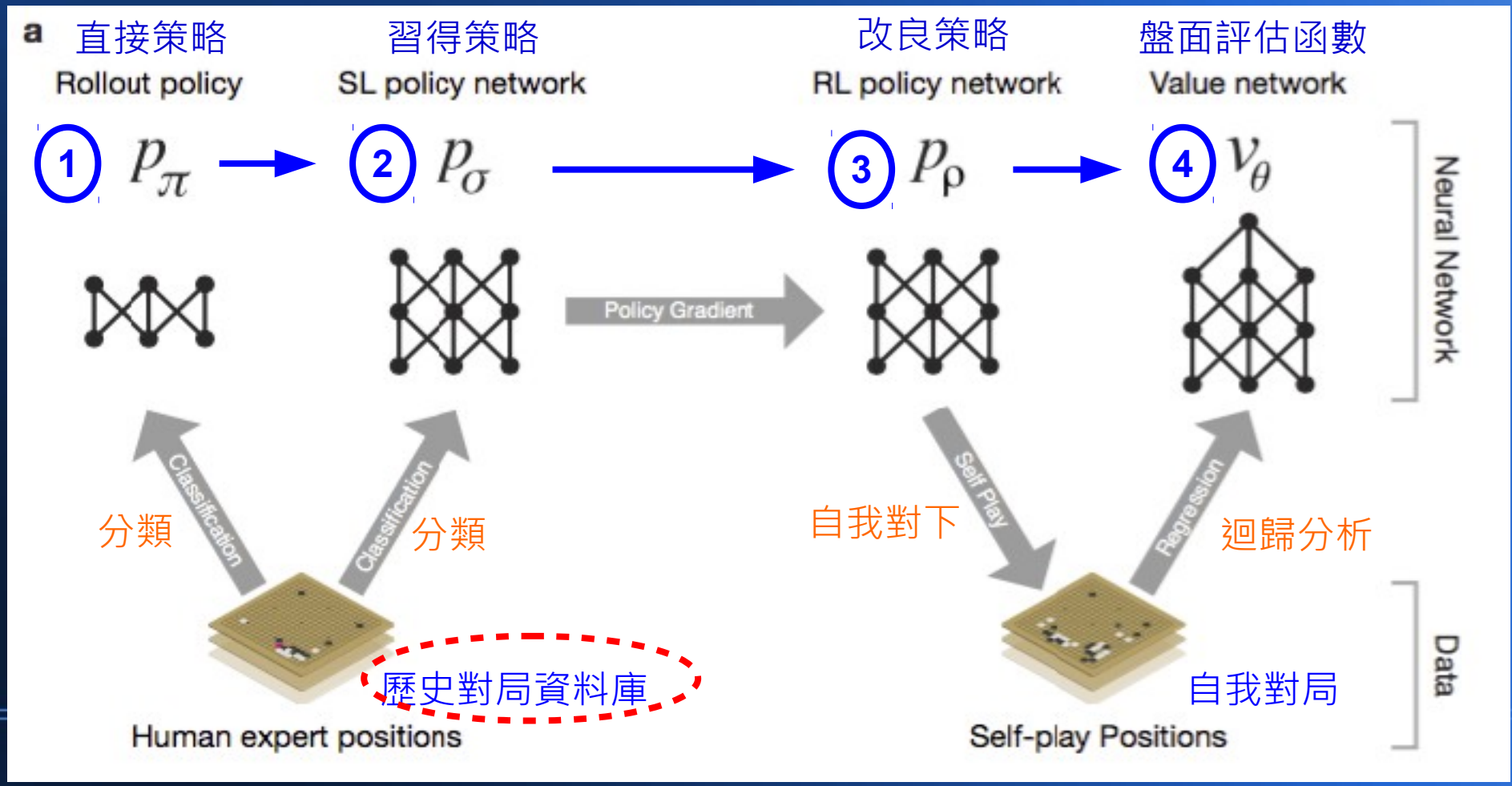
在這四個《機器學習》的過程中
只要抓到任何一個弱點，都可以破解 AlphaGo 的對局

1. 進行分類 (Classification) 之後得到《直接策略》
2. 然後再用神經網路一般化之後得到《習得策略》
3. 接著用強化學習 RL 《自我對下》得到《改良策略》
4. 最後利用《迴歸》從中得到《盤面評估函數》



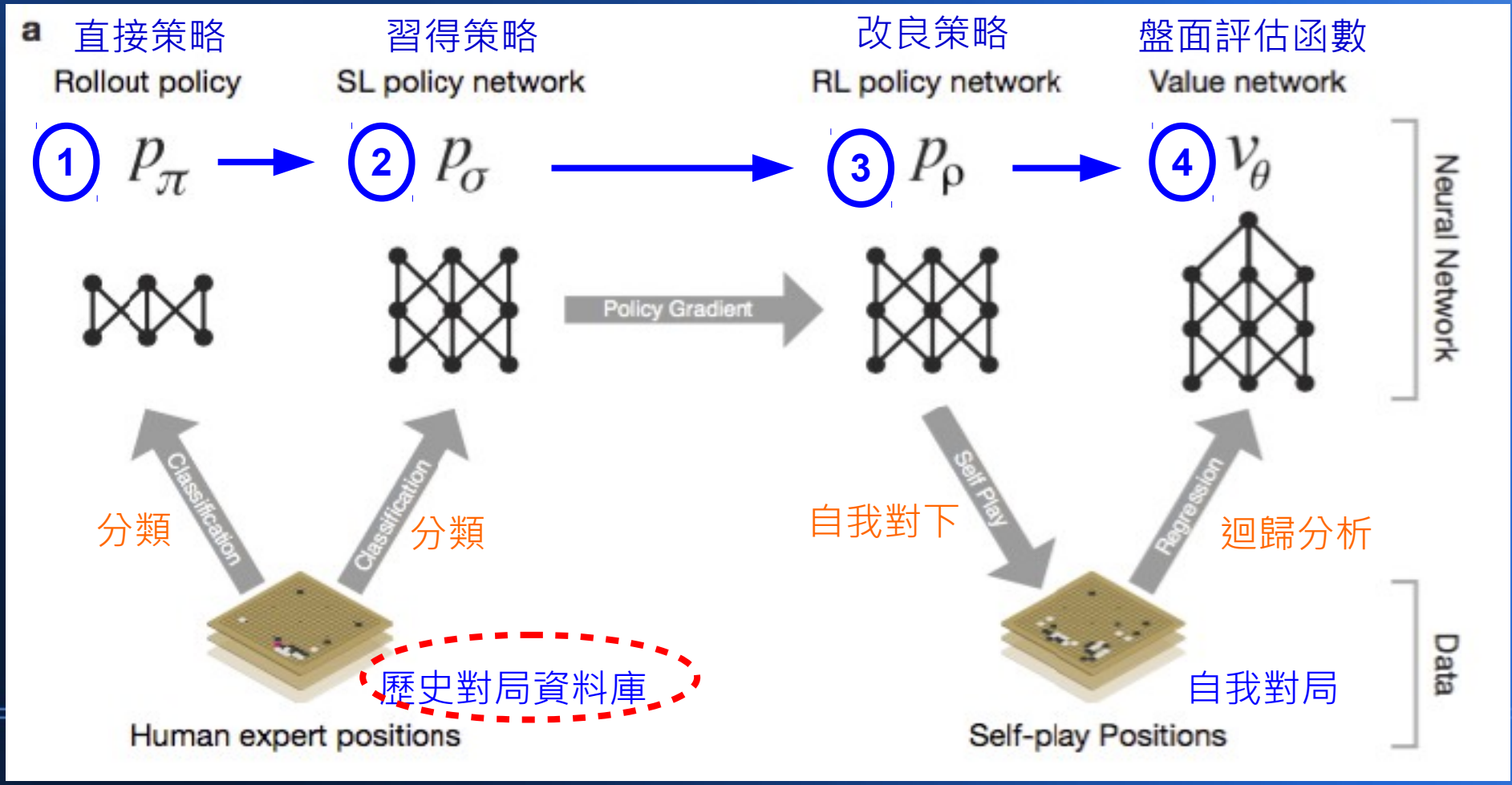
但問題是、弱點會出現在哪裡呢？

1. 進行分類 (Classification) 之後得到《直接策略》
2. 然後再用神經網路一般化之後得到《習得策略》
3. 接著用強化學習 RL 《自我對下》得到《改良策略》
4. 最後利用《迴歸》從中得到《盤面評估函數》



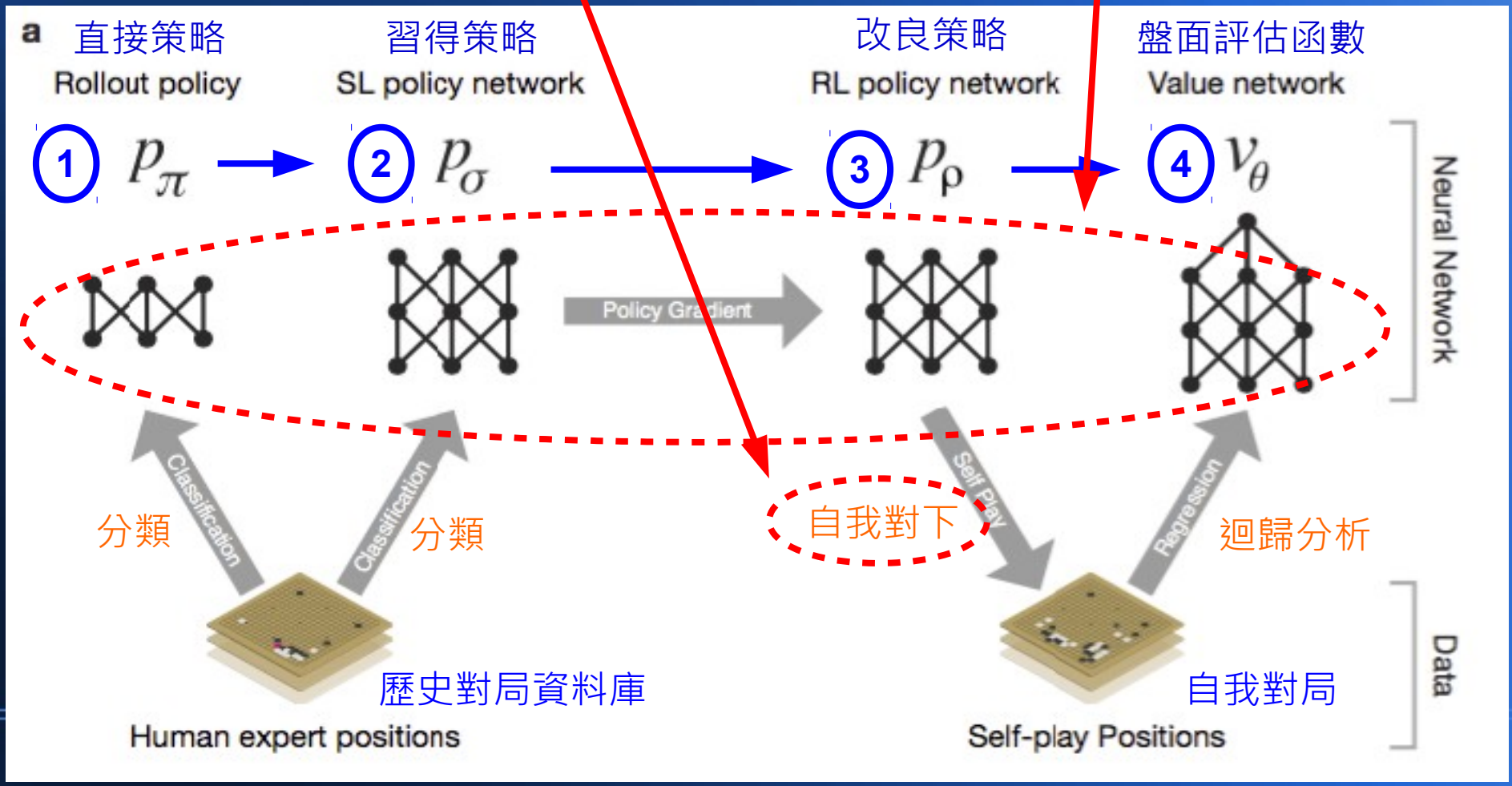
關於這點、讓我們進一步研究一下 AlphaGo 所使用的方法

1. 進行分類 (Classification) 之後得到《直接策略》
2. 然後再用神經網路一般化之後得到《習得策略》
3. 接著用強化學習 RL 《自我對下》得到《改良策略》
4. 最後利用《迴歸》從中得到《盤面評估函數》



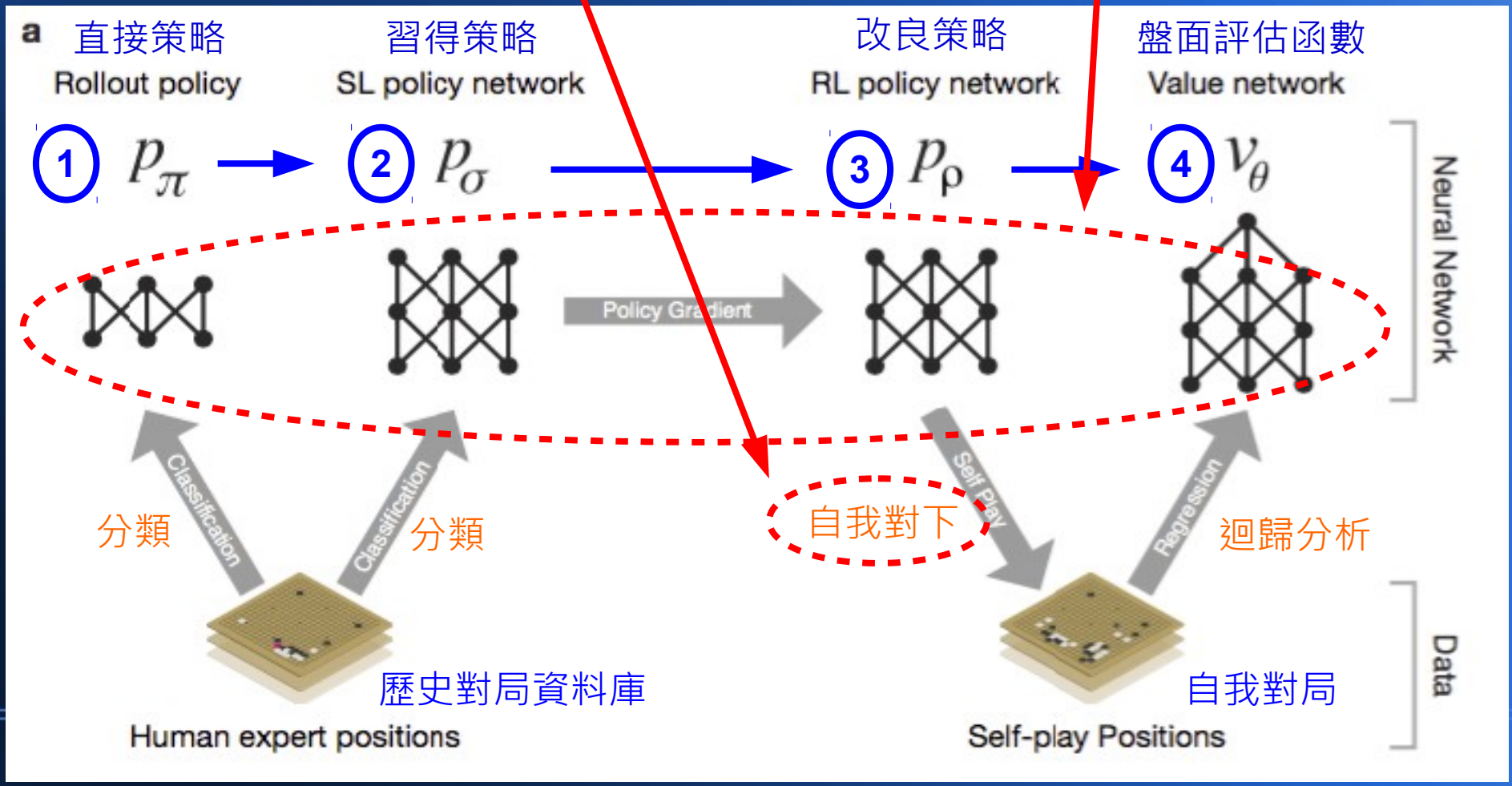
AlphaGo 的比較特別的方法有兩個

1. 蒙地卡羅對局搜尋法（MCTS）2. 深捲積神經網路（DCNN）



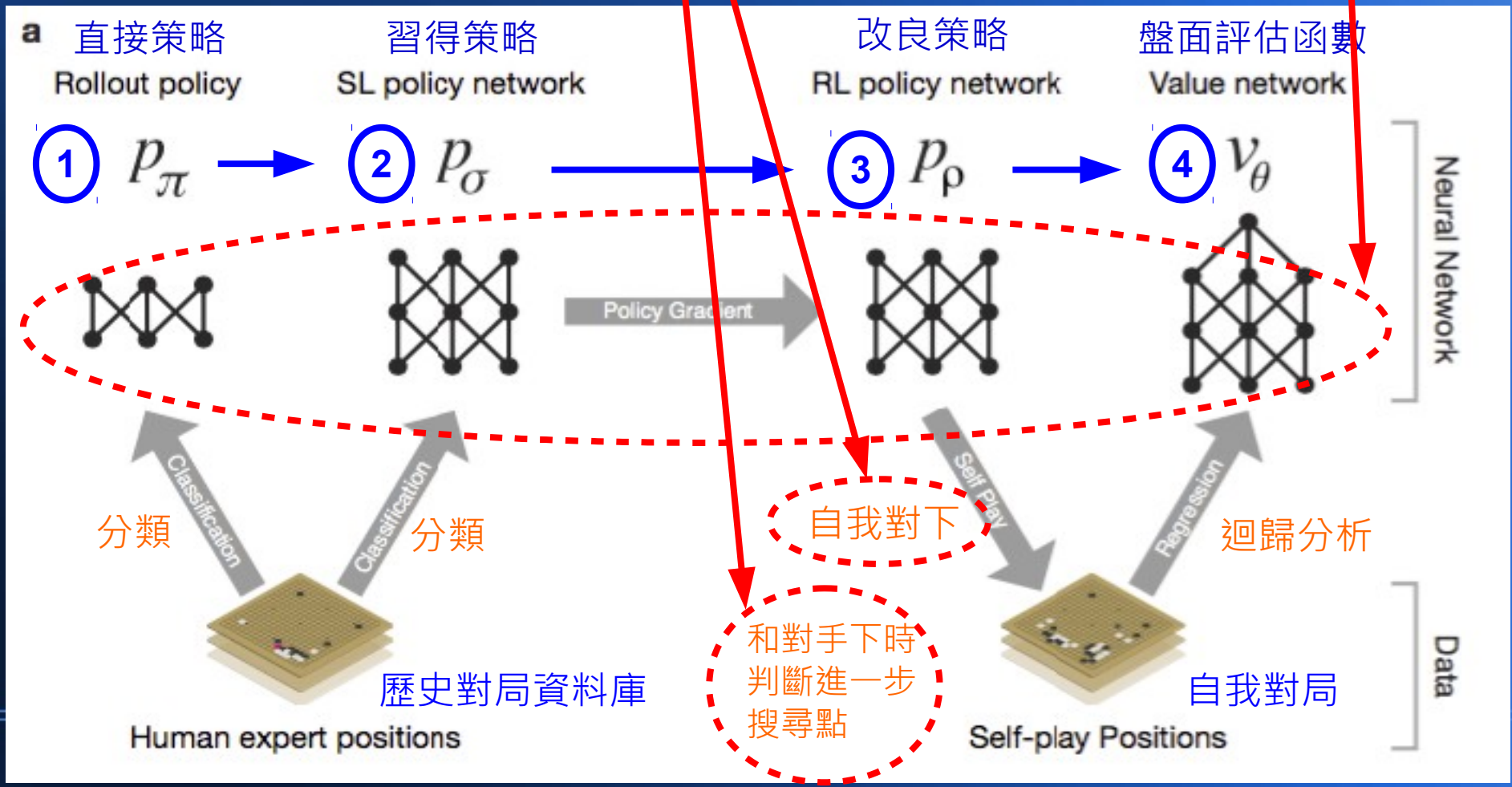
1. 蒙地卡羅對局搜尋法（MCTS）2. 深捲積神經網路（DCNN）

1. 《自我對局》時是採用蒙地卡羅搜尋法，這樣可以避免多層對局的複雜度指數成長問題。
2. AlphaGo 的很多網路學習都是用深捲積神經網路 (DCNN)， DCNN 可以照顧到整體與局部。



1. 蒙地卡羅對局搜尋法（MCTS）2. 深捲積神經網路（DCNN）

1. 關於 MCTS 可以參考 Jeff Bradberry 的 Introduction to Monte Carlo Tree Search這篇文章
2. 關於 DCNN 可以參考 尹相志 的 淺談Alpha Go所涉及的深度學習技術這篇文章



接著、讓我們針對 AlphaGo 所採用的這兩個方法做初步的介紹

1. 蒙地卡羅對局搜尋法(MCTS) 2. 深捲積神經網路(DCNN)

以便能理解方法的強處與缺陷

然後從可能的缺陷中找出破綻

首先從《蒙地卡羅對局搜尋法》(MCTS)下手

- 這種方法是一種隨機式的方法
- 和傳統採用 MinMax 對局樹與 AlphaBeta 修剪法有明顯不同

對於不熟悉電腦下棋技術的人，可以參考《用十分鐘瞭解電腦到底是怎麼下棋的》這篇投影片

由於圍棋的搜尋樹太大

- 如果用 MinMax 對局搜尋樹
- 加上 AlphaBeta 修剪法
- 會導致只能搜尋幾層，而沒有辦法深入搜尋
- 於是棋力就無法提高！
- 這是為何不採用 MinMax+AlphaBeta 的原因。

為了處理這個問題

- 研究電腦圍棋的人，開始提出一種機率式的《蒙地卡羅對局搜尋法》（MCTS）。

- 其主要想法是：我不需要暴力的搜尋全部，而是用《樣本與結果》來指導我應該要往哪邊更深入的搜尋！

1. 關於 MCTS 可以參考 Jeff Bradberry 的 Introduction to Monte Carlo Tree Search這篇文章
2. 關於 DCNN 可以參考 尹相志 的 淺談Alpha Go所涉及的深度學習技術這篇文章

那麼、到底《樣本與結果》如何用來指導搜尋呢？

- 關於這點，細節請看最下方 Jeff Bradberry 的那篇文章，以下是關鍵想法：

Imagine, if you will, that you are faced with a row of slot machines, each with different payout probabilities and amounts. As a rational person (if you are going to play them at all), you would prefer to use a strategy that will allow you to maximize your net gain. But how can you do that? For whatever reason, there is no one nearby, so you can't watch someone else play for a while to gain information about which is the best machine. Clearly, your strategy is going to have to balance playing all of the machines to gather that information yourself, with concentrating your plays on the observed best machine. One strategy, called UCB1, does this by constructing statistical *confidence intervals* for each machine

$$\bar{x}_i \pm \sqrt{\frac{2 \ln n}{n_i}}$$

where:

- $\bar{x}_i$: the mean payout for machine i
- n_i: the number of plays of machine i
- n: the total number of plays

如果已經探索了一些樣本與結果，就可以利用信賴區間的上界來作為分數。

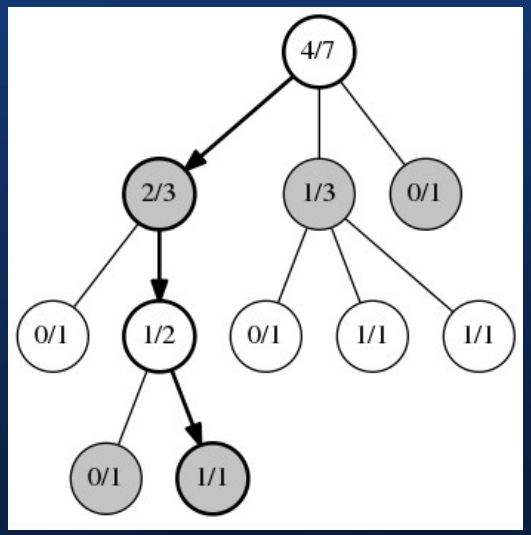
每次都選擇上界 (UCB) 最高的節點進行更深入的探索，讓這棵樹有系統的成長。

由於樣本數越多的話，上界範圍就會越小，所以不會對好的節點進行過多的探索。

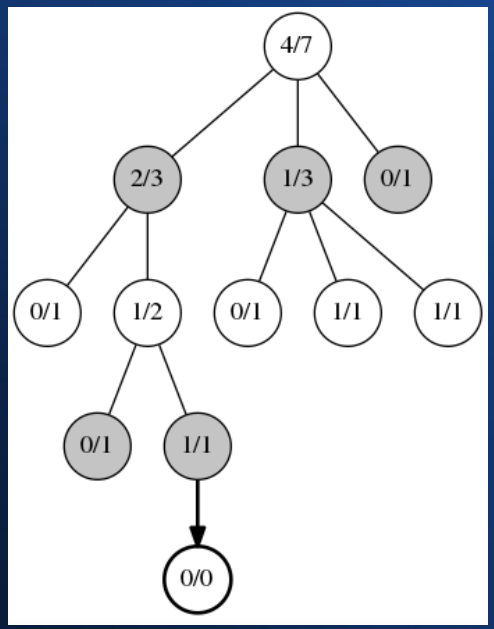
1. 關於 MCTS 可以參考 Jeff Bradberry 的 Introduction to Monte Carlo Tree Search這篇文章

以下是《蒙地卡羅對局搜尋法》(MCTS)的一個搜尋擴展範例

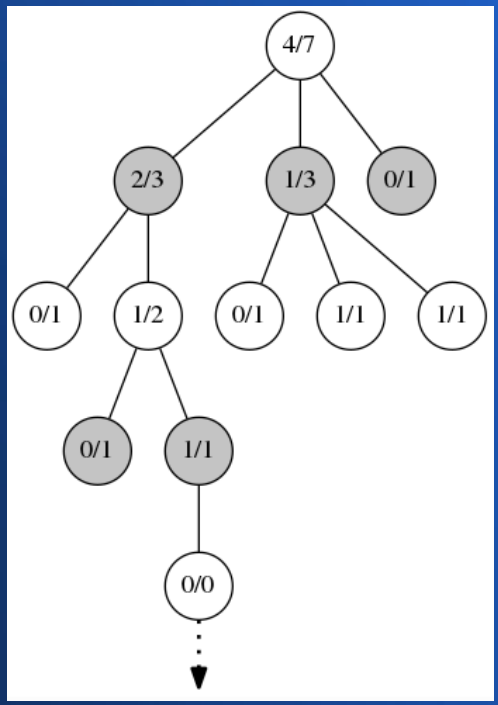
1. 選擇上界 UCB 最高的一條路直到末端節點



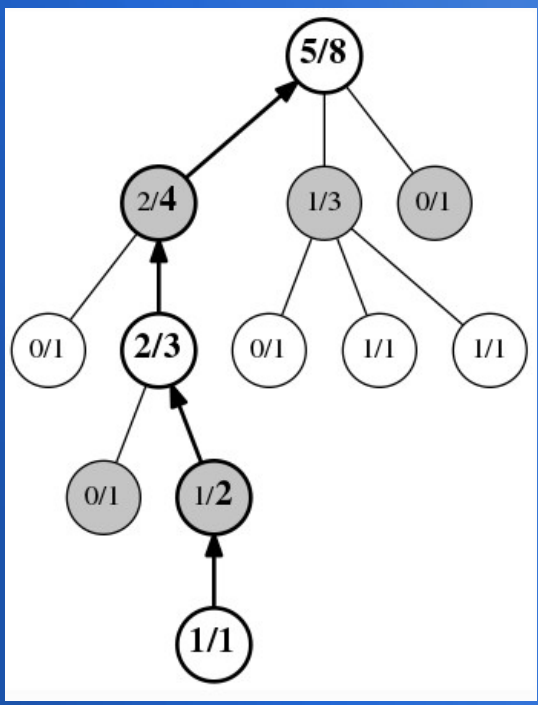
2. 對該末端節點進行探索 (隨機對下，自我對局)



3. 透過自我對局，直到得出本次對局的勝負結果



4. 用這次的對局結果，更新路徑上的勝負統計次數！

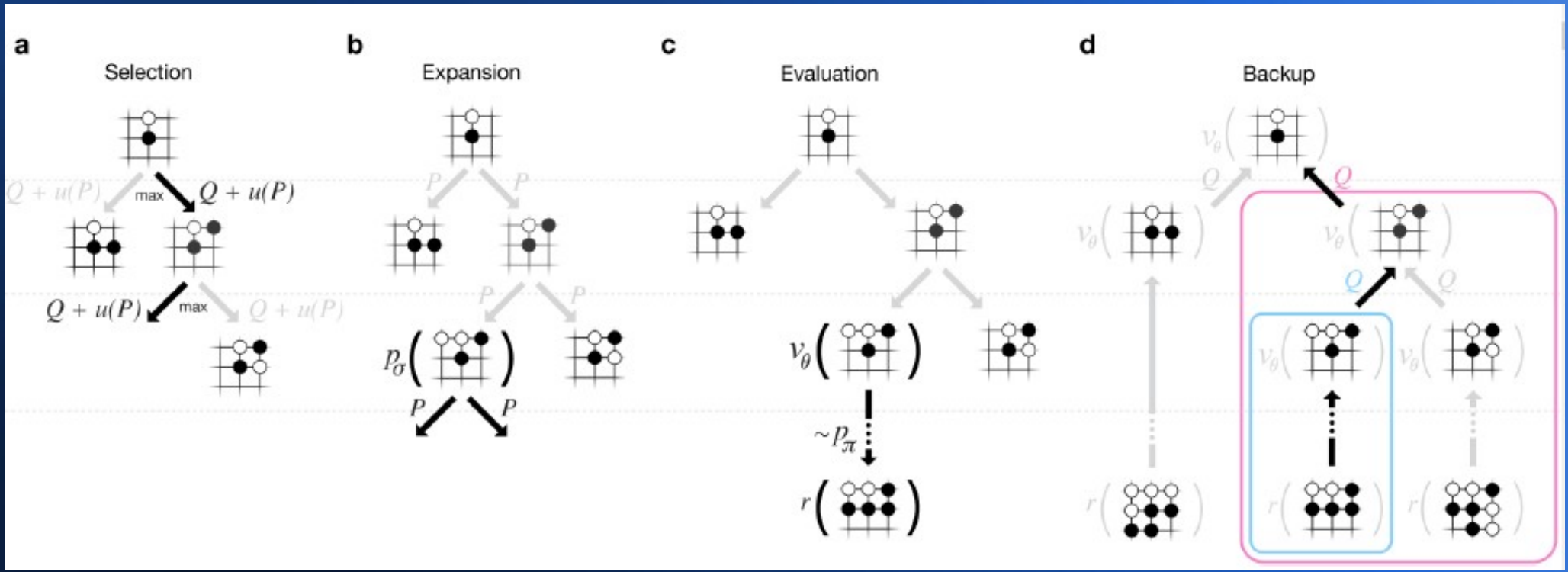


說明：上圖中白色節點為我方下子時的《得勝次數 / 總次數》之統計數據，

灰色的為對方下子的數據，本次自我對局結束後，得勝次數與總次數都會更新！

以下是 AlphaGo 論文中的蒙地卡羅搜尋樹範例

- 這個範例和上述範例類似，只是改以圍棋為例而已。

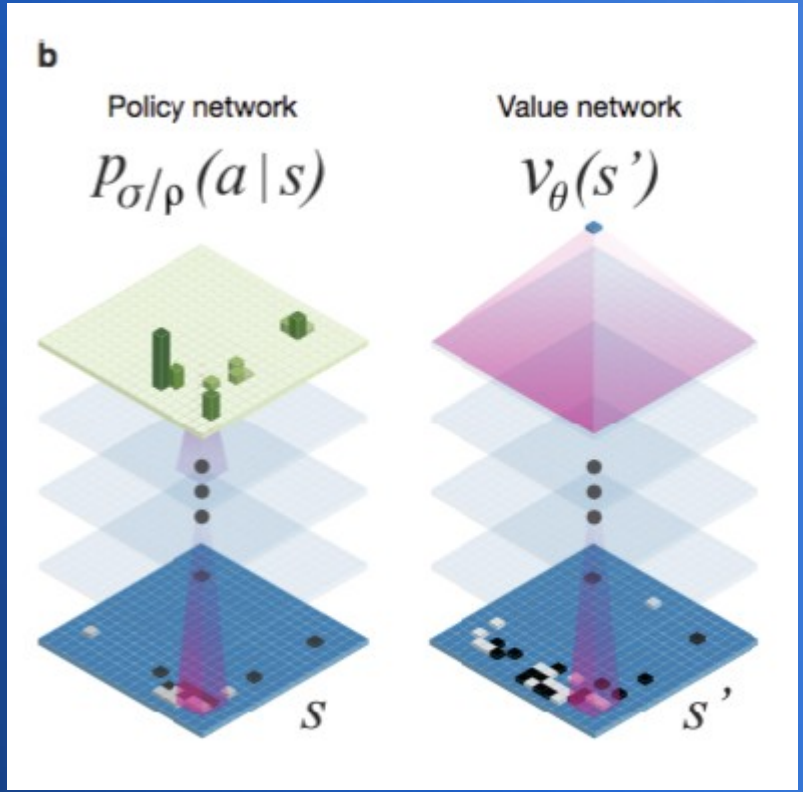


以下文字是對上圖的描述

Figure 3: **Monte-Carlo tree search in *AlphaGo*.** **a** Each simulation traverses the tree by selecting the edge with maximum action-value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b** The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c** At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d** Action-values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

這種《蒙地卡羅對局搜尋法》(MCTS)能讓電腦自己和自己下

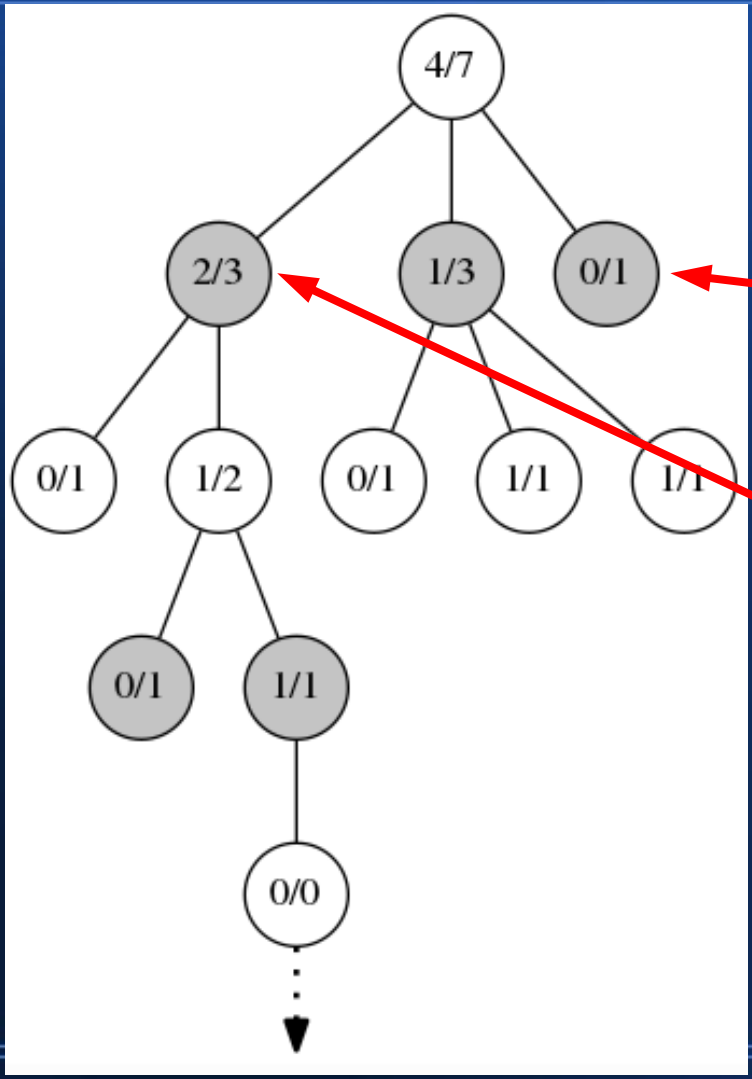
- 看起來好像很神奇，說穿了並沒有那麼困難！
- 但是、這種方法有甚麼弱點呢？
- 我認為，由於沒有進行全面探索，所以 MCTS 勢必會漏掉一些分支！
- 問題是、這些漏掉的分支是否真的不太影響《策略網路》的品質呢？



就我看來、肯定是有漏洞的

- 請容我解釋一下！

讓我們看看剛剛的 MCTS 搜尋過程的第三張圖



MCTS 所選擇的，是信賴區間上界 (UCB) 最高的節點，繼續往下探索，這可能會有兩個弱點：

1. 如果有個《第二層節點》，隨機探索的幾次結果都是輸掉，於是被《策略網路》認為不應該下，但如果底下那層的分數卻是翻盤的，那麼就會被漏掉。

2. 如果有個《第二層節點》，隨機探索的幾次結果都是相對不錯，於是被《策略網路》認為應該下，但如果底下那層的分數卻是翻盤的，那麼也會被漏掉。

另一個弱點：由於是自我對下，因此採用策略都會偏向雷同，所以比較不容易探索到差異太大的策略，因此我認為 AlphaGo 在這方面應該還是有弱點！

接著、讓我們看看《深度卷積神經網路》(DCNN)可能會有甚麼問題

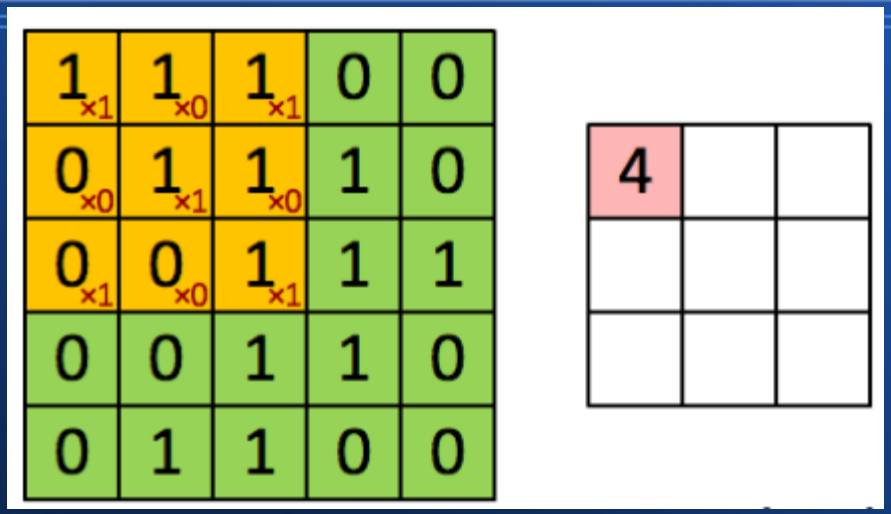
卷積神經網路(Convolutional Neural Network)

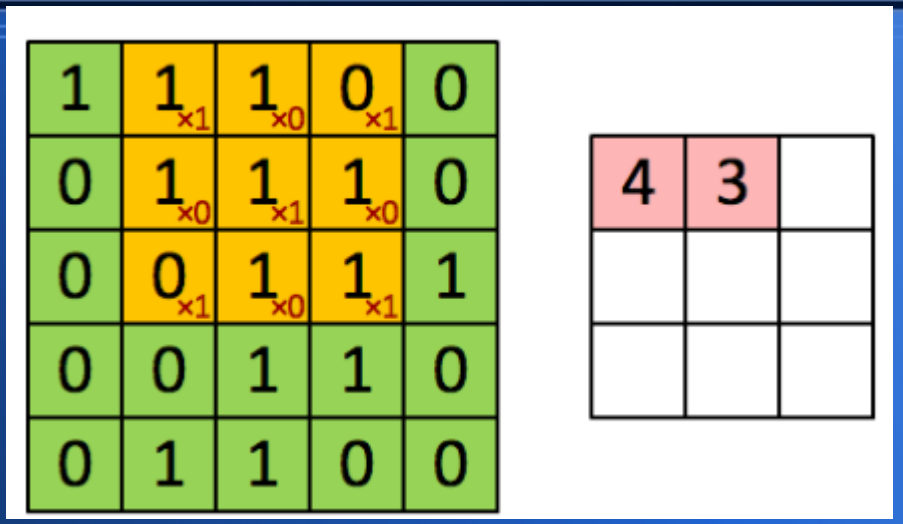
在圖像識別的問題上，我們處理的是一個二維的神經網路結構，以100*100像素的圖片來說，其實輸入資料就是這10000像素的向量(這還是指灰階圖片，如果是彩色則是30000)，那如果隱藏層的神經元與輸入層相當，我們等於要計算10的8次方的權重，這個數量想到就頭疼，即使是透過平行計算或者是分布式計算都恐怕很難達成。因此卷積神經網路提出了兩個很重要的觀點：

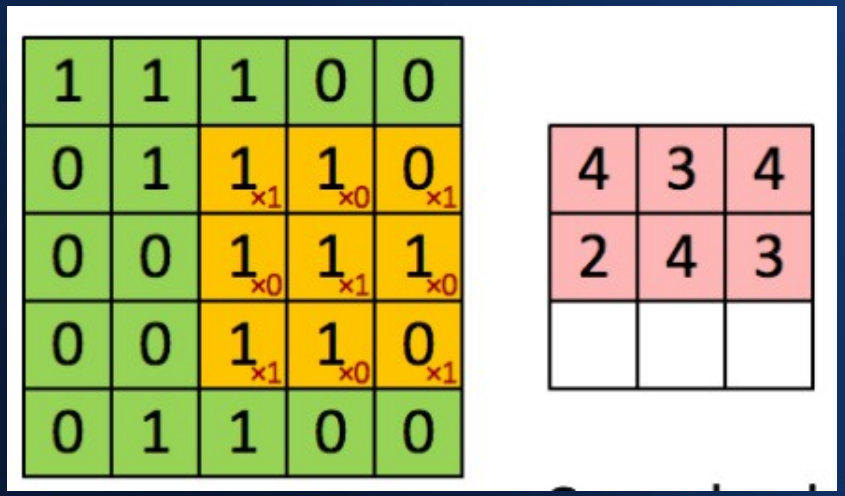
1. **局部感知域**：從人類的角度來看，當我們視覺聚焦在圖片的某個角落時，距離較遠的像素應該是不會影響到我們視覺的，因此局部感知域的概念就是，像素指需要與鄰近的像素產生連結，如此一來，我們要計算的神經連結數量就能夠大幅降低。舉例來說，一個神經元指需要與鄰近的10*10的像素發生連結，那麼我們的計算就可以從10的8次方降低至100*100*(10*10)=10的6次方了。
2. **權重共享**：但是10的6次方還是很多，所以這時要引入第二個觀念就是權重共享。因為人類的視覺並不會去認像素在圖片上的絕對位置，當圖片發生了平移或者是位置的變化，我們都還是可以理解這個圖片，這表示我從一個局部所訓練出來的權重(例如10*10的卷積核)應該是可以適用於照片的各個位置的。也就是說在這個10*10範圍所學習到的特徵可以變成一個篩選器，套用到整個圖片的範圍。而權重共享造成這10*10的卷積核內就共用了相同的權重。一個卷積核可以理解為一個特徵，所以神經網路中可以設計多個卷積核來提取更多的特徵。下圖是一個3*3的卷積核在5*5的照片中提取特徵的示意圖。

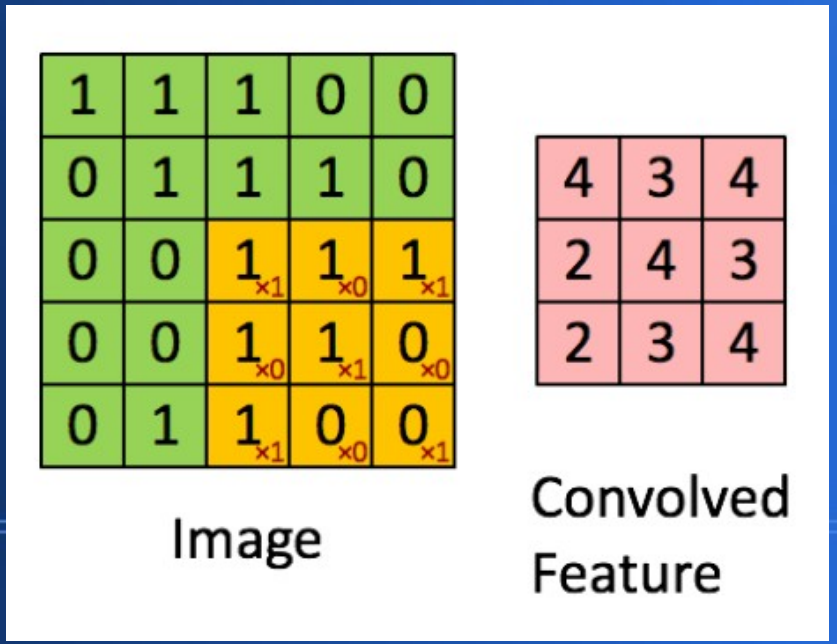
1. 接下來 DCNN 資料擷取自 尹相志 的 淺談Alpha Go所涉及的深度學習技術 這篇文章

卷積神經網路 CNN 是對《局部感知域》進行捲積
而且這些《捲積的權重會共享》

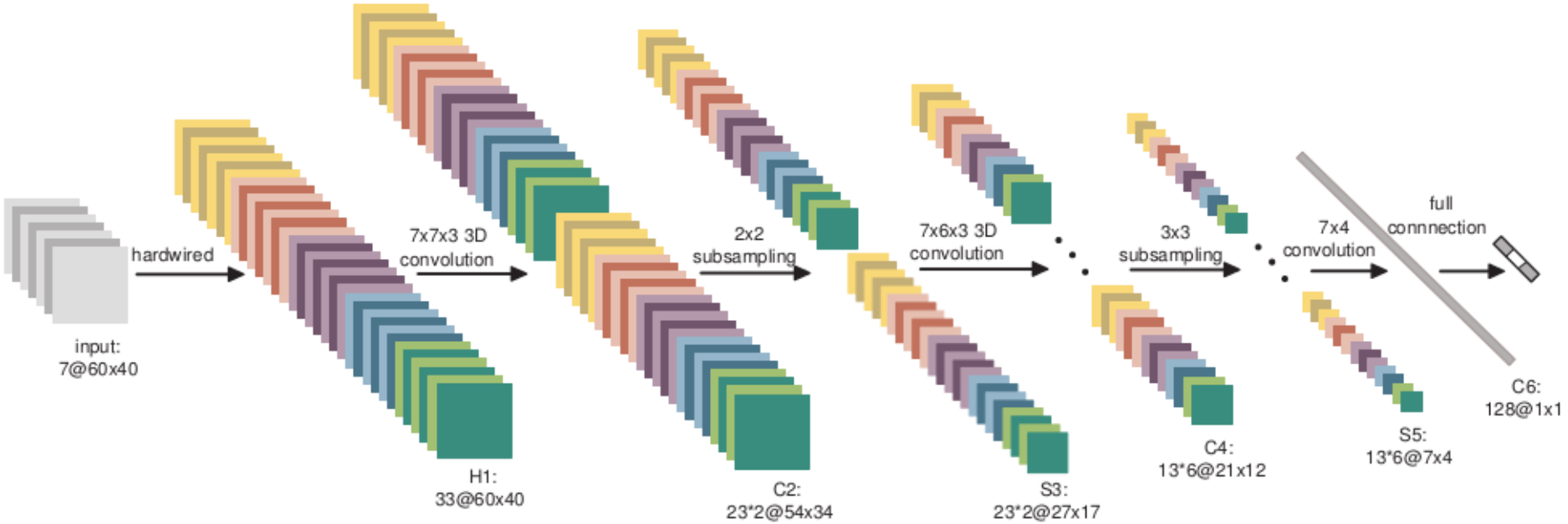




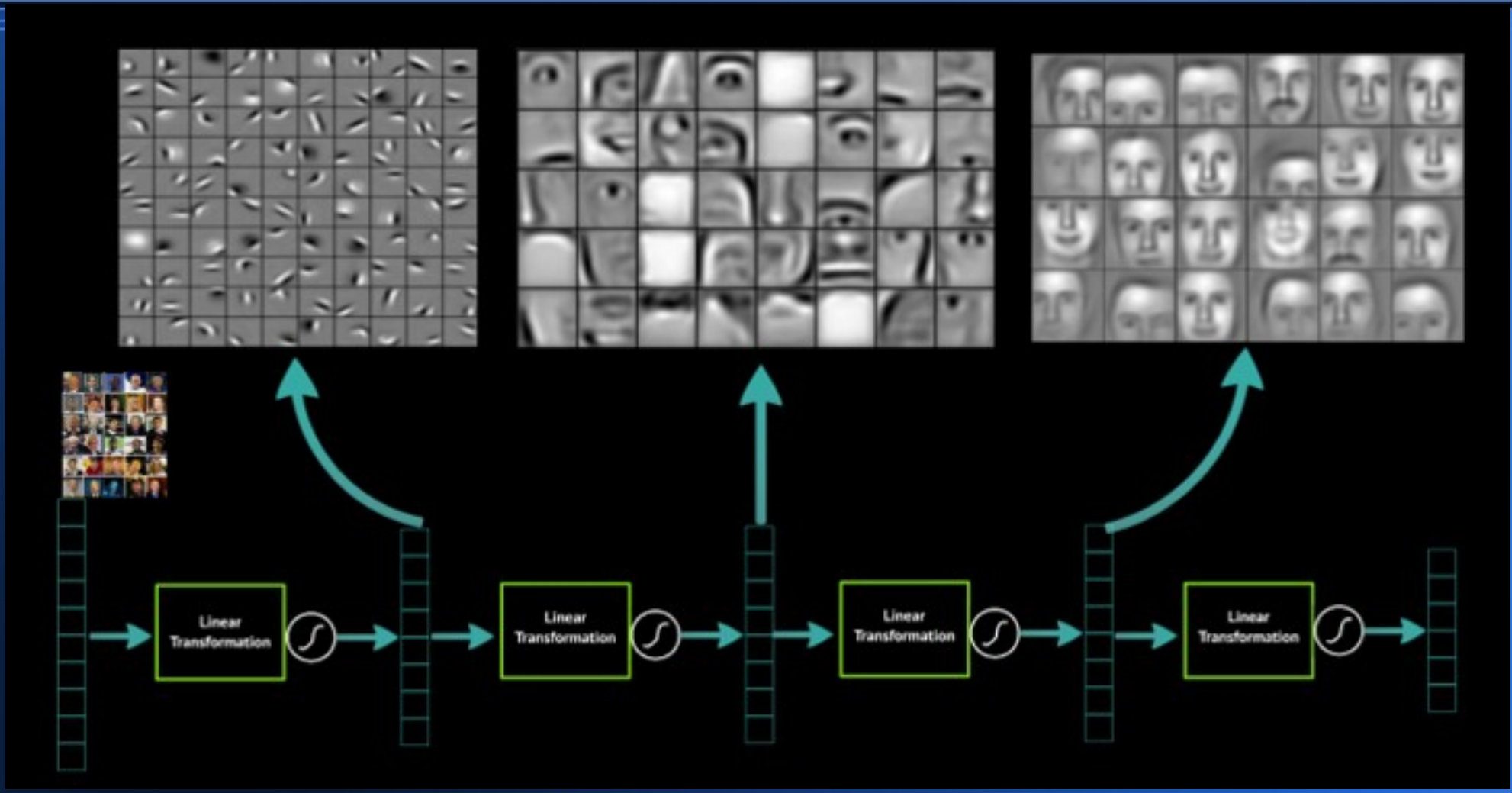




然後、為了同時照顧到局部和全局資訊
於是會用各種不同層次的捲積一起運作



最後透過捲積的《線性組合》
可以代表整體資訊



這個方法原本用在《影像處理》上面

- 這次被 AlphaGo 用在圍棋上面
- 以便同時能照顧到《整體與局部》

AlphaGo 的策略網路

- 先透過 5*5 的卷積核排除掉一些區域不去進行計算，以大區塊統計過濾掉一些不好的區域，這種機制卻能讓 AlphaGo 計算速度提升千倍以上。
- 然後再從沒被過濾掉的區域找出最好的下子點（或對方最可能的下子點）

這種捲積網路

- 看來可以平衡整體與區域
- 這樣是否就沒有缺陷呢？
- 或許、 5*5 的區塊大小，會是一個問題。
- 如果必須看 6*6 才能知道該區塊好，那是否會有個 5*5 的區塊因表現太差而被忽略，結果可能會導致致命性的問題呢？

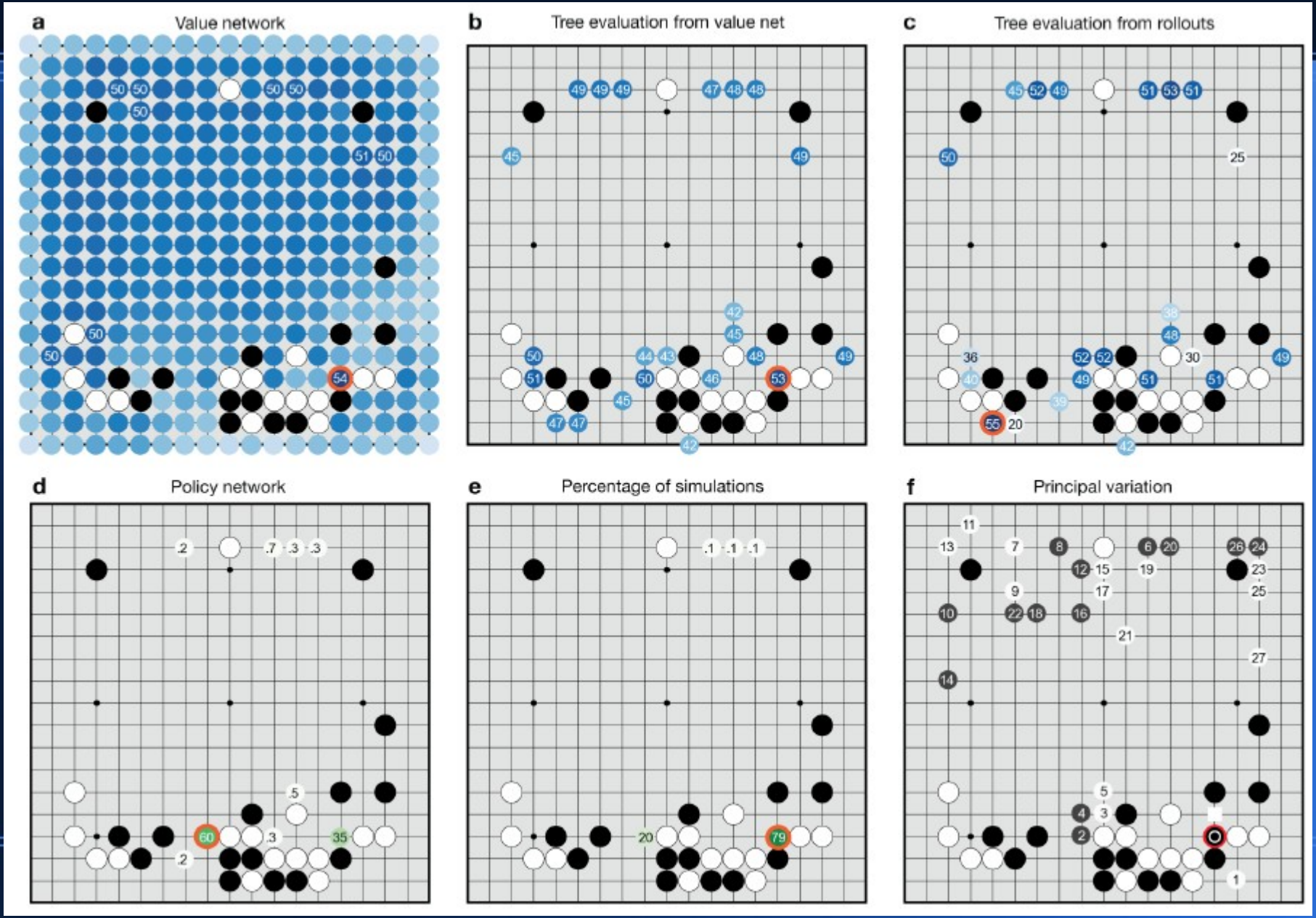
關於這些猜測

- 我不是圍棋高手
- 也沒辦法探測 AlphaGo
- 只能留給《李世石》去想了！

最後、AlphaGo 的論文中

- 提出了一個和樊麾對戰時的盤面範例，讓我們可以觀察 AlphaGo 是如何決策的！

該案例如下

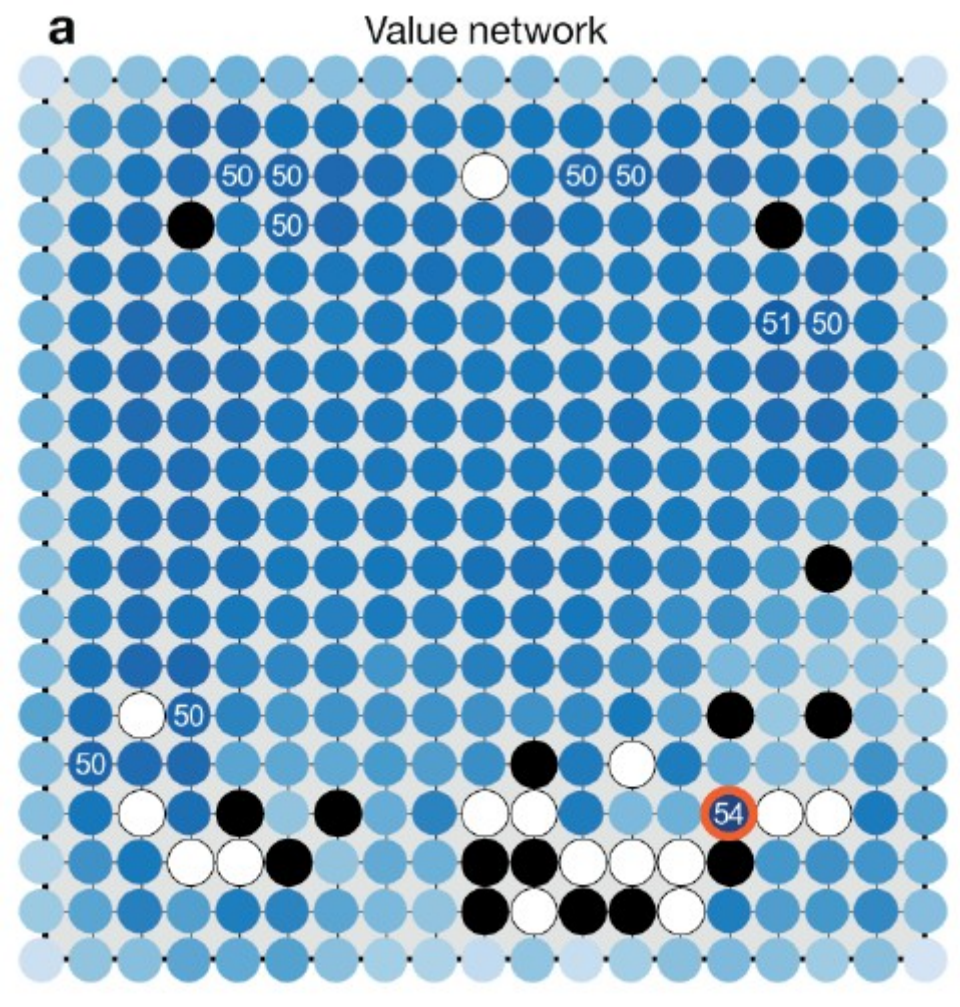

a
Value network
b
Tree evaluation from value net
c
Tree evaluation from rollouts
d
Policy network
e
Percentage of simulations
f
Principal variation

以下是該圖的說明文字

Figure 5: **How *AlphaGo* (black, to play) selected its move in an informal game against Fan Hui.** For each of the following statistics, the location of the maximum value is indicated by an orange circle. **a** Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b** Action-values $Q(s,a)$ for each edge (s,a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c** Action-values $Q(s,a)$, averaged over rollout evaluations only ($\lambda = 1$). **d** Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e** Percentage frequency with which actions were selected from the root during simulations. **f** The principal variation (path with maximum visit count) from *AlphaGo*'s search tree. The moves are presented in a numbered sequence. *AlphaGo* selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (1) predicted by *AlphaGo*.

讓我們一張一張仔細看清楚

首先是價值網路



AlphaGo 為黑子，樊麾下白子

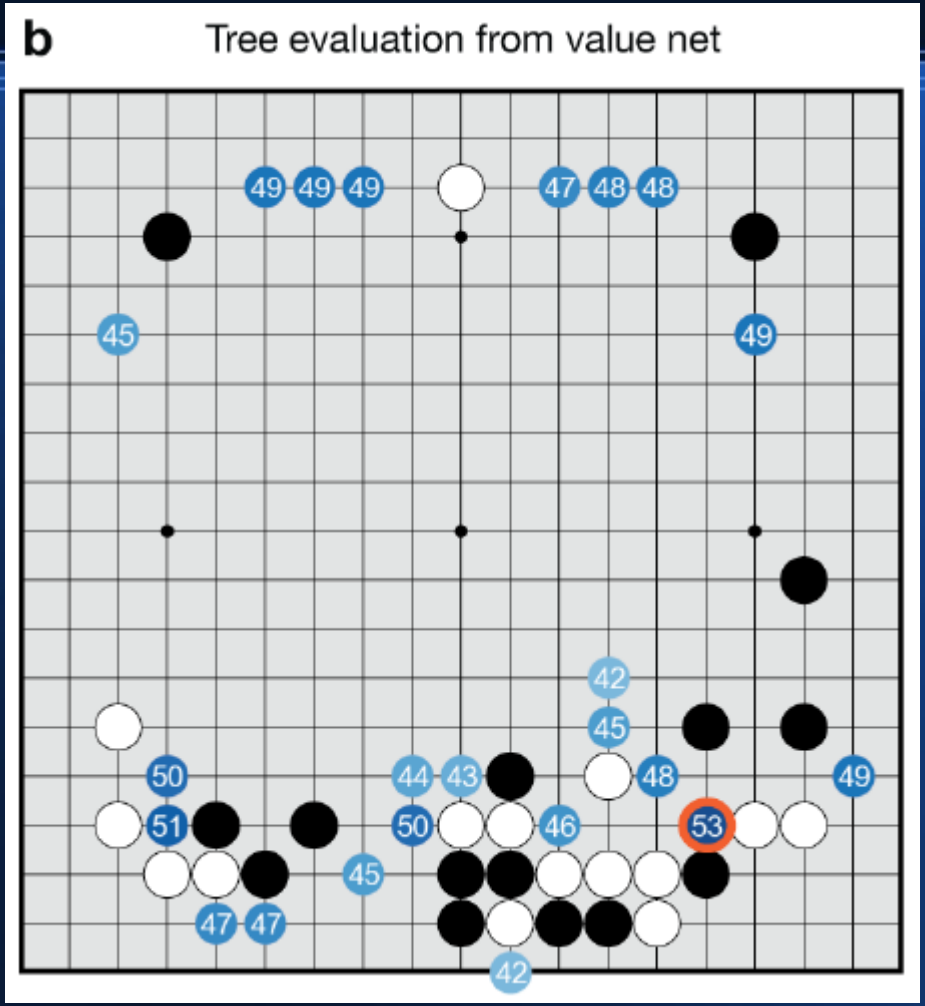
使用《價值網路》 Value Network
評估哪些點最值得下，然後就可以
進一步搜尋確認到底這些點是否真的
夠好或者沒問題！

其中的數字為下該子之後的獲勝比率。

價值網路也可以用來評估對手
可能會下的點。

orange circle. **a** Evaluation of all successors s' of the root position s, using the value network $v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b** Action-values $Q(s, a)$

接著對這些有價值的點進行對局樹搜尋與評估



AlphaGo 為黑子，樊麾下白子

本圖是只採用價值網路的結果

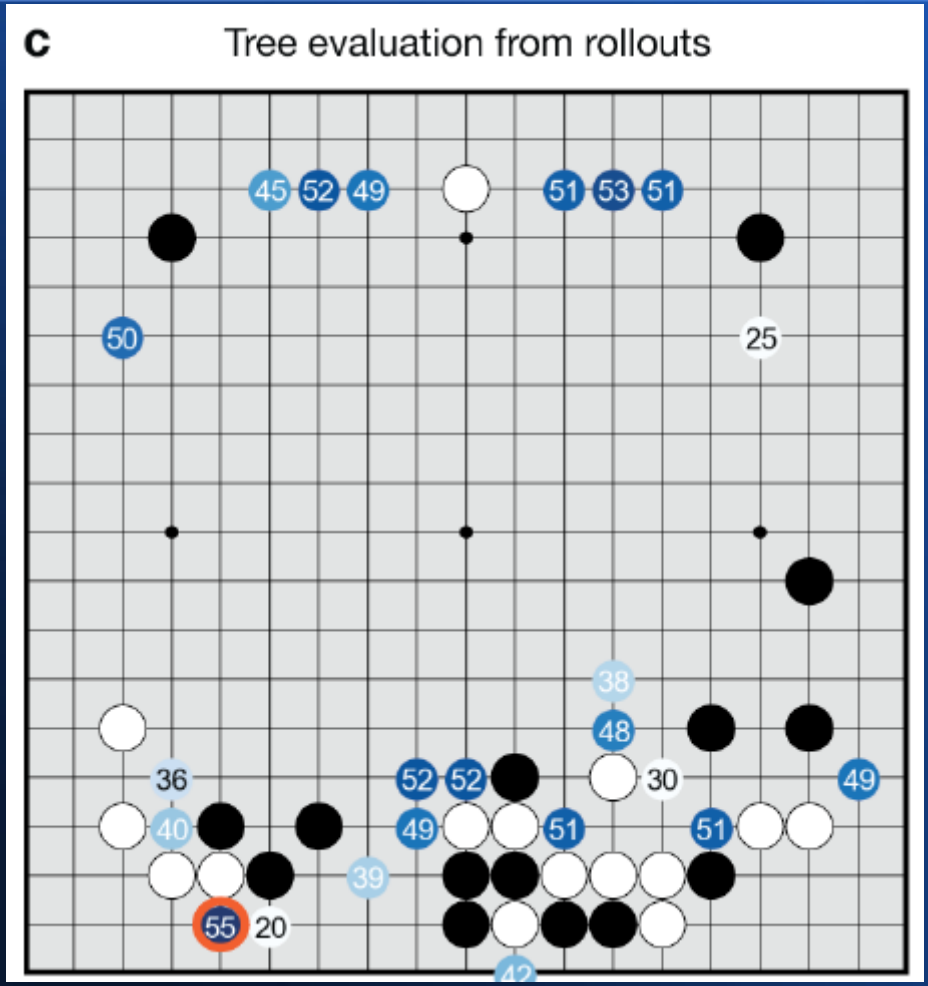
(把資料庫的權重設為 0 ，

也就是盡量不用棋譜的狀況)

此種情況下 53 分那格分數最高！

$v_\theta(s')$; estimated winning percentages are shown for the top evaluations. **b** Action-values $Q(s,a)$ for each edge (s,a) in the tree from root position s; averaged over value network evaluations only ($\lambda = 0$). **c** Action-values $Q(s,a)$, averaged over rollout evaluations only ($\lambda = 1$). **d** Move

圖 c 是單由《棋譜資料庫》預測所得到的評估結果



AlphaGo 為黑子，樊麾下白子

只採用棋譜資料庫的結果

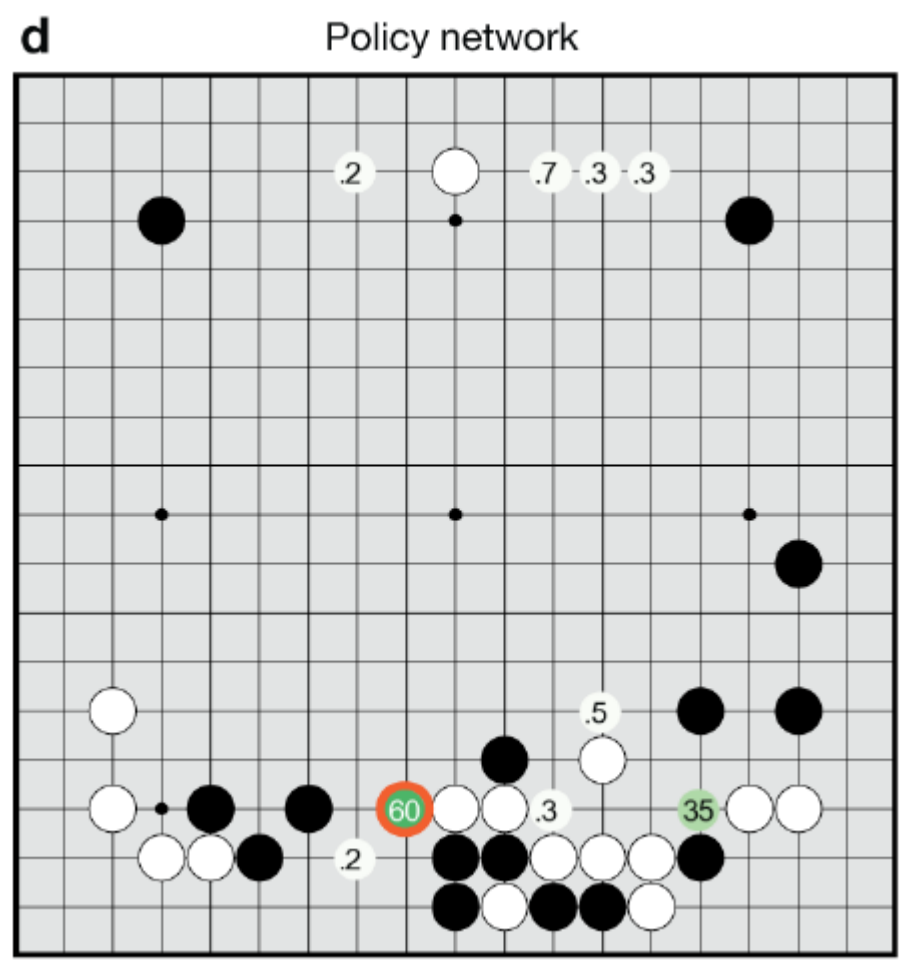
(把對戰訓練的權重設為 0 ，

也就是只用棋譜的狀況)

棋譜預測的 55 位置和前一個評估並不相同。

c Action-values $Q(s, a)$, averaged over rollout evaluations only ($\lambda = 1$).

圖 d 是策略網路的輸出
進行對局樹搜尋與評估



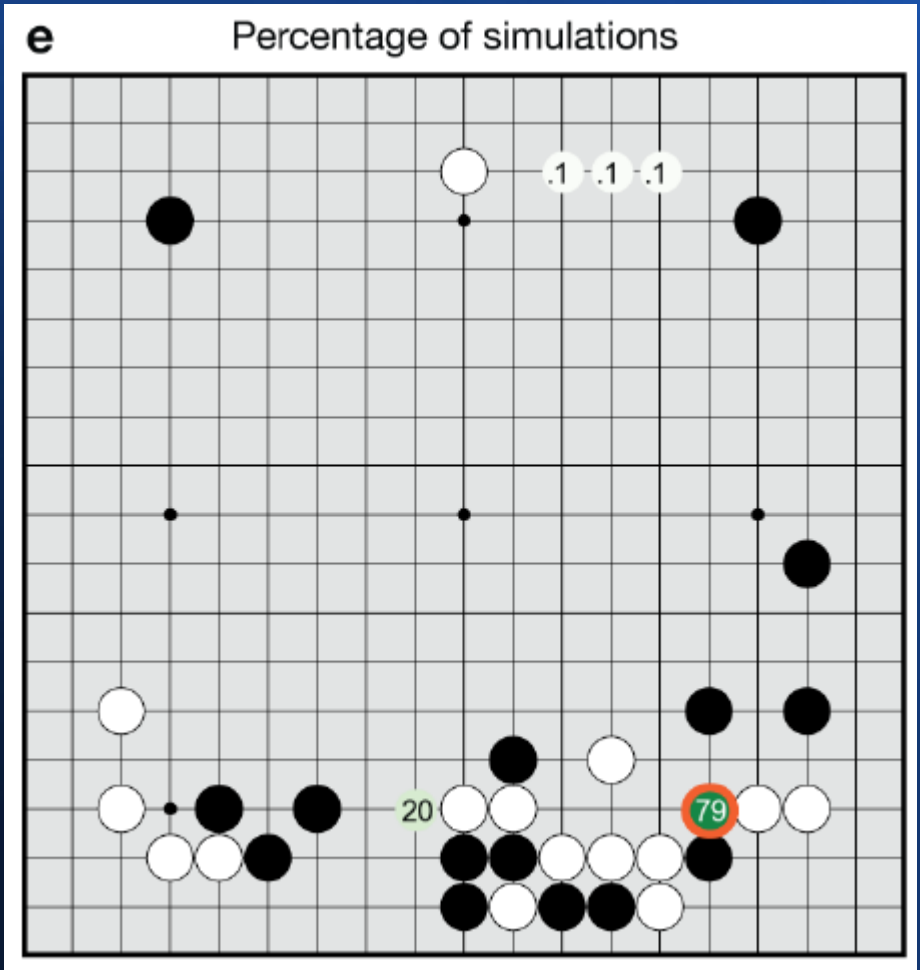
AlphaGo 為黑子，樊麾下白子

策略網路的評估，所得到的是如果 AlphaGo 當對手時，下在每一格的機率之百分比。

所有格點的機率加總為 1，圖中只列出機率比 0.01 大的點。

only ($\lambda = 0$). **c** Action-values $Q(s, a)$, averaged over rollout evaluations only ($\lambda = 1$). **d** Move probabilities directly from the SL policy network, $p_\sigma(a|s)$; reported as a percentage (if above 0.1%). **e** Percentage frequency with which actions were selected from the root during simulations.

圖 e 代表要進行搜尋模擬的比率



AlphaGo 為黑子，樊麾下白子

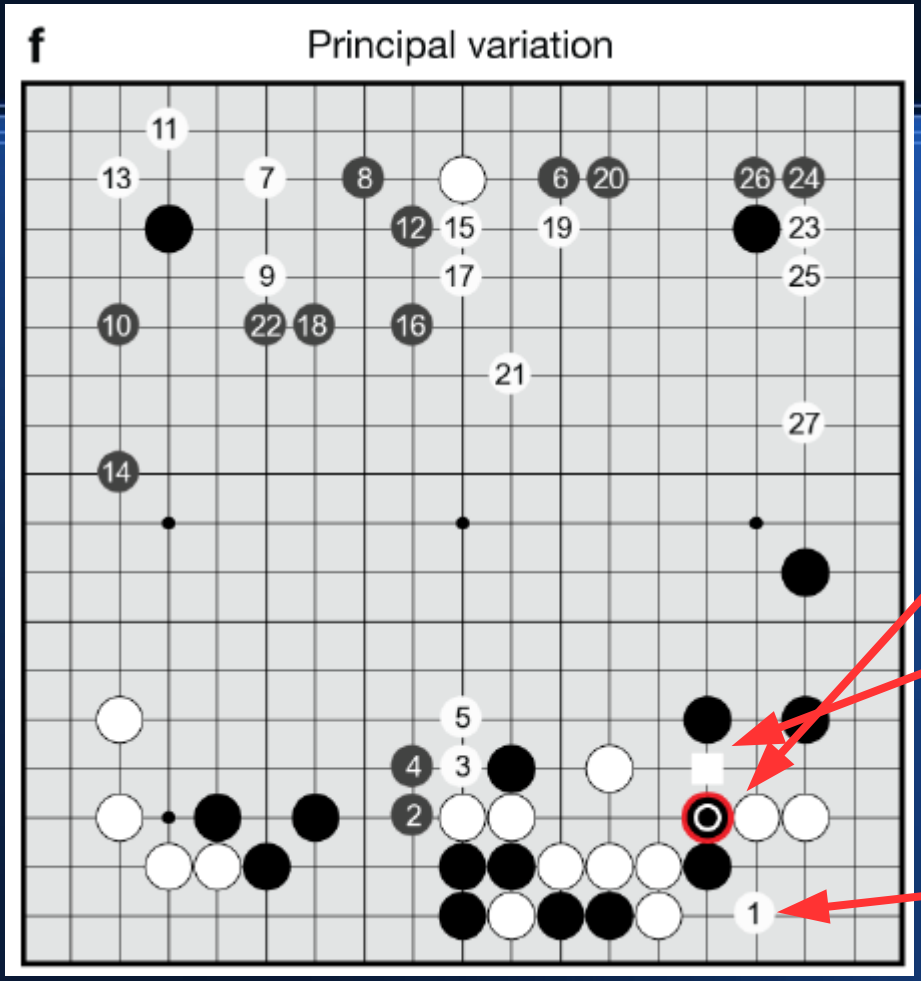
79 代表要把 79% 的模擬量

放在這裡 ??? 是這樣嗎 ???

其實我還沒完全看懂！

e Percentage frequency with which actions were selected from the root during simulations.

圖 f 是最終評估結果



AlphaGo 為黑子，樊麾下白子

AlphaGo 下黑子的點

樊麾接著下的點

但是 AlphaGo 認為

對手最應該下的點在此

f The principal variation (path with maximum visit count) from *AlphaGo*'s search tree. The moves are presented in a numbered sequence. *AlphaGo* selected the move indicated by the red circle; Fan Hui responded with the move indicated by the white square; in his post-game commentary he preferred the move (1) predicted by *AlphaGo*.

總而言之

- AlphaGo 是一個採用

  《棋譜資料庫 + 自我對下》

  兩種方法結合的圍棋系統

AlphaGo 可以選擇《棋譜和自動對下》所佔的權重比例

- 經測試發現混合參數取 0.5 的時候棋力最強！

We also assessed variants of *AlphaGo* that evaluated positions using just the value network ($\lambda = 0$) or just rollouts ($\lambda = 1$) (see Figure 4,b). Even without rollouts *AlphaGo* exceeded the performance of all other Go programs, demonstrating that value networks provide a viable alternative to Monte-Carlo evaluation in Go. However, the mixed evaluation ($\lambda = 0.5$) performed best, winning $\geq 95\%$ against other variants. This suggests that the two position evaluation mechanisms are complementary: the value network approximates the outcome of games played by the strong but impractically slow p_ρ, while the rollouts can precisely score and evaluate the outcome of games played by the weaker but faster rollout policy p_π. Figure 5 visualises *AlphaGo*'s evaluation of a real game position.

看到這裡

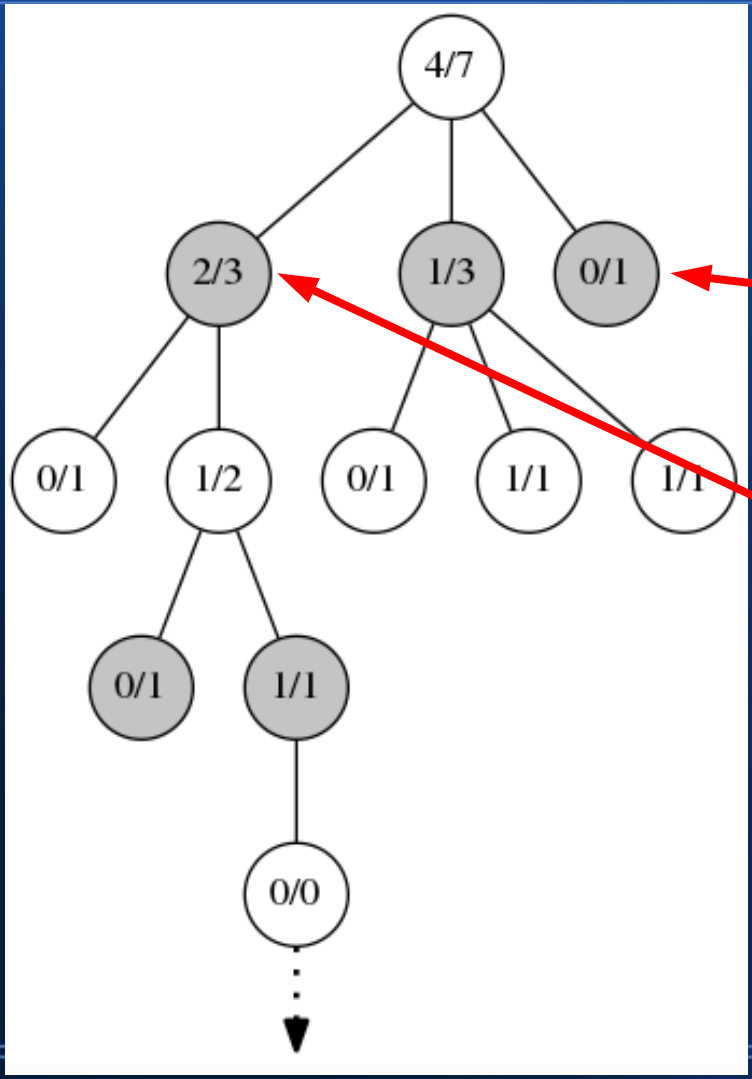
- 我們已經將 AlphaGo 所使用的方法看完了！

讓我們總結一下

- AlphaGo 是一個融合《棋譜資料庫》與《電腦自我對下》兩種策略的系統。
- 主要融合了《蒙地卡羅對局搜尋法 + 深度卷積神經網路》兩種方法。
- 《蒙地卡羅對局搜尋法》用來指導接下來要搜尋哪個區域，並用自我對下的方式進行訓練強化。
- 《深度卷積神經網路》則可以考量《區域 + 全局》，AlphaGo 採用 5*5 的區塊進行捲積。

我們認為 AlphaGo 的弱點可能會出在

《蒙地卡羅對局搜尋法》MCTS 可能會有以下弱點



MCTS 所選擇的，是信賴區間上界 (UCB) 最高的節點，繼續往下探索，這可能會有兩個弱點：

1. 如果有個《第二層節點》，隨機探索的幾次結果都是輸掉，於是被《策略網路》認為不應該下，但如果底下那層的分數卻是翻盤的，那麼就會被漏掉。

2. 如果有個《第二層節點》，隨機探索的幾次結果都是相對不錯，於是被《策略網路》認為應該下，但如果底下那層的分數卻是翻盤的，那麼也會被漏掉。

另一個弱點：由於是自我對下，因此採用策略都會偏向雷同，所以比較不容易探索到差異太大的策略，因此我認為 AlphaGo 在這方面應該還是有弱點！

《深度卷積神經網路》的弱點可能出現在

- 5*5 捲積矩陣的邊界，考慮以 6*6 的思惟造成 AlphaGo 的盲點

當然、這些僅只是

- 我研究 AlphaGo 論文所猜測的弱點

我沒有辦法確認

- AlphaGo 在這些地方是否真的有弱點

不過、或許可以提供參考！

就算

- AlphaGo 沒有這些弱點

那也沒有關係！

因為透過這次的研究

- 我已經把 AlphaGo 的論文看完了！

而且

- 我還把
  - 蒙地卡羅對局搜尋
  - 還有捲積神經網路
- 以及
  - 電腦下棋的最新進展

充分的吸收消化了一遍！

我想

- 這就是 AlphaGo 和李世石這場對戰
- 帶給我的最大收穫了！

感謝您

- 收看這次的十分鐘系列！

我們下次見！

Bye bye!