

國 立 虎 尾 科 技 大 學

機 械 設 計 工 程 系

cd2023 2a-pj1ag2 分組報告

網際足球機器人

Web-based Football Scene Design

指導教授： 嚴 家 銘 老 師

班 級： 四 設 二 甲

學 生： 第 一 位 (41023146)

第 二 位 (41023148)

中華民國

112 年 3 月

國立虎尾科技大學 機械設計工程系
分組報告製作合格認可證明

分組報告製作修習學生： 四設二甲 41023146 第一位
四設二甲 41023148 第二位

分組報告題目：網際手足球場景設計

經評量合格，特此證明

評審委員： _____

指導老師： _____

中華民國一一年三月三十一日

摘要

本課程將在設計簡單的移動機器人 BubbleRob 的同時，介紹相當多的 CoppeliaSim 功能。本教程相關的 CoppeliaSim 場景文件位於 scenes/tutorials/BubbleRob。

此專題是運用足球機器人，將其導入 CoppeliaSim 模擬環境並給予對應設置，將其機電系統簡化並運用 AI 進行訓練，找到適合此系統的演算法後，再到 CoppeliaSim 模擬環境中進行測試演算法在實際運用上的可行性。並嘗試透過架設伺服器將 CoppeliaSim 影像串流到網頁供使用者觀看或操控。

關鍵字：類神經網路、強化學習、caht gpt、CoppeliaSim、OpenAI Gym

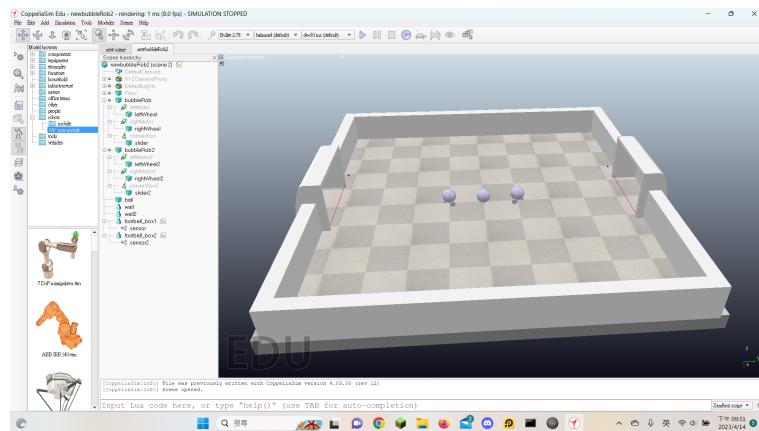
誌 謝

在此鄭重感謝製作以及協助本分組報告完成的所有人員，首先感謝學長範本，讓我們比較好改。

第一章 前言

1.1 研究動機

機器學習與各領域結合的應用越來越廣泛，在機電系統採用強化學習是為了讓機電系統的控制達到最佳化。本專題以實體的足球機之機電系統作為訓練模型，將實體機器轉移到虛擬環境進行模擬，為了找到適合的訓練參數，因此將模型簡化後再進行測試各種參數的優劣，透過不斷的訓練來得到一個優化過的對打系統，以下是成品圖。



1.2 研究目的與方法

本研究分三大部分，第一運用 OpenAI Gym 裡內建編譯的 ATARI 2600 遊戲 Pong-v0，來作為訓練環境，加上強化學習的理論，測試不同演算法參數以訓練出最佳化的對打系統，第二將整個簡化後冰球機導入 CoppilaSim 模擬環境並嘗試進行虛擬訓練，成為優化的對打機電系統。第三則是嘗試透過架設伺服器與虛擬環境結合。

透過簡化實體冰球機並導入虛擬環境，進行虛擬訓練，使用 Gym 的 Pong 當作對應的 2D 虛擬訓練環境，測試算法和訓練效果，篩選適合的算法與參數。

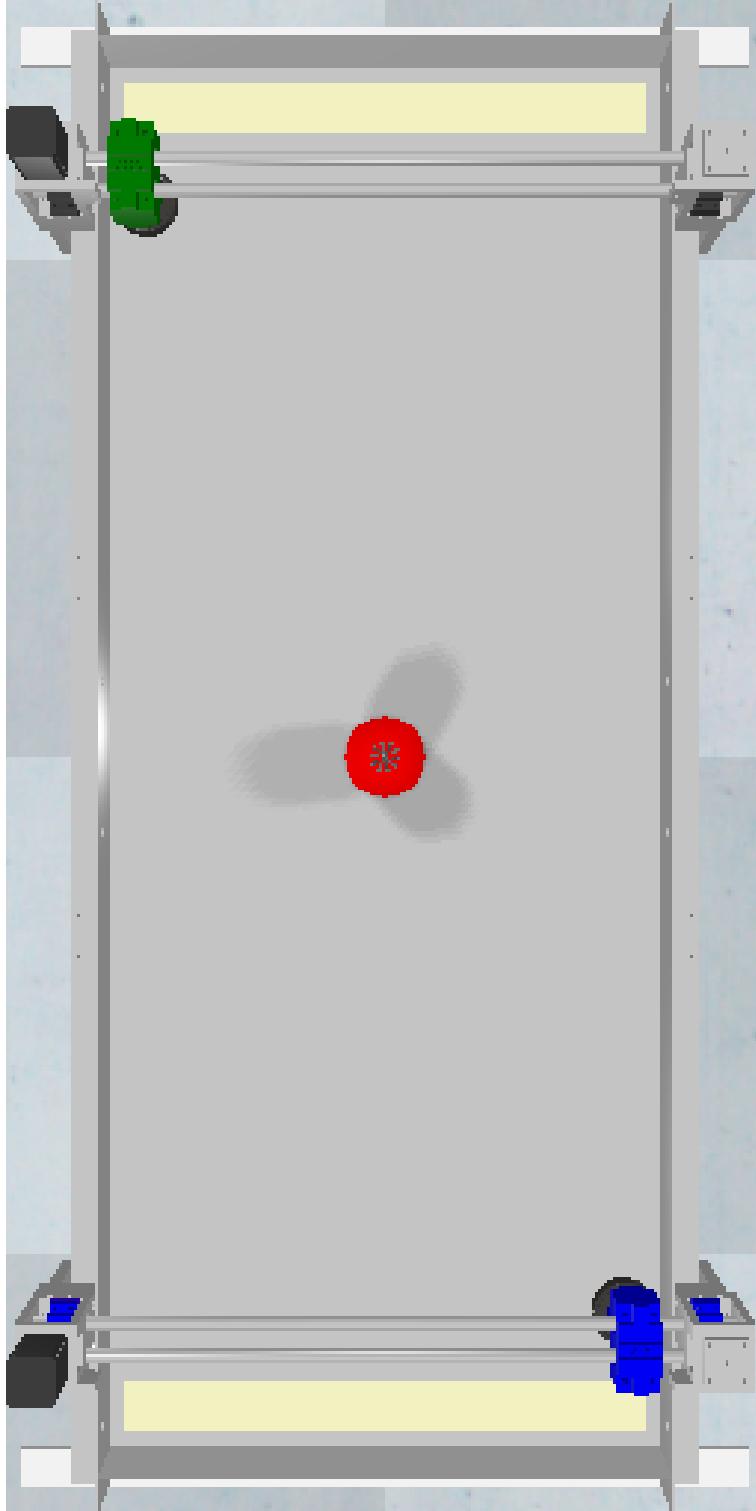


圖. 1.1: 虛擬環境簡化後的冰球機

建置 CoppilaSim 模擬環境，嘗試將 2D 訓練概念套用到 3D 環境進行測試，加入電腦視覺與 RemoteAPI，電腦視覺抓取球與擊錘的位置，

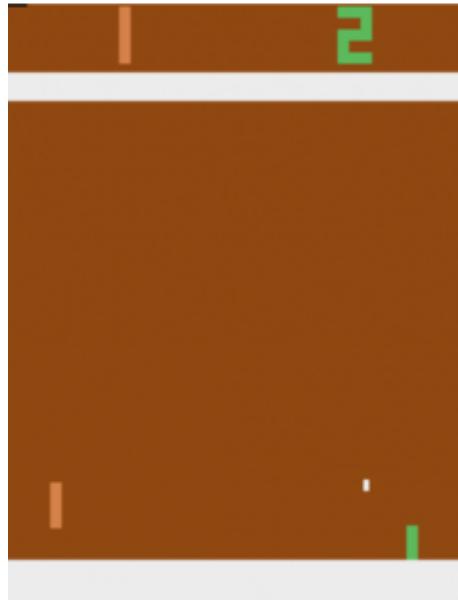


圖. 1.2: Gym 的 Pong game

透過 RemoteAPI 進行遠端控制，在 3D 環境測試算法可確保後續套用到實體機器上的可行性。

再透過架設伺服器與虛擬環境結合：讓虛擬環境的影像透過網伺服器串流影像供使用者遠端進行操控虛擬環境的擊錘進行打球，或是提供多人進行觀看對打影像。

1.3 未來展望

此專題希望能利用現有完成的機械學習的算法，能發展成虛擬訓練，再將訓練完的機器學習應用到虛擬環境或是實體機電系統，並透過伺服器將影像串流提供玩家網頁介面進行遠端操控，同時提供多人觀看及時的比賽影像，將整個冰球機的控制和使用者間有更完善串聯，機電系統的部分達到最優化控制和虛實整合的應用。

1.4 規則說明

Pong game 的遊戲規則簡單，透過擊錘將球打入對方球門即得一分，只要其中一方得 21 分就結束該局。擊錘只能沿單方向來回移動來進行防

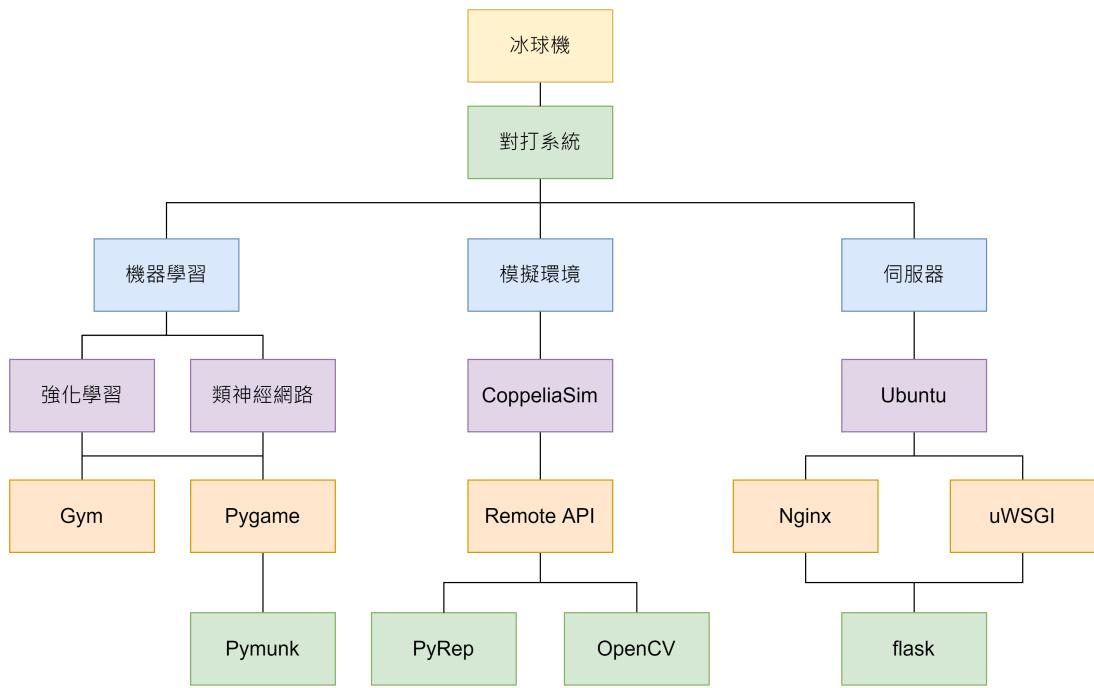


圖. 1.3: 研究架構

守和進攻。

遊戲規則如下：

1. 球打入敵方即得一分。
2. 擊錘只單一方向移動。
3. 最快贏得 21 分者獲勝，並結束該局遊戲。

第二章 機器學習

2.1 類神經網絡

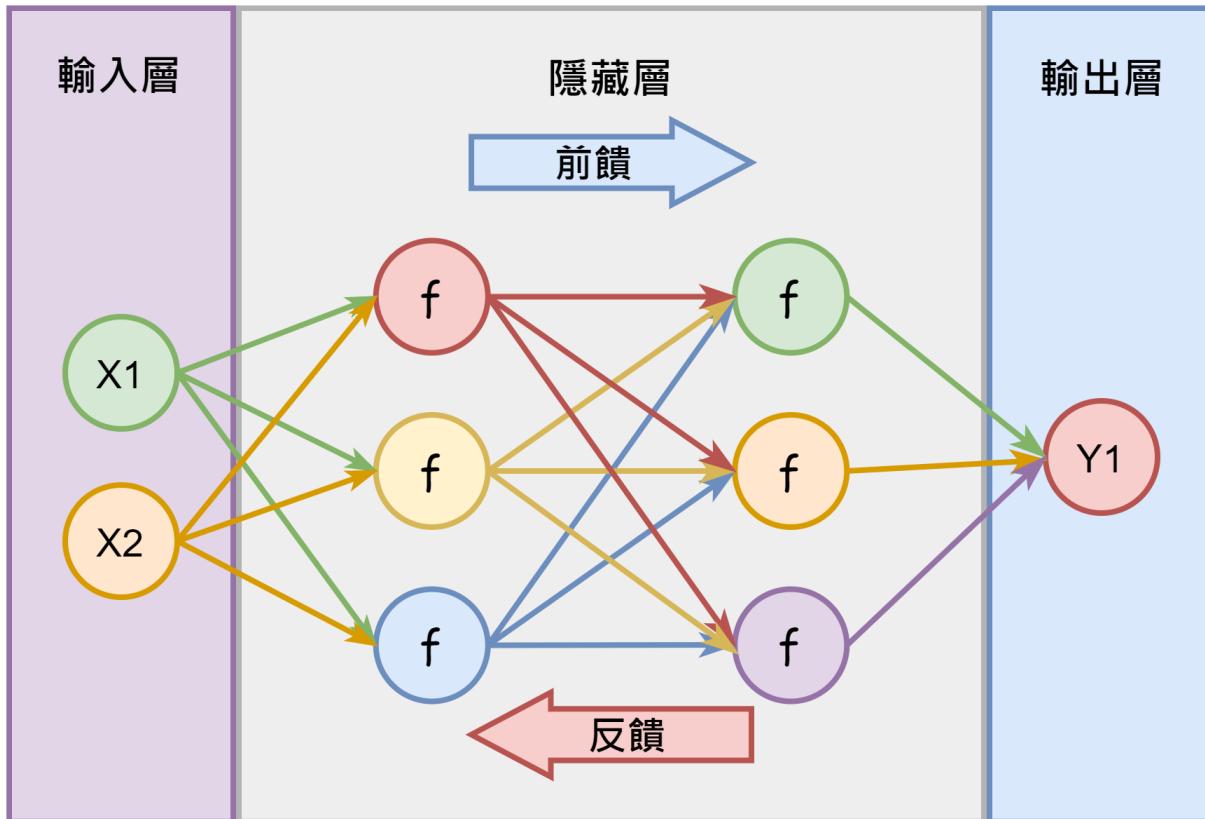


圖. 2.1: 類神經網路架構

典型的類神經網路架構，如(圖.2.1)所示每一層的每一個神經元都會連接到下一層全部的神經元，對於每個神經元輸出會有不同的權重。

神經元是AI系統中使用的數學模型，其行為與實際的大腦神經元運作方式相仿，模型以數字的方式表達，神經元間傳遞會有不同強度，而以數值的大小代表不同強度，這個數值我們稱為權重，對結果產生重要的影響。

(圖.2.1)基礎的類神經網路架構主要由輸入層、隱藏層和輸出層這三部分組成，實際運用上還有更多樣更複雜的類神經網路架構，深度學

習則是有更多的隱藏層，從意義上來說就是增加了類神經網路的深度。

此外，如(圖.2.1)所示，資料由輸入層傳入，經過隱藏層運算和記憶，再由輸出層進行輸出，這種資料被傳遞的方式被稱為前饋輸入(feed-forward)。

類神經網路架構有了記憶，就能進一步讓網路學習。當類神經網路接收資料並猜測答案，如果答案與實際答案不符、有落差或有錯誤的情況，它會回授並修改對每個神經元權重和偏差修正的程度，並嘗試調配各項數值來修正輸出的結果，讓結果的正確性提高，這樣的修正行為就被稱為反向傳播(back-propagation)。透過迭代方法進行反複試驗，模擬人們學習的行為，而每一次的迭代被稱為epoch，經過一定的迭帶次數後會透過反向傳播修正輸出的誤差，經過不斷執行的修正，最終類神經網路的學習會不斷進步並給出更好的答案，訓練時間長短取決於訓練項目的複雜程度。

可以看到該神經網絡的輸出僅取決於互連的權重，還取決於神經元本身的偏差，雖然權重會影響啟動函數曲線的陡度，但是偏差會將發生變化的整個曲線，向右或向左，權重和偏差的選擇，決定了單個神經元的預測強度，而訓練類神經網路使用的輸入數據可以來微調權重和偏差。(圖.2.2)

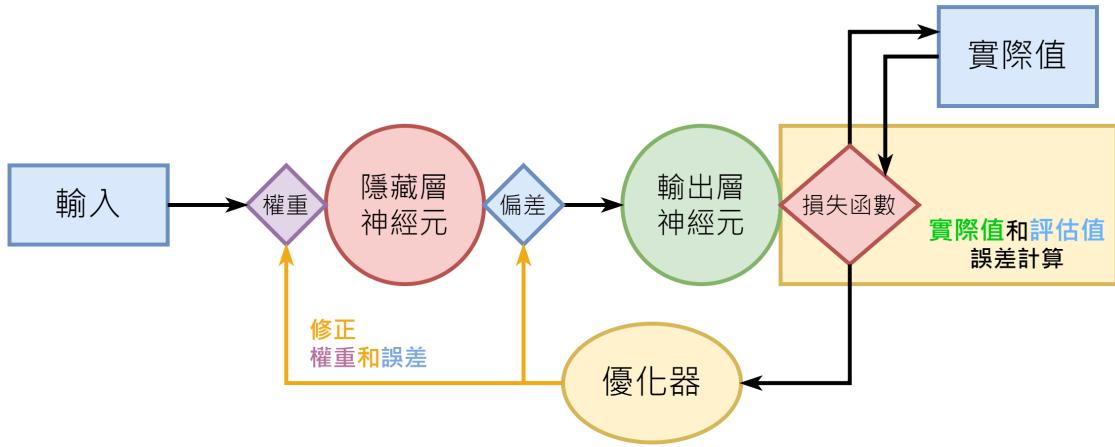


圖. 2.2: 類神經網路關係

2.1.1 啟動函數

啟動函數是設計類神經網路的關鍵部分，如果不使用啟動函數，神經元的計算只會有線性組合，這樣的類神經網路缺乏活性而且記憶性差；啟動函數能讓神經元計算呈現非線性，讓類神經網路因為計算的非線性而提高整個網路的活性和記憶性。

以下介紹幾種較為常見的啟動函數及其特性：

- Sigmoid Function(圖.2.3)：

輸出介於 0 到 1 之間，適用於二元分類，方程式具有非線性、可連續微分、且具有固定輸出範圍等特性，並可以讓類神經網路呈現非線性。

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

SigmoidPrime Function(圖.2.3，紅色的)是從 Sigmoid Function(圖.2.3，藍色)微分得來，以梯度運算的方式，可以減少梯度誤差，但也是造成梯度消失的主要原因，若要改善梯度消失需要搭配優化器使用，方程式如下：

$$\sigma'(x) = \sigma(x)[1 - \sigma(x)]$$

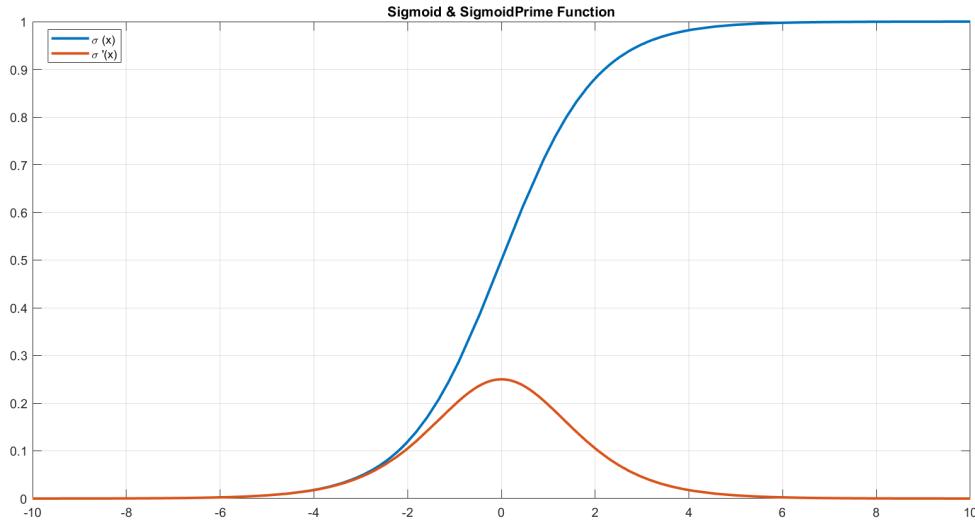


圖. 2.3: SigmoidFunction

- Softmax :

Softmax 會計算每個事件分布的機率，適用多項目分類、其機率總合為 1。以此專案為例，假設擊錘移動有向上移動、向下移動及不移動這三個決策選項，則這三個決策機率值總和為 1。

$$S(x) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_i}}$$

- ReLU Function :

ReLU Function 方程式特性：若輸入值為負值，輸出值為 0；若輸入值為正值，輸出則維持該輸入數值。ReLU 計算方式簡單、收斂速度快，這是類神經網路最普遍拿來使用的啟動函數，因為可以解決梯度消散的問題，但須注意：起始值若設定到不易被激活範圍或是權重過渡所導致權重梯度為 0 就會造成神經元難以被激活。

$$f(x) = \max(0, x)$$

$$if, x < 0, f(x) = 0$$

$$else f(x) = x$$

2.1.2 損失函數

損失函數是類神經網路的另一個重要的部分，損失函數會將類神經網絡的結果與期望結果進行比較，且必須重複估算模型當前狀態的誤差；該函數可用於估計模型的損失，以便可以更新權重減少下次評估時的損失，下一節會有詳細說明，以下簡述幾種優化的方法：

- Gradient Descent

利用梯度的方式尋找最小值的位置，其特色可找到凸面 error surface 的絕對最小值，在非凸面 error surface 上找到相對最小值。其缺點是在非凸面 error surface 要避免被困在次優的局部最小值。

- Batch gradient descent

用批次的方式計算訓練資料，整個資料集計算梯度只更新一次，因此計算和更新時會占用大量記憶體。整體效率較差、速度較緩慢。由 Gradient Descent 延伸出來的算法。其收斂行為與 Gradient Descent 相同。

- Stochastic gradient descent

每次執行時會更新並消除誤差，有頻繁更新和變化大的特性，較不容易困在特定區域。由 Gradient Descent 延伸出來的算法。其收斂行為與 Gradient Descent 相同。

- Mini-batch gradient descent

結合 Batch gradient descent 和 Stochastic gradient descent 的特點：批量計算和頻繁更新，所延伸的算法。利用小批量的方式頻繁更新，並使收斂更穩定。其缺點：學習率挑選不易、預定義 threshold 無法適應數據集的特徵、對很少發生的特徵無法執行較大的更新、非凸面 error surface 要避免被困在次優的局部最小值等。

- Gradient descent optimization algorithms

為了改善前面幾種算法而發展出來的優化算法。以下將列出數種優化算法。

- Momentum

在梯度下降法加上動量的概念，會加速收斂到最小值並減少震盪。

- Nesterov accelerated gradient

NAG，有感知能力的 Momentum：在坡度變陡時減速，避免衝過最
小值所造成的震盪(為了修正到最小值，來回修正而產生的震盪)。

- Adagrad

其學習率能適應參數：頻繁出現的特徵用較低的學習率，不經常出
現的特徵則用較高的學習率，且無須手動調整學習率。其缺點是，
學習率會急遽下降，最後會無限小，這算法就不再獲得知識。

- Adadelta

為 Adagrad 的延伸，下降激進程度，學習率從更新規則中淘汰，不
需設定預設學習率。

- RMSprop

為了解決 Adagrad 學習率急劇下降的問題，學習率除以梯度平方的
RMS，解決學習率無限小的情形。

- Adam Function

結合了 Adagrad 和 RMSprop 的優勢，有論文表示，在訓練速度方面
有巨大性的提升，但在某些情況下，Adam 實際上會找到比隨機梯度
下降法更差的解決方法。以下是計算過程：

$$g_t = \delta_\theta f(\theta)$$

一次矩指數移動均線：

$$m_t = \beta(m_{t-1}) + (1 - \beta_1)(\nabla w_t)$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

二次矩指數移動均線:

$$v_t = \beta_2(v_t - 1) + (1 - \beta_2)(\nabla w_t)^2$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

因此, Adam Function:

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{\hat{v}_t - \epsilon}} \hat{m}_t$$

- AdaMax

與 Adam 相似，依靠 (u_t) 最大運算。

- Nadam

結合 Adam 和 NAG，應用先前參數執行兩次更新，一次更新參數一次更新梯度。

- AMSGrad

改善 Adam 算法所導致收斂較差的情況(用指數平均會減少其影響)，換用梯度平方最大值來做計算，並移除去偏差的步驟。是否有比 Adam 算法好仍有待觀察。

- Gradient noise

有助於訓練特別深且復雜的網絡，noise 可改善不良初始化的網路。

- Mean Squared Error

他能告訴你一組點與回歸線接近的程度，透過獲取點與回歸線之距離(這些距離就是誤差)並對它們進行平方來做到這點，而平方是為了消除所有負號，也能讓更大的差異賦予更大的權重。

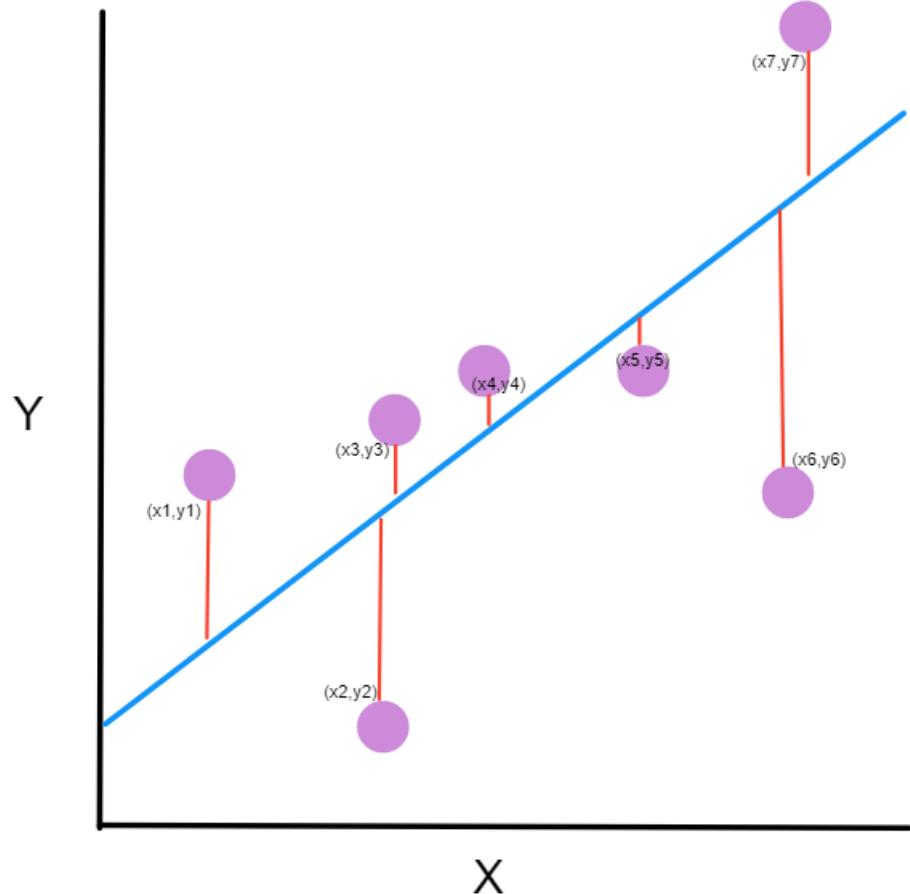


圖. 2.4: 線性回歸

回歸線: 數據點間最小距離的一條線。

n : 數據點的數量

y_i : 觀測值

\hat{y}_i : 預測值

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2$$

2.1.3 優化算法

- Gradient Descent Optimizer[11]

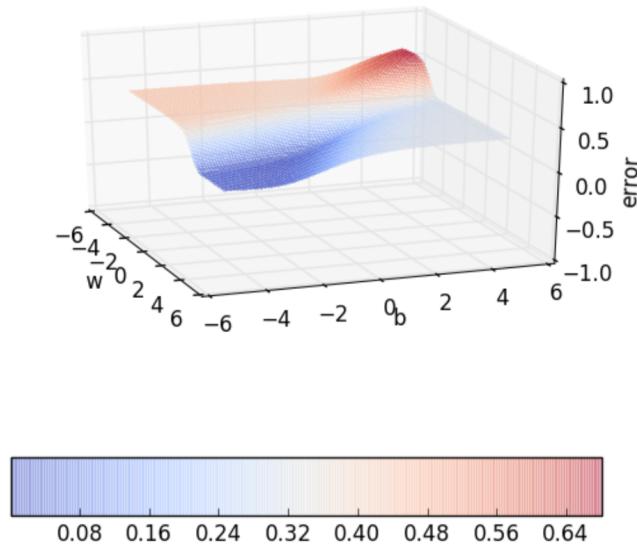


圖. 2.5: Error surface [11]

藉由梯度下降將目標函數值最小化，目標函數以 loss function $L(\theta)$ 為例， θ 為 weight(w) 和 bias(b) 的向量函數，為了找到(圖.2.5)上的最小值，因此加上 $\Delta\theta$ 將 θ 的方向修正並引導到正確方向，避免每次修正的過多導致錯過最小值，利用係數 η (學習率)縮放 $\Delta\theta$ 的修正量(圖.2.6)，修正後方程式為：

$$\theta = \theta + \eta \cdot \Delta\theta$$

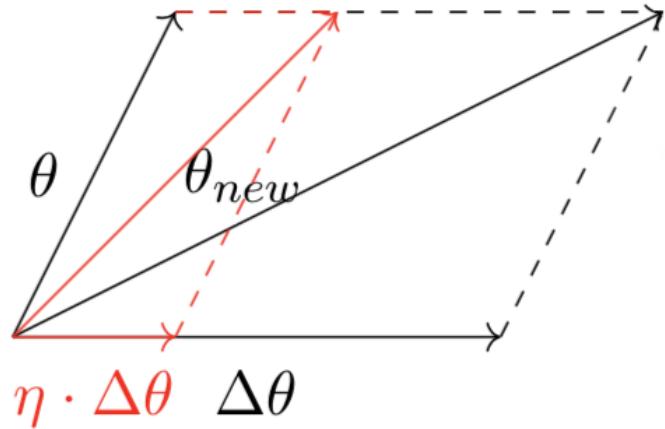


圖. 2.6: theta vector[11]

將 θ 以泰勒展開式表示，假設並 $\Delta\theta$ 為 u :

$$L_{(\theta+\eta u)} = L_{(\theta)} + \eta u^T \cdot \nabla_{\theta} L_{(\theta)} + \frac{\eta^2}{2!} u^T \cdot \nabla^2 L_{(\theta)} u + \frac{\eta^3}{3!} \dots + \frac{\eta^4}{4!} \dots + \frac{\eta^n}{n!} \dots$$

以泰勒展開式的型式表示的好處是： θ 些微的更動產生新值。 η 值通常小於一，當 $\eta^2 \ll 1$ ，因此可以忽略高階項

$$L_{(\theta+\eta u)} = L_{(\theta)} + \eta u^T \cdot \nabla_{\theta} L_{(\theta)} [\eta \text{ is typically small, so } \eta^2, \eta^3, \dots \rightarrow 0]$$

新的 $L(\theta + \eta u)$ 輸出的值會小於 $L(\theta) - L(\theta + \eta u) < 0$ ，同理可證 $u^T \cdot \nabla_{\theta} L(\theta)$ ，符合 u 這條件：當新的值小於舊的值， u 就是一個好的值。假設 u 和 $\nabla_{\theta} L(\theta)$ 的夾角為 β

$$\cos(\beta) = \frac{u^T \cdot \nabla_{\theta} L_{(\theta)}}{|u^T| |\nabla_{\theta} L_{(\theta)}|}$$

因為 $\cos(\theta)$ 的值介於 1 和 -1 之間

$$\begin{aligned} -1 < \cos(\beta) &= \frac{u^T \cdot \nabla_{\theta} L_{(\theta)}}{|u^T| |\nabla_{\theta} L_{(\theta)}|} \leq 1 \\ k &= |u^T| |\nabla_{\theta} L_{(\theta)}| \\ -k &\leq k \cos(\beta) = u^T \cdot \nabla_{\theta} L_{(\theta)} \leq k \end{aligned}$$

所以盡可能的讓新值小於舊值 ($L(\theta + \eta u) - L(\theta) < 0$)，loss 值就會減少得越多。因此 $uT \cdot \nabla \theta L(\theta)$ 應該為負，在這情況下 $\cos(\beta)$ 於 -1 ， β 的角度為 180° 這就是 θ 移動的方向與梯度方向相反的原因。梯度下降法告訴我們：當 θ 在特定值，並想減少新的 θ 值，使 loss 值逐漸減少就應該與梯度相反的方向找(若梯度為正值，找最小值就需往負的方向找)：

$$\begin{aligned} w_{t+1} &= w_t - \eta \nabla w_t \\ b_{t+1} &= b_t - \eta \nabla b_t \\ \text{where at } w &= w_t, b = b_t \\ \left\{ \nabla w_t = \frac{\partial L_{(\theta)}}{\partial w} \right. &\quad \left. \nabla b_t = \frac{\partial L_{(\theta)}}{\partial b} \right. \end{aligned}$$

- Batch gradient descrnd [12]

Vanilla gradient descent 又稱 Batch gradient descent(批次梯度下降法)，計算目標函數的梯度，參數 θ 對於整個訓練資料：

$$\theta = \theta - \eta \cdot \nabla \theta L_{(\theta)}$$

目標函數以為例 loss function $L(\theta)$ ，參數 θ 為 weight(w) 和 bias(b) 的函數， η 為學習率。由於計算整個資料集計算梯度只更新一次，Batch gradient descent 可能非常慢並且對於資料集無法符合及記憶體來說棘手(一次需要儲存整個資料集的資料，當更新和計算時會占用大量記憶體)。

程式. 2.1: Batch gradient descrnd

```
for i in range(nb_epochs) :
    params_grad = evaluate_gradient (loss_function, data, params)
    params = params - learning_rate * params_grad
```

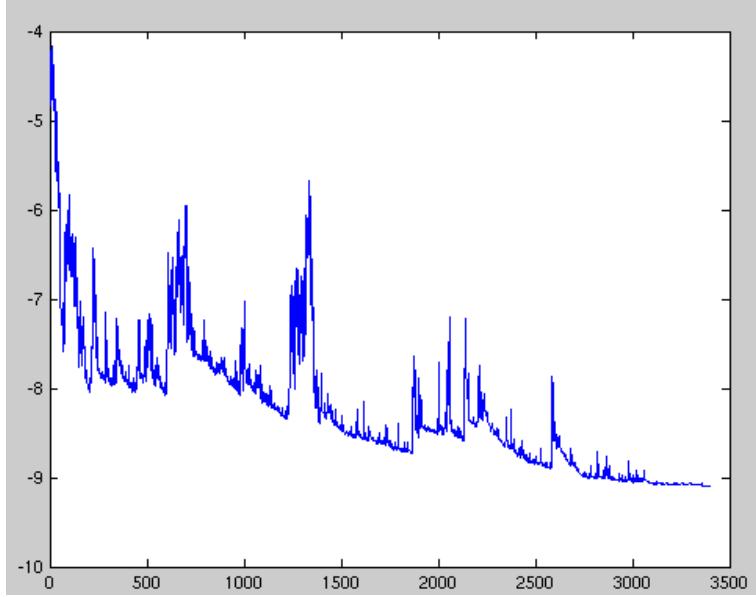


圖. 2.7: SGD fluctuation[12]

預定義每次 epoch，先計算 loss function 梯度向量對於整個資料集參數向量。如果梯度值來自於先前計算出的梯度值，就會檢查梯度，並以梯度相反的方向更新參數 θ ，學習率 η 決定多大的更新量。Batch gradient descent 對於凸面誤差可以保證收斂到廣域最小值，對於非面凸誤差可以收斂到局部最小值。

- Stochastic gradient descent(SGD)[12]

隨機梯度下降法，這裡的目標函數為 $J(\theta, x^i, y^i)$ (變數 θ 為 w(weight) 和 b(bias) 的函數，也可以寫成 $J(w, b, \theta, x^i, y^i)$)。

$$\theta = \theta - \eta \cdot \nabla_{\theta} J_{(\theta, x^i, y^i)}$$

批量梯度下降他會在每個參數更新前重新計算相似梯度。SGD 每次執行會更新來消除多餘(誤差)，因此通常速度很快。SGD 頻繁更新並變化很大，因為目標方程式波動很大(圖.2.7)。

SGD 的方程式一方面會跳到新的值和潛在局部最小值，另一方面 SGD 會持續超調(誤差超過預期)最後收斂到廣域最小值。無論如何

他被顯示當學習率下降緩慢，SGD 顯示與 Batch gradient descent 同樣收斂行為，幾乎可以肯定地，對於凸面或非凸面優化，會收斂到絕對或是局部最小值。這程式碼片段 [程式.2.2] 在訓練樣本上加入一個迴圈來對每個樣本評估梯度。每個 epoch(訓練循環) 會打亂訓練數據。

程式. 2.2: Stochastic gradient descrnd

```
for i in range(nb_epochs) :
    np.random.shuffle(data)
    for example in data :
        params_grad = evaluate_gradient(loss_function, example, params)
        params = params - learning_rate * params_grad
```

- Mini-batch gradient descent[12]

Mini-batch gradient descent(小批量梯度下降) 各取前兩者的優點，將資料集分割成小區塊，每個小區塊大小稱作 batch size，每次跑完 batch size 算迭代 (iteration) 一次，算完一次資料集即完成一次 epoch。舉例：資料集大小為 1000，若 batch size 為 50，iteration 為 datasets 的 batch size = $1000 \div 50 = 20$ ，當 iteration 跑完 20 次算完成一次 epoch。這方式可以減少參數更新的方差，並且可以穩定收斂；可利用深度學習庫所共有的高度優化的矩陣優化，從而由一個小批量計算出梯度非常有效。通常 batch sizes 的範圍介於 50~256，會因為應用而有所差異。訓練神經網絡時，通常選擇 Mini-batch gradient descent 算法，而當使用這算法時，通常也用 SGD 稱呼。

$$\theta = \theta - \eta \cdot \nabla_{\theta} J_{(\theta, x^{(i:i+n)}, y^{(i:i+n)})}$$

下面 [程式.2.3] 為迭代範例，batch size 大小為 50：

程式. 2.3: Mini-batch gradient descrnd

```
for i in range(nb_epochs) :
    np.random.shuffle(data)
    for batch in get_batches(data, batch_size = 50):
        params_grad = evaluate_gradient(loss_function, batch, params)
        params = params - learning_rate * params_grad
```

Mini-batch gradient descent 無論如何還是無法確保收斂的很好，存在一些需要解決的挑戰：

- 1 選擇適當的學習率是有難度的。如果學習率太小會導致收斂困難或緩慢，學習率太大則會阻礙收斂導致 loss function 來回波動或發生偏離。
- 2 學習率清單嘗試在訓練的時候調整學習率，即根據預定義清單或當目標下降於閾值 (threshold) 時降低學習率。但清單和閾值須預先定義，因此無法適應數據集的特徵。
- 3 另外相同學習率適用全部參數更新。如果資料稀疏而且外型有很特別的頻率，我們可能不希望將所有特徵更新到相同的程度，而是對很少發生的特徵執行較大的更新。
- 4 最小化神經網路常見的高度非凸面誤差方程式 (error function) 的另一關鍵挑戰則是要避免被困在大量次優的局部最小值區域中。認為困難實際上不是由局部最小值引起的，而是由鞍點引起的，即一維向上傾斜而另一維向下傾斜的點。這些鞍點通常被相同誤差的平穩段包圍，這使得 SGD 很難逃脫，因為在所有維度上梯度都接近於零。

- Gradient descent optimization algorithms[12]

SGD 難以在陡峭的往正確的方向，那就是說在一個維度上，曲面的彎曲比另一個維度要陡得多，這在局部最優情況下很常見。下圖 (圖.1) 的同心圓代表中心下凹的曲面。在這些情況下，SGD 會在陡峭的地方振盪，而僅沿著底部朝著局部最優方向猶豫前進，如 (圖.2.9) 所示。Momentum(動量) 是一個幫助加速 SGD 在正確方向和抑制震盪的方法，在 (圖.2.10)。

這麼做會增加一個係數 γ 來更新上次的向量到正確向量 (修正偏差)， γ 通常設為 0.9 左右。

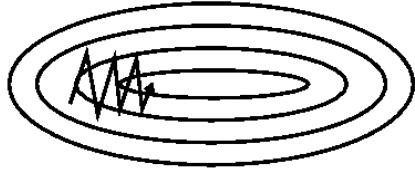


圖. 2.9: SGD without momentum[12]

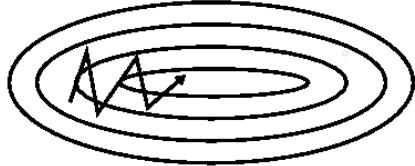


圖. 2.10: SGD with momentum[12]

$$v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J_{(\theta)} \quad \theta = \theta - v_t$$

實際上，使用動量的時候，就像將球推下山坡。球在下坡時滾動時會累積動量，在途中速度會越來越快（如果存在空氣阻力，直到達到極限速度，也就是 $\gamma < 1$ ）參數更新也發生了同樣的事情：動量 (momentum) 對於梯度指向相同方向的維度增加，而對於梯度改變方向的維減少動量。結果，我們獲得了更快的收斂並減少了振盪。

Nesterov accelerated gradient (NAG) 是一種使動量具有一個去向的概念，以便在山坡再次變高之前知道它會減速。我們知道使用動量 γv_{t-1} 來移動參數。計算 $\theta - \gamma v_{t-1}$ 這樣就給了參數的下一個位置的近似值（完整更新缺少的梯度），這是參數將要存在的大致概念。現在，通過計算與當前參數無關的梯度來有效地看到目前的參數 *theta* 將會移動到的位置：

$$v_t = \gamma v_{t-1} + \eta \cdot \nabla_{\theta} J_{(\theta - \gamma v_{t-1})} \quad \theta = \theta - v_t$$

同樣，我們設置動量 γ 約為 0.9。動量首先計算當前梯度（(圖.2.10) 中的藍色小向量），然後在更新的累積梯度（藍色向量）的方向上發生較大的跳躍，而 NAG 首先在先前的累積梯度的方向上進行較大的

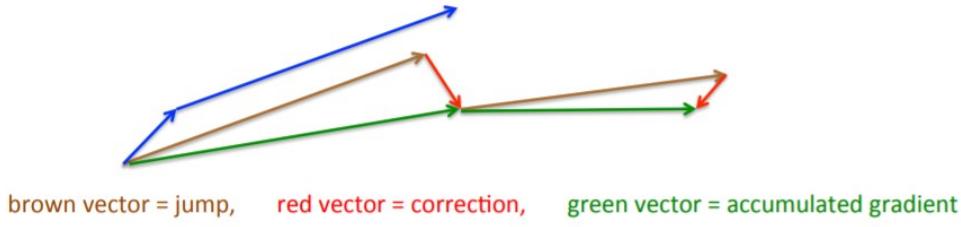


圖. 2.10: NAG[12]

跳躍（棕色向量），測量梯度，然後進行校正（紅色向量），從而完成 NAG 更新（綠色向量）。這種預期的更新可防止我們過快地進行，並導致響應速度增加，從而顯著提高了 RNN 在許多任務上的性能。

Adagrad[12]：

Adagrad 是一個梯度優化的算法，它可以做到：學習率適應參數，對於頻繁出現的特徵相關參數執行較小的更新（較低的學習率），以及對不經常出現的特徵相關參數進行較大更新（即學習率較高）。Adagrad 可以提高 SGD 的強度，用於訓練大型神經網絡。

先前，在同一次 θ 參數（更新後就算另一次），每個 θ 都使用相同的 η （學習率）。Adagrad 則是對每個 θ 參數使用不同的 η ， t 代表 time step。先將 Adagrad 的更新參數向量化。用 g_t 表示目標函數（參數 θ 在 time step t ）對參數做偏微分計算。

$$g_{t,i} = \nabla_{\theta} J_{(\theta_{t,i})}$$

當 SGD 更新每個參數 θ_i ，在每個 time step t ，因此變成：

$$\theta_{t+1,i} = \theta_{t,i} - \eta \cdot g_{t,i}$$

更新規則，Adagrad 根據先前 θ_i 計算的梯度，對每個參數 θ_i 修改整個學習率 η 在每個 time step t：

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}}$$

$G_t \in \mathbb{R}^{d \times d}$ 這是一個對角矩陣每個對角元素 i, i 是關於 θ 梯度平方和取決於 time step t， ϵ 是避免分母為 0 (ϵ 通常為 10^{-8})，如果沒

有平方根運算，該算法的性能將大大降低。 G_t 包含了過去梯度平方根，由於全部 θ 參數沿著對角線，通過向量的內積計算 G_t 和 g_t :

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t$$

Adagrad 主要好處之一是，無需手動調整學習率。大多數實現使用預設值 0.01 並將其保留為預設值。Adagrad 主要弱點是會累積分母的平方梯度：由於每項都是正的，累積和會在訓練中不斷增長。反過來，學習率下降，並最終變得無限小，這算法就不再獲得知識。

Adadelta[12]：

Adadelta 是 Adagrad 的延伸，下降其激進的程度，單調的降低學習率。Adadelta 會限制過去累積的梯度，並將其限制在某個特定大小 w ，並代替 Adagrad 過去累積的梯度平方，以梯度總和是遞迴定義為所有過去衰減梯度平方平均值。流動平均 $E[g^2]_t$ 在 time step t 然後取決於（像 Momentum 的 γ ）先前平均和最近梯度：

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g_t^2$$

γ 值和 Momentum 的相似，約為 0.9，現在根據參數更新向量 $\Delta\theta_t$ 來重寫 SGD：

$$\Delta\theta_t = -\eta \cdot g_{t,i}$$

$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

Adagrad 的參數更新向量替換成：對角矩陣 G_t 過去梯度平方的衰退平均 $E[g^2]_t$

$$\begin{aligned} \Delta\theta_t &= -\frac{\eta}{\sqrt{G_t + \epsilon}} \cdot g_t \\ \text{replace } G_t \text{ with } E[g^2]_t &\Rightarrow \Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \cdot g_t \end{aligned}$$

由於分母只是梯度的均方根 (RMS)，我們可以取代成縮寫：

$$\Delta\theta_t = -\frac{\eta}{\text{RMS}[g]_t} \cdot g_t$$

這個更新單位和 SGD、Momentum 以及 Adagrad 的單位不符合，因此更新需有相同的參數。為了實現這一點，首先定義另一個指數衰減平均值，這次不是梯度平方更新而是參數平方更新：

$$E[\Delta\theta^2]_t = \gamma E[\Delta\theta^2]_{t-1} + (1 - \gamma) \Delta\theta_t^2$$

RMS 參數更新：

$$RMS[\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

$RMS[\Delta\theta]_t$ 是未知的，更新參數的 RMS 取近似值到上個 time step。用 $RMS[\Delta\theta]_t$ 取代學習率 η ，最後產生新的規則：

$$\begin{aligned}\Delta\theta_t &= -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t \\ \theta_{t+1} &= \theta_t + \Delta\theta_t\end{aligned}$$

使用 Adadelta，甚至不需要設定預設學習率，因為它已從更新規則淘汰。

RMSprop[12]：

RMSprop 是 Geoffrey Hinton 在他的課程中提出的未公開自適應學習率的方法。

RMSprop 和 Adadelta 都是為了解決 Adagrad 的學習率急劇下降的問題個別獨立開發出來的解決方式。RMSprop 實際上與 Adadelta 得出的第一個更新向量相同：

$$\begin{aligned}E[g^2]_t &= 0.9 E[g^2]_{t-1} + 0.1 g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t\end{aligned}$$

RMSprop 也將學習率除以梯度平方的指數衰減平均值。Hinton 建議 γ 設為 0.9，好的預設學習率 γ 數值為 0.001。

Adaptive Moment Estimation : [12]

Adaptive Moment Estimation 自適應矩評估 (Adam) 是另一種計算每個

評估學習率的方法。除了儲存過去梯度平方的指數衰減平均值 v_t ，就像 Adadelta 和 RMSprop 一樣，Adam 還保留過去梯度的指數衰減平均值 m_t ，類似動量 (Momentum)。如果 Momentum 被視為順著斜坡下滑的球，而 Adam 則是像一個帶有摩擦的沉重的球，因此更適合待在 error face 平坦的最小值區域。計算過去梯度平方的衰減平均值 m_t 和 v_t 分別如下：

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

m_t 和 v_t 分別是第一階矩平均估計值和第二階矩無中心方差估計值，因此是方法的名稱。像 m_t 和 v_t 被初始化為向量 0，Adam 的作者觀察到它們偏向零，特別是在初始 time step，尤其是在衰減率較小的時候 (也就是說 β_1 和 β_2 趨近於 1) 藉由計算校正偏差第一矩 \hat{m}_t 和第二矩 \hat{v}_t 抵消偏差：

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

使用他們去更新參數，就像 Adadelta 和 RMSprop 中所看到的那樣，這將產生 Adam 更新規則：

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

β_1 預設值建議為 0.9， β_2 預設值建議為 0.999， ϵ 預設值建議為 10^{-8} 。根據經驗證明 Adam 表現良好，並且與其他自適應學習算法相比具有優勢。在 Adam 更新規則中的 v_t 係數是與梯度成反比地縮放過去梯度的範數 (通過 v_{t-1} 項) 和當前梯度 $|g_t|^2$ ：

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) |g_t|^2$$

我們轉換這個更新到 ℓ_p 。注意 β_2 參數化為 β_2^p ：

$$v_t = \beta_2^p v_{t-1} + (1 - \beta_2^p) |g_t|^p$$

大規範 p 值使數值上變得不穩定，這就是為什麼 ℓ_1 和 ℓ_2 規範在實踐中是最常見的。然而 ℓ_∞ 通常也表現出穩定的行為。為了避免與 Adam 混用，所以使用 u_t 來表示無窮範數約束 v_t ：

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty) |g_t|^\infty \\ &= \max(\beta_2 \cdot v_{t-1}, |g_t|) \end{aligned}$$

替換為 Adam 更新公式 $\sqrt{\hat{v}_t} + \epsilon$ 和 u_t 得出 AdaMax 更新規則：

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \hat{m}_t$$

替換為 Adam 更新公式 $\sqrt{\hat{v}_t} + \epsilon$ 和 u_t 得出 AdaMax 更新規則：

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t} \hat{m}_t$$

注意 u_t 依靠最大運算，不建議 Adam 中的 m_t 和 v_t 偏向零，這就是為什麼不需要針對 u_t 計算偏差。好的預設值 $\eta = 0.002$ $\beta_1 = 0.9$ 和 $\beta_2 = 0.999$ 。

2.2 強化學習

強化學習 (Reinforcement Learning, 簡稱為 RL) 是通過 agent(代理)與已知或未知的環境持續互動，不斷適應與學習，會得到正向或負面的回饋，對應到獎賞 (reward) 和懲罰 (punishments)。考慮到 agent 與環境 (environment) 互動，進而決定要執行哪個動作，強化學習的學習模式是建立在獎賞與懲罰上。

強化學習與其他學習法不一樣的地方在於：不需要事先收集大量數據提供當作學習樣本，而是透過與環境互動，在環境下發生的狀態當作學習的資料來源，透過不斷嘗試使所得到的獎勵最大化。其他類型的機器學習大都需要給予特定資料且有明確的答案。

由於強化學習是建立在 agent 與環境互動上，因此許多參數進行運算，需要大量資訊來學習，並根據資訊採取行動。強化學習的環境可以是真實世界、2D 或 3D 模擬世界的場景。強化學習的範圍很廣，因為環境的規模可能很大，且在環境中有多相關因素，影響著彼此。強化學習以獎勵的方式，促使學習結果趨近或達到目標結果。

強化學習涵蓋範圍 (圖.2.11)：

強化學習可以運用在計算機科學、神經科學、心理學、經濟學、數學、工程等領域，涵蓋領域相當廣泛。

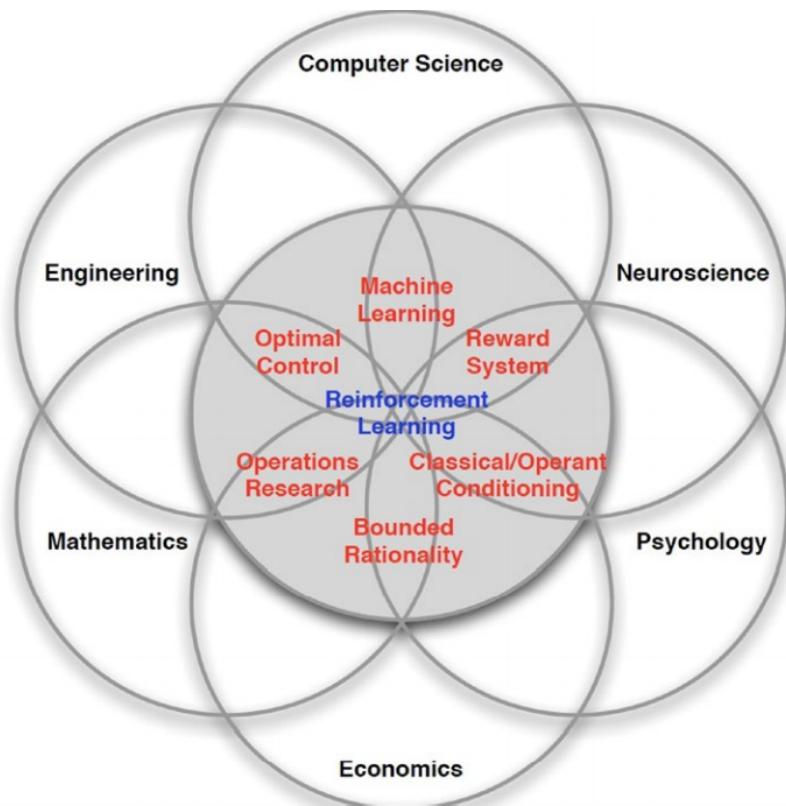


圖. 2.11: 各領域與機器學習應用範圍

強化學習的流程：

透過 agent 與環境間互動而產生狀態和獎勵，由於狀態的轉移，agent 會決定接下來執行動作（圖.2.12）。

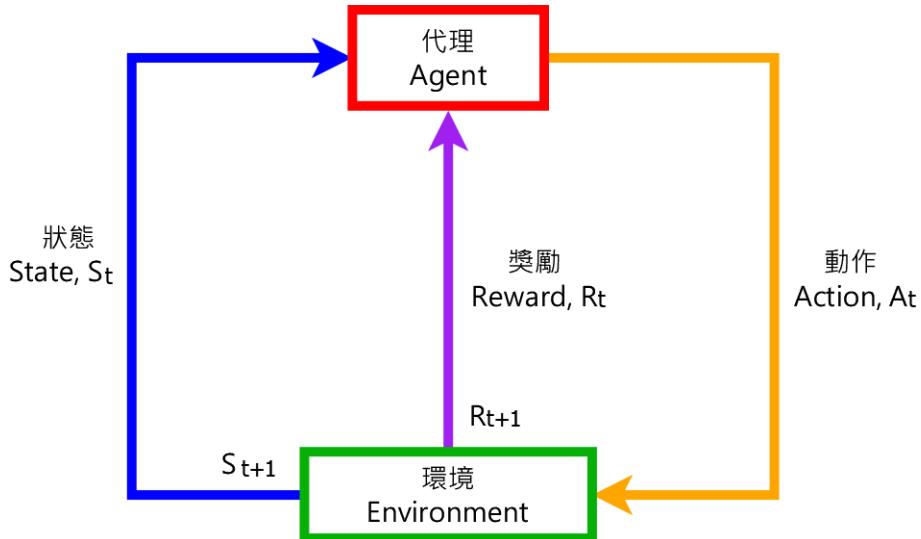


圖. 2.12: 強化學習架構

需要考慮的重點：

強化學習的狀態、獎勵和動作是互相關聯，agent 與環境之間存在著關聯，兩者都影響著狀態和動作並互相影響著彼此：機器人會因動作而造成狀態轉移，狀態的移轉也會影響機器人做出的決策。

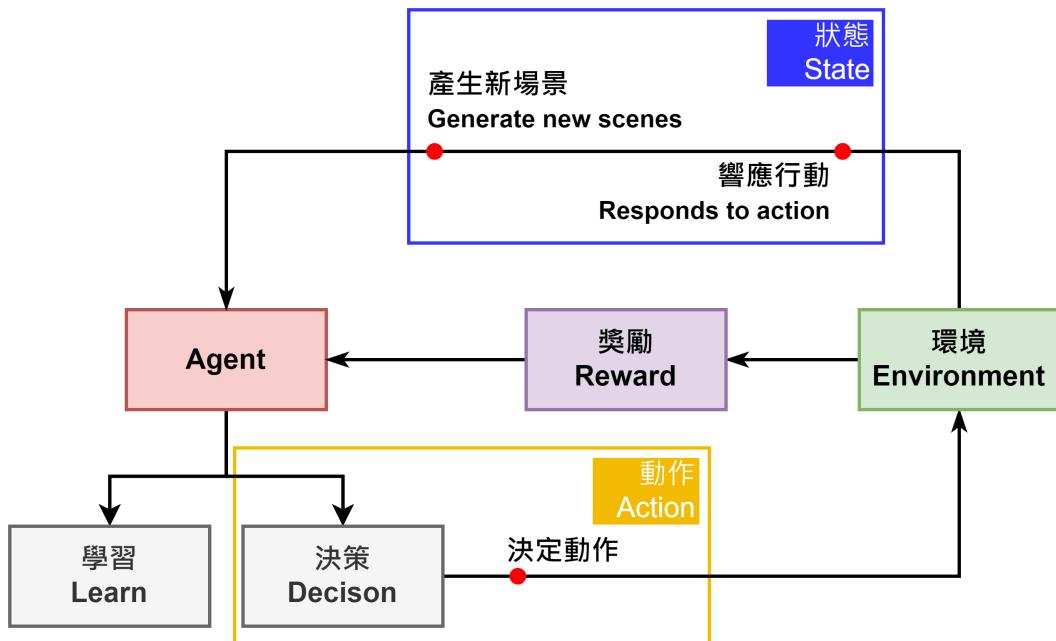


圖. 2.13: 整個互動過程

如(圖.2.13)agent 會透過輸入的狀態來決定採取何種行為(動作)，並試圖採取獲得最高獎勵的行動。當 agent 開始與環境互動時，agent 會透過當前狀態來決定將採取的行動，在 agent 採取行動後，環境的狀態也因此而改變，若 agent 採取行動後所到達我們所要的狀態就會得到獎勵，反之則會給予懲罰。在場景裡透過反覆的訓練，讓強化學習的行為漸漸趨近預期的目標。

強化學習中有兩個很重要的常數： γ 和 λ 。

γ 會影響所獲得的獎勵。 γ 又稱為衰減因子，正常狀態下為小於 1 的常數用於每個狀態改變，當狀態改變時為時常數。 γ 允許使用者在每個狀態給予不同形式的獎勵(這種狀況下 γ 為 0)，如果著重在長期的決策時，獎勵就不受決策順序所影響(此時 γ 為 1)

λ 一般在我們處理時間差異問題時使用。這是涉及更多地連續狀態的預測。在每個狀態中 λ 值的增加代表演算法正在快速學習。

強化學習的互動是透過 agent 和環境之間的互動會產生獎勵，agent 採取行動，導致狀態改變是一種強化學習實現如何將情況映設為行動的方法，從而找到最大化獎勵的方法，機器或機器人不會像其他機器學習形式的機器人那樣被告知要採取哪些行動。

獎勵的目的與運作以獎勵的方式誘導機器採取我們所期望的動作，機器會採取最大化獎勵的方式，因此可將目的定為最大獎勵，以吸引機器執行期望做的行為。



圖. 2.14: agent

強化學習的環境：

強化學習中的環境由某些因素組成，會對 agent 產生影響，agent 必須根據環境適應各種因素，並做出最佳決策，這些環境可以是各種形式，其中包括 2D、3D 或是真實世界。強化學習的環境具有確定性、可觀察性，可以是離散或是連續的狀態，則 agent 可以是單一或多個所組成。

2.2.1 馬可夫決策

- Markov Chain

馬可夫鏈 (Markov Chain) 主要是狀態變化的隨機過程 (stochastic process) 和馬可夫屬性 (Markov property) 結合。隨機過程 (stochastic

process) 狀態隨著時間變化，而狀態的變化存在著隨機性，並以數學模式表示。馬可夫屬性 (Markov property) 指在目前以及所有過去事件的條件下，任何未來事件發生的機率，和過去的事件不相關僅和目前狀態相關。當前決策只會影響下個狀態，當前狀態轉移 (action) 到其他狀態的機率會有所差異。

- Markov Reward Process

- action 到指定狀態會獲得獎勵。

$$R(s_t = s) = \mathbb{E}[r_t | s_t = s]$$

$$\gamma \in [0, 1]$$

- Horizon：在無限的狀態以有限的狀態表示。

- Return：越早做出正確決策獎勵越高。

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \dots + \gamma^{T-t-1} R_T$$

- State value function(決策價值)：

$$V_t(S) = \mathbb{E}[G_t | s_t = s]$$

$$P(s_{s+1} = s' | s_t = s, a_t = a)$$

- Discount Factor (γ) 嘉勵衰減有幾種作法：第一種，越早做出有獎勵的決策，獎勵越高；第二種，做出有價值的決策 $\gamma = 1$ ，不分決策順序先後；第三種，無用的決策 $\gamma = 0$ ，不會得到獎勵。

以 Bellman equation 的方式描述互動關係狀態：

$$V(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s)V(s')$$

$R(s)$: 立即獎勵

$$\gamma \sum_{s' \in S} P(s'|s)V(s') : \text{未來獎勵衰減總和}$$

Anaytic solution(分析性解法)，MRP 的分析性解法：

$$V = (1 - \gamma P)^{-1} R$$

Bellman equation 及 Anaytic solution 的方式只適合小的 MRP(個數比較少的)，矩陣複雜度為 $O(N^3)$ ，N 為狀態個數。若要計算大型的 MRP 會使用疊代法：動態規劃 (Dynamic programming)、Temporal-Difference learning 和 Monte-Carlo evaluation 以評估採樣的方式：

$$g = \sum_{i=t}^{H-1} \gamma^{1-t} r_i$$

$$G_t \leftarrow G_t + g, i \leftarrow i + 1$$

$$V_t(s) \leftarrow \frac{G_t}{N}$$

- Markov Decision Process 在 MRP 中加入決策 (decision) 和動作 (action)

– S : state 狀態

– A : action 動作

– P : 狀態轉換 $P(s_{t+1} = s' | s_t = s, a_t = a)$

– R : 獎勵，取決於當前狀態和動作會得到相對應的獎勵

$$R(s_t = s, a_t = a) = \mathbb{E}[r_t | s_t, a_t = a]$$

– D : 折扣因子 (discount factor)

$$\gamma \in [0, 1]$$

policy(決策)：可以是一個決策行為的機率或確定執行的行為，若以數學方程式表示：

$$\pi(a|s) = P(a_t = a | s_t = s)$$

MRP 和 MDP 方程式互相轉換：

MRP	\longleftrightarrow	MDP
$P^\pi(s's)$	=	$\sum_{a \in A} \pi(a s) P(s' s, a)$
$P^\pi(s)$	=	$\sum_{a \in A} \pi(a s) P(s, a)$

state value function(狀態值方程式) $v^\pi(s)$

$$\begin{aligned}
 v^\pi(s) &= \mathbb{E}[G_t | s_t = s] \\
 &= \mathbb{E}[R_{t+1} + \gamma v^\pi(s_{t+1}) | s_t = s] \\
 &= \sum_{a \in A} \pi(a|s) q^\pi(s, a)
 \end{aligned}$$

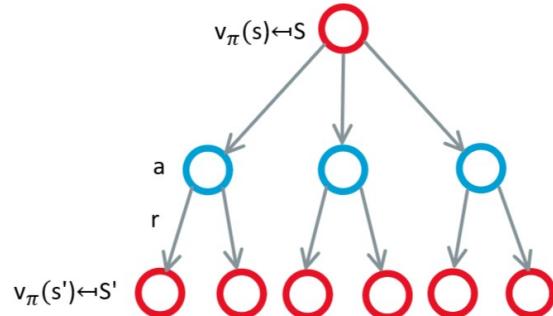


圖. 2.15: v^π 程序圖

$$v^\pi(s) = \sum_{a \in A} \pi(a|s) (R(s, a) + \gamma \sum_{s' \in s} P(s'|s, a) v^\pi(s'))$$

state value function(狀態值方程式) $q^\pi(s)$

$$v^\pi(s) = \mathbb{E}[G_t | s_t = s]$$

$$= \mathbb{E}[R_{t+1} + \gamma v^\pi(s_{t+1}) | s_t = s]$$

$$= \sum_{a \in A} \pi(a|s) q^\pi(s, a)$$

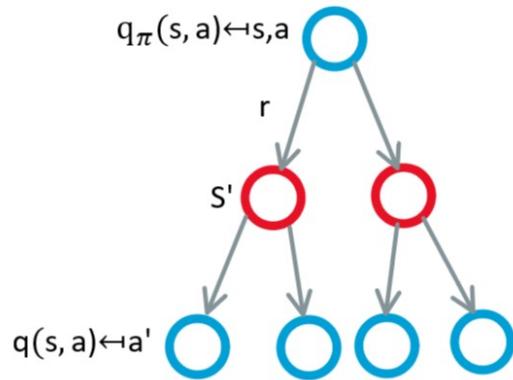


圖. 2.16: q^π 程序圖

$$q^\pi(s, a) = R(s, a) + \gamma \sum_{s' \in S} P(s'|s, a) \sum_{a' \in A} \pi(a'|s') q^\pi(s', a')$$

2.3 Policy Gradient 理論

Actor, Environment, Reward

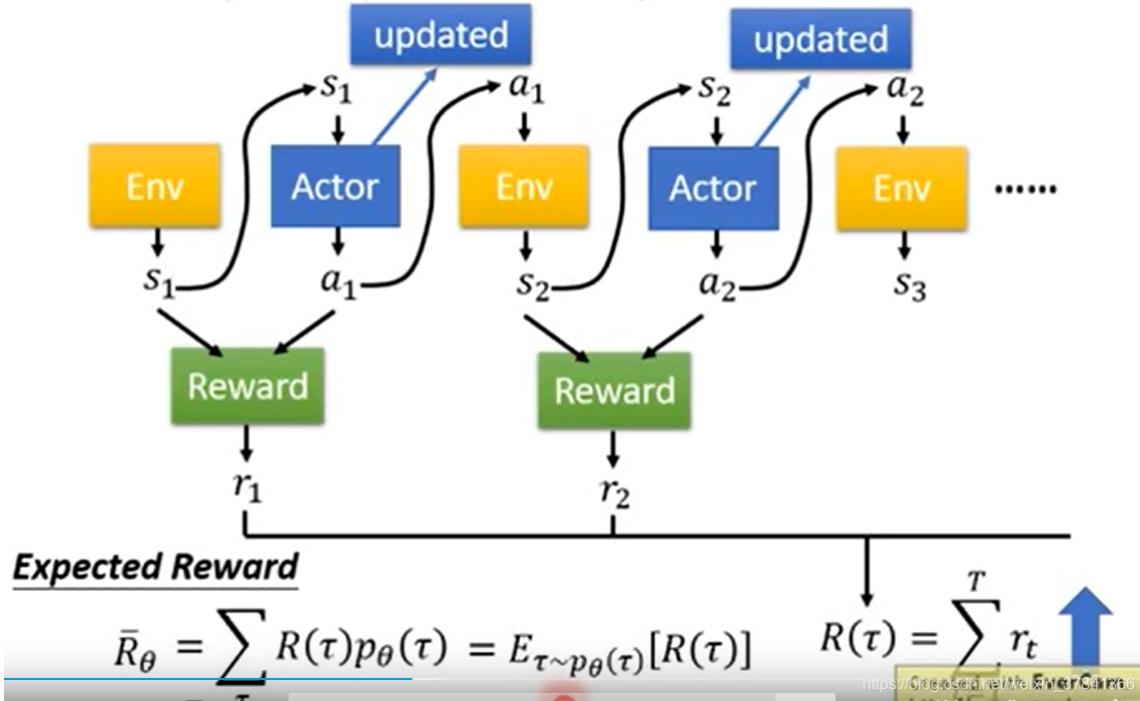


圖 . 2.17: Policy Gradient 原理

Policy Gradient 主要目的是直接對決策進行建模與優化。該決策 (policy) 通常使用參數化函數建模，獎勵 (目標) 函數的值取決於此決策，可以應用各種算法來優化，以獲得最佳獎勵。(參數化：當軟體建置於一給定環境時，再依該環境的實際需求填選參數，即可成為適合該環境。)

參數介紹 [7]:

π : policy

s : 狀態 (States)。

a : 動作 (Actions)。

r : 獎勵 (Rewards)。

S_t, A_t, R_t : 一個軌跡時間步長't' 的 State, Action and Reward。

γ : 衰減因子 (Discount Factor); 懲罰未來的不確定獎勵 (reward)。

G_t : 回傳衰減後的未來獎勵 (Discounted future reward) $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$

$P(s', r|s, a)$ ：伴隨著現在狀態 (state) 的 a 和 r ，前往下一個狀態 s' 的轉移機率矩陣 (單階)。

$\pi(a|s)$ ：隨機策略 (agent 的行為策略)。

$\mu(s)$ ：確定的策略；我們還使用不同的字母將其標記為 π_s ，以提供更好的區分，以便我們可以輕鬆判斷策略是隨機的還是具有確定性的

$V(s)$ ：'狀態值函數' 測量狀態的預期收益 (報酬率)

$V^\pi(s)$ ：根據 policy 的狀態值函數 $V^\pi(s) = \mathbb{E}_{a \sim \pi}[G_t | S_t = s]$

$Q(s, a)$ ：行為值函數，評估一對狀態和動作的預期收益。

$Q^\pi(s, a)$ ：根據 policy 的行為值函數 $Q^\pi(s, a) = \mathbb{E}_{a \sim \pi}[G_t | S_t = s, A_t = a]$ 。

$A(s, a)$ ：Advantage Function， $A(s, a) = Q(s, a) - V(s)$ ：像是另一種版本的 Q-value，由狀態值為基準降低方差。

reward function 的值：取決於策略，可應用各種算法優化 θ ，獲得最佳獎勵。

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)$$

Policy Gradient 通過反覆評估梯度來最大化預期的總獎勵 (reward)

$$g = \nabla_\theta \mathbb{E}[\sum_{t=0}^{\infty} r_t] ; g = \mathbb{E}[\sum_{t=0}^{\infty} \psi_t \nabla_\theta \log \pi_\theta(a_t | s_t)]$$

ψ_t 可能方法為下列：

- $\sum_{t=0}^{\infty} r$ ：決策軌跡的獎勵總和。
- $\sum_{t'=t}^{\infty} r'$ ：根據動作 (action) 的獎勵 (reward) a_t 。
標準表示式： $\sum_{t'=t}^{\infty} r' - b(s_t)$
- $Q^\pi(s_t, a_t)$ ：state-action value function。
- $A^\pi(s_t, a_t)$ ：Advantage Function。
- $r_t + V^\pi(s_t + 1) - V^\pi(s_t)$ ：TD residual。

公式使用定義 [7] :

$$V^\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t:\infty}} [\sum_{l=0}^{\infty} r_t + l]$$

$$Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1:\infty}, a_{t+1:\infty}} [\sum_{l=0}^{\infty} r_t + l]$$

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \text{ (Advantage Function)}$$

2.3.1 Actor Critic

原始的 policy gradient 沒有偏差，但方差大；所以提出了許多以下算法來減少方差，同時保持偏差不變：

$$g = \mathbb{E} \left[\sum_{t=0}^{\infty} \psi_t \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right]$$

Actor-Critic：減少原始政策中的梯度方差包括兩個模型

Critic：更新值函數參數 w ，根據算法，它可以是操作值 $Q_w(a|s)$ 或狀態值 $V_w(s)$

Actor：按照 Critic 的建議，將策略參數 θ 更新為 $\pi_\theta(a|s)$

它如何在簡單的行動價值參與者批評中發揮作用：

- 隨機的初始化 s, θ, w ；取樣 $a \sim \pi_\theta(a|s)$

- For $t = 1 \sim T$:

1 取樣 reward $r_t \sim R(s, a)$ 隨後下一階段 $s' \sim P(s'|s, a)$

2 樣本的下一個動作 $a' \sim \pi_\theta(a'|s')$

3 更新 policy 參數 θ :

$$\theta \leftarrow \theta + \alpha_{\theta} Q_w(s, a) \nabla_{\theta} \ln \pi_{\theta}(a|s)$$

4 計算校正 (TD error) 對於時間 t 的動作值：

$$\delta = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$

並使用它來更新操作 action - value function:

$$w \leftarrow w + \alpha_w \delta \nabla_w Q_w(s, a)$$

5 更新 $a \leftarrow a'$ 和 $s \leftarrow s'$ ；學習率： α_θ 和 α_w 。

第三章 訓練環境

3.1 OpenAI Gym

Gym 是用於開發和比較強化學習算法的工具包，他不對 agent 的結構做任何假設，並且與任何數據計算庫兼容，而可以用來制定強化學習的算法。這個環境具有共享的介面，使我們能用來編寫常規算法，也就能够教導 agents 如何步行到玩遊戲。

3.2 Pong

取自 1977 年發行的一款家用遊戲機 ATARI 2600 中的遊戲，內建於 Gym，這是一個橫向的乒乓遊戲，左方是預設電腦玩家，右邊由使用者或是由訓練程式控制（圖.3.1）。在強化學習範例中，Pong 與實體冰球機簡化後環境相似。

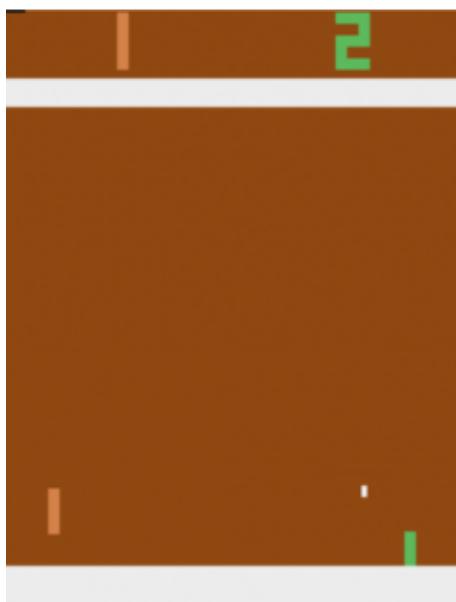


圖. 3.1: ATARI Pong

第四章 模擬環境

4.1 模擬模型

在模擬的模型上，延用了學長設計的冰球機，並進行了部分的設計變更，將原本的人機對打更改為機器對打，且因為搭配深度強化學習的訓練，所以將兩邊的擊球器都僅保留 X 軸向（左右）移動，而冰球則是使用原本設計。多虧了學長們所設計的冰球機模型，讓我們在運作上有問題時可以直接發問，設計變更的地方也可以快速完成。

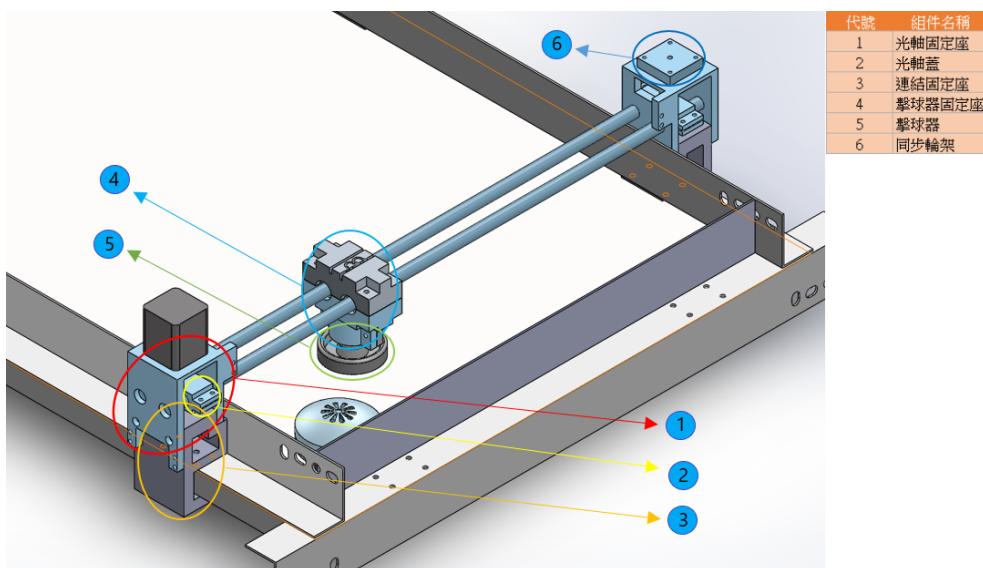


圖. 4.1: 組合圖

將原本 Y 軸移動機構移除，並將其改為固定在特定位置上，此固定座設計是取代原本鎖在光軸固定座上的（圖.4.1 代號 1）Y 軸皮帶固定座（圖.4.2），並使光軸固定座可以通過連結固定做鎖固於桌面，如（圖.4.3）。

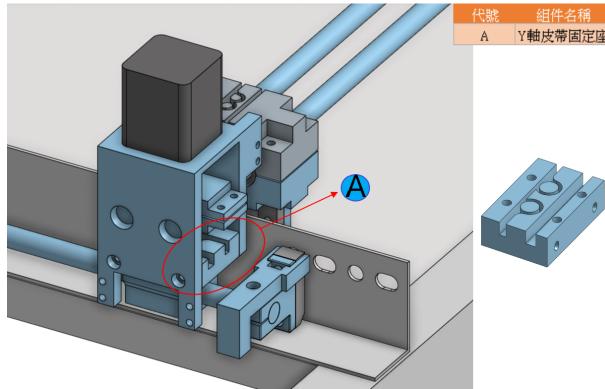


圖. 4.2: Y 軸皮帶固定座

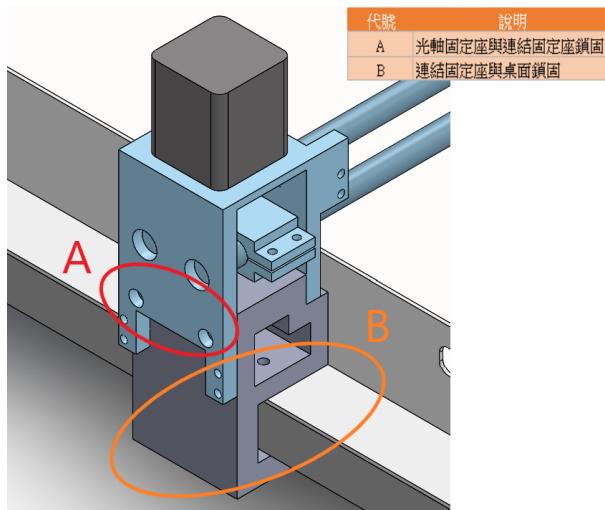


圖. 4.3: 連結固定座

分別在擊球器外側保留約冰球直徑 1.5 倍之區域作為得分判定區，如圖 4.4 中的紅色區域。

4.2 CoppeliaSim 模擬

CoppeliaSim 是具有集成開發環境的機器人模擬器，基於分佈式控制體系架構，可以通過嵌入式腳本，插件，ROS 或 BlueZero 節點，RemoteAPI 客戶端或自定義解決方案進行模型控制。

且 CoppeliaSim 中，控制器可以用 C / C ++、Python、Java、Lua、Matlab 或 Octave 編寫。



圖. 4.4: CoppeliaSim Logo

4.2.1 使用原因

本專題之最終目標是希望可以在虛擬環境中進行深度強化學習來訓練機器對打，通過虛擬環境中的模擬後，可以更直接地看到深度強化學習訓練的狀況，且因為在虛擬環境中不會有金費的支出，所以可以不斷的重複模擬直到模擬達到最佳的狀態，除此之外 CoppeliaSim 的虛擬環境更接近真實環境，基於以上原因，所以使用了 CoppeliaSim 開發。

4.2.2 RemoteAPI

RemoteAPI(Remote Application Programming Interface) 是 CoppeliaSim API 框架的一部分。它允許 CoppeliaSim 與外部應用程序之間的通訊，是跨平台並支持服務調用和雙向數據流。有兩個不同的版本/框架分別為:Remote API 和 The B0-based remote API。

4.2.3 常用功能

1. 以下為簡易功能說明:



圖. 4.5: CoppeliaSim 工具列



圖. 4.6: CoppeliaSim 工具列 (續)

代號	功能說明	代號	功能說明
1	畫面平移	10	複製所有設定
2	畫面旋轉	11	回復/取消回復
3	畫面縮放	12	模擬設定
4	畫面視角	13	開始/暫停/停止模擬
5	畫面縮放至適當大小	14	即時模擬切換
6	選取物件	15	模擬速度控制
7	移動物件	16	線程渲染/視覺化
8	旋轉物件	17	場景/頁面選擇
9	加入/移出樹狀結構		

表. 4.1: 功能說明

4.3 影像處理

在影像處理中我們主要使用了 Python 套件中的 OpenCV(全稱:Open Source Computer Vision Library), 並搭配其他套件或模組進行了影像處理, 藉此來取得訓練神經網路訓練時所需的資訊。



圖. 4.7: OpenCV 及 Python logo

4.3.1 CoppeliaSim 中的 Vision sensor(視覺傳感器)

CoppeliaSim 的視覺傳感器輸出的影像是以每個像素中以 RGB 三個位元組所組成的，舉例來說: 在 CoppeliaSim 中視覺傳感器取出畫面像素為 512×256 ，則我們會接收到 $(512 \times 256) \times 3 = 393,216$ 個資料，是一筆相當大的資料，所以在影像處理上會消耗掉大量的資源。

4.3.2 影像辨識

透過 CoppeliaSim 中的 Vision sensor 接收場景影像並輸出後，便可以開始進行影像辨識的處理。

1. RGB 與 HSV 的轉換 [13]

RGB 即光的三原色 Red(紅)Green(綠)Blue(藍)，HSV 則是一種將 RGB 色彩模型中的點在圓柱坐標系中的表示法，HSV 分別表示 Hue(色相)、Saturation(飽和度)、Value(明度)，而會將 RGB 轉換為 HSV 是因為 HSV 相較於 RGB 可以更直接的判斷色彩、明暗和鮮豔度對於顏色過濾可以更方便定義出色彩範圍。

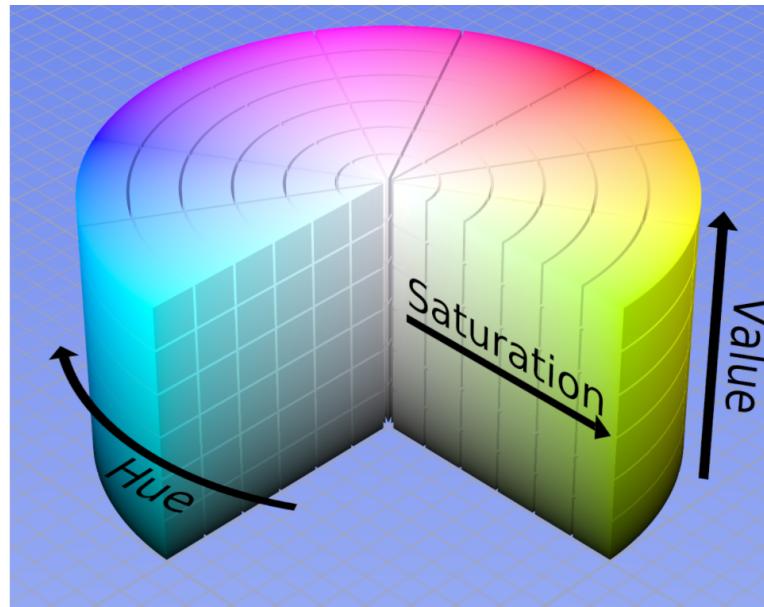


圖. 4.8: HSV 色彩空間

下列為 RGB 與 HSV 之間轉換的公式，首先是 RGB 轉為 HSV，其中 \max 及 \min 分別為 (r, g, b) 中的最大與最小值：

$$h = \begin{cases} 0^\circ, & \text{if } \max = \min \\ 60^\circ + \frac{g-b}{\max-\min} + 0^\circ, & \text{if } \max = r \text{ and } g \geq b \\ 60^\circ + \frac{g-b}{\max-\min} + 360^\circ, & \text{if } \max = r \text{ and } g < b \\ 60^\circ + \frac{g-b}{\max-\min} + 120^\circ, & \text{if } \max = g \\ 60^\circ + \frac{g-b}{\max-\min} + 240^\circ, & \text{if } \max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } \max = 0 \\ \frac{\max-\min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise} \end{cases}$$

$$v = \max$$

接著是 HSV 轉為 RGB：

$$\text{when } 0 \leq H < 360, 0 \leq S \leq 1, 0 \leq V \leq 1$$

$$C = V \times S$$

$$X = C \times (1 - |(H/60^\circ) \bmod 2 - 1|)$$

$$m = V - C$$

$$(R', G', B') = \begin{cases} (C, X, 0), & 0^\circ \leq H < 60^\circ \\ (X, C, 0), & 60^\circ \leq H < 120^\circ \\ (0, C, X), & 120^\circ \leq H < 180^\circ \\ (0, X, C), & 180^\circ \leq H < 240^\circ \\ (X, 0, C), & 240^\circ \leq H < 300^\circ \\ (C, 0, X), & 300^\circ \leq H < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

2. 顏色過濾

進行顏色過濾時，需要先定義出過濾顏色的上下限，在開始過濾後僅會保留介於上下界線範圍的影像，而介於上下限範圍之外的影像則會被剔除，如圖.4.10所示以上限 (77, 255, 255) 及下限 (35, 43, 46) 為例。

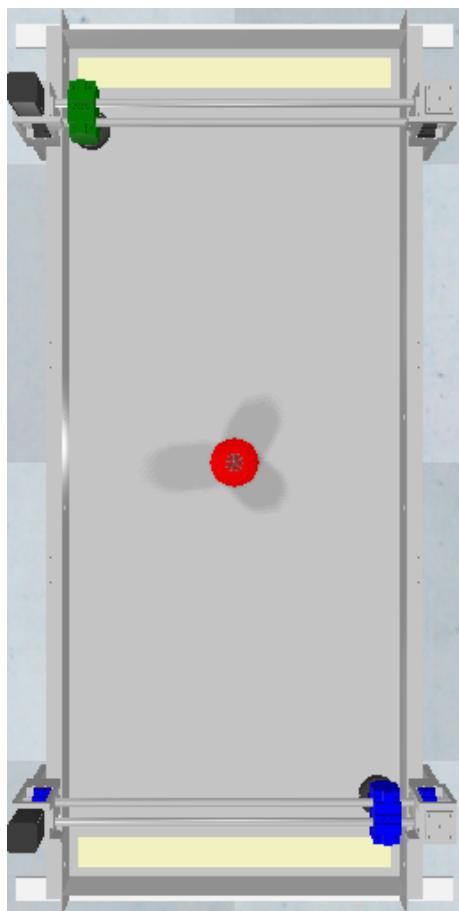


圖. 4.9: 場景原圖

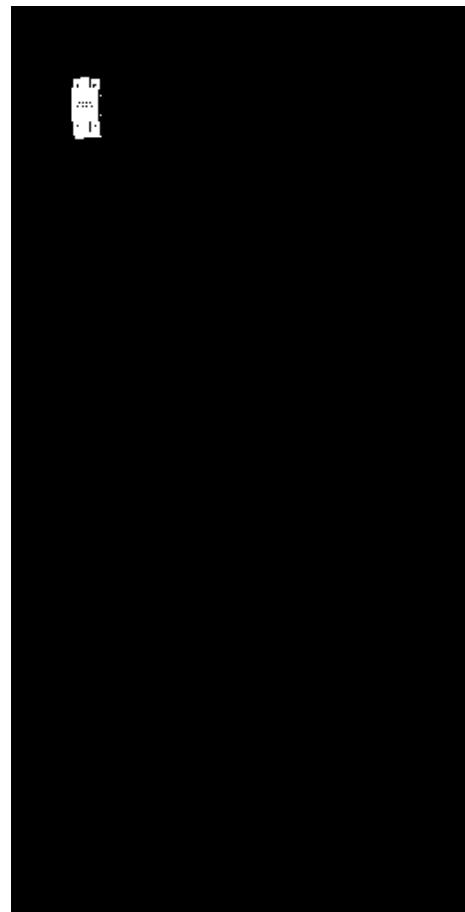


圖. 4.10: 顏色過濾後的場景

第五章 伺服器

此專題採用 Ubuntu 20.04 版本作為我們的架設所使用作業系統，由於 Ubuntu 功能尤為繁多，以下說明重點只著重在專題製作所用到功能上。

Ubuntu 作業系統是 Linux 系統的一個發行版，目前免費且開源，Ubuntu 基於 Debian 發行版和 GNOME 桌面環境，其目標在於為一般使用者提供一個最新、穩定又主要以自由軟體建構而成的作業系統。

其開發目的是為了使個人電腦變得簡單易用，它與其他基於 Debian 所發行的 Linux 版本更加接近 Debian 的開發理念，它主要使用自由、開源的軟體，而其他則帶很多閉源的軟體。

5.1 Ubuntu 環境配置

在一開始會先使用套件管理系統 apt 指令去下載 Xorg , fluxbox , lxde 套件。Xorg 是 Ubuntu 操作系統的一個顯示服務器軟件包，它在被導入 Ubuntu 操作系統後會載入一系列的文件或軟件，這些都是跟顯示卡驅動，圖形環境庫相關的一些文件、軟件。Gnome , kde，包括我們使用的 lxde 也需要 xorg 才能實現。而 Lxde 它的全名是 Lightweight X11 Desktop Environment ，是自由軟體桌面環境，其優點在於提供了輕量而快速的桌面環境，它比較重視實用、輕巧，除此之外它還可以在 Linux 平台執行。

之後需選擇 display manager (顯示管理器) 的種類，Display manager 是操作系統 Ubuntu 的組件，其中登錄的動作即為 Display manager 負責。該操作系統中常見的類型有 gdm ,gdm3 , lightdm ,kdm ... 。各類型的

Display manager 功能其實大同小異，差別在於外觀、操作、格式、複雜度和使用者感受等，可依使用者需求變更（有些較為輕量，適合比較低階的運行器）。選擇其中一個後繼續，之後可以切換更動。

再來是模組的導入，此處同樣用 apt 指令安裝：Pip , uwsgi , Nginx , 以及 Git 。如果要從 Ubuntu 系統上安裝軟體，其中一種方式是" pip "。「pip」是" pip Installs Packages " 的縮寫，是一個用命令列作為基礎的套件管理系統，可以用它來安裝 python 的應用程式。而使用 Git 是因為在備份資料時，可幫助使用者有效管理原始碼，而 github 就是由 Git 伺服器和網頁介面組成，用來當作放置原始碼的倉庫。

另外 Nginx 和 uwsgi 是為拿來配合把 python 程式應用在網路上實現，並且把想要的結果使其能在網路實時觀看操控結果之反饋。

5.2 Oracle VM VirtualBox 介紹

假使建構虛擬環境時需要在同一主機使用不同電腦作業系統環境，則可使用「虛擬機器工作站」—Oracle VM VirtualBox 。

選擇 Oracle VM VirtualBox 是為了因應當要使用不同作業系統（比如本機與虛擬環境不同作業系統）且不想與其資料存放時共用一個硬碟（無多餘硬碟，不想硬碟之間有資料重疊... ）時，即可使用其軟體做練習，降低操作失誤帶來的成本，而此軟體目前為免費，並隨時會更新，另外其特色有：

- 只要自備作業系統（光碟片, ISO 映像檔），即可在啟動 Oracle VM VirtualBox 後直接開啟要操作的執行檔（作業系統），不必再把主機本身重新關機，當然開啟多個作業系統之間也有共通性，可直接從視窗 A 做網路、檔案分享、複製貼上等動作到視窗 B 。

- 除了作業系統裡面的執行，還可在其中練習磁碟分割、格式化以及 BIOS 啟動等（但是未支援 USB 啟動）。
- 空間的佔用上並不是真實佔用空間，而是依據使用者的操作而變化（使用者用多少就是多少）。相對的，使用者雖然一開始設定該虛擬電腦的記憶體大小與硬碟空間是實時依據操作者決定，但終究還是佔掉電腦效能，所以 VirtualBox 的效能還是依據電腦本身的硬體配備。為了配置網路，首先在：

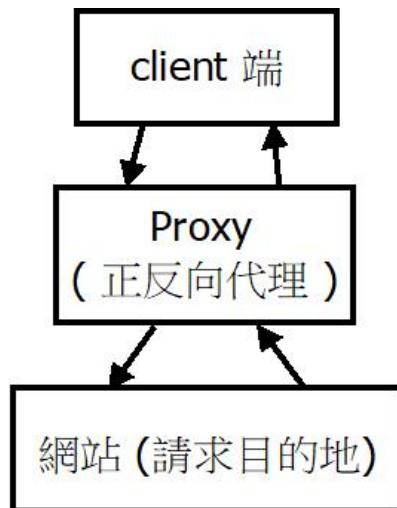
1. File / Preferences / Network 位置，新增一個或右鍵點擊現有的網路設定，填入該電腦網路設定。
2. Settings / Network / Adapter 1 / Attached to：該位置改成 Bridged Adapter

此處配置之比較（取常用例子）：NAT , Bridged , Internal , Host-only

- NAT：最為基本之設定，主要讓該虛擬主機可連上網，但在與其他網路使用者互動時找不到該虛擬主機網路位址，外部網路也無法偵測，虛擬主機所有的網路請求都會把該來源視為宿主機的。
- Host-only：虛擬主機被分配到一個網址，但是還是只有虛擬主機運行的環境可訪問該網路位址
- Internal：此種設定主要為虛擬主機彼此間的連線，它可向外部提供資料，但反之則不行。
- Bridged：在與其宿主機的網卡設定橋接與設定好外部網路位址後，可被外部網路訪問。

5.3 Web server

Nginx 是提供 web 相關服務的伺服器 (Web server)，除了是高效能的 HTTP (HTTPS) 服務器外，還可處理靜態資源，負載平衡，代理等工作。代理工作為根據不同域名轉發到 Application Server 的不同 port 上去處理，其中又分正向和反向，正向代理為 client 端發送 request 經由 proxy server 再到目標網站，反向則反之。正向代理操作中 server 只知道 proxy server 給他 request，不知道 client 是誰，而相同地反向代理則是 client 只知道 proxy server 給他 responses，不知道 server 是誰。正向代理隱藏真實 Client，反向代理隱藏真實 Server。另外在高流量的狀況下，需要多個 Application Server 來分擔流量，負載平衡就是負責 request 的分發，決定 request 要被分到哪一個 Application Server 處理。而關於處理靜態資源，Nginx 與 Apache 等 Web Server 處理靜態資源的能力是遠遠高於 Application Server 的。



其中代理可由 Nginx 負責

圖. 5.1: clientProxy

5.4 Nginx

網路設定：

首先要修改網路設定檔，而其設定檔放在/etc/netplan 目錄下的 yaml 檔。

其中需更改的設定：

1. addresses：為靜態 IP，可以是 IPV4 或 IPV6。
2. gateway：即為該電腦之閘道器。
3. nameservers：該電腦之 DNS 服務器。

WSGI (Python Web Server GateWay Interface) 為一種用在 Python 語言上的規範，用來規範 Web Server 與 Web Application 之間如何溝通。而 uWSGI 同為實現了 WSGI、uwsgi、http 協議等的 Web server，通常用於接收前端伺服器轉發的動態請求並轉發給 Web Application。前者可以使用 Nginx 提供的 https 協定，且同上表述中 Nginx 的靜態資源處理能力較佳所以也能將靜態資源轉給其處理。

Nginx 的主要設定檔 nginx.conf 可藉由 include 指令添加其他 nginx 設定檔的設定去擴增不同域名的設定，常見的設定有：

1. 預設：

程式. 5.1: nginx 預設

```
recvAPP {
    server localhost:5000;
    server localhost:5001;
}
server {
    listen 80;
    listen [::]:80;
    server_name SERVER_IP;
    root /home/hostname;
    location / {
        uwsgi_pass http://api/;
        include uwsgi_params;
    }
}
```

2. 負載平衡 LoadBalance :

程式. 5.2: load balance 設定

```
receiveAPP {
    ip_hash;
    server localhost:5000;
    server localhost:5001;
}
server {
    listen 80;
    listen [::]:80;
    server_name SERVER_IP;
    root /home/ryan;
    location / {
        uwsgi_pass 127.0.0.1:8000;
        include uwsgi_params;
    }
}
```

receiveAPP 定義了將 request proxy 過去的應用，例子中 server localhost 語法代表可以請求 proxy 到分別監聽 5000 與 5001 port 的兩個應用，同時這個 block 可達到 load balancer 負載平衡的功能。

server 這個 block 則是定義了 proxy server 的相關設定，包括要監聽的 port (listen 80 為監聽所有 IPV4 位址，listen [::]80 則為監聽所有 IPV6 位址)、規定哪些 domain 或 ip 的 request 會被 nginx server 處理 (server_name)。

location 像是路由 (routing) 的概念，設定不同的 path 要對應到怎麼樣的設定。location 中則是指對不同路徑的處理。

1. location :

程式. 5.3: location 設定

```
location / #匹配所有目錄
location /static #匹配所有 /static 的開頭目錄
```

要達到 load balancer 透過一開始介紹的 upstream block 就可以達成，在上面的例子中，來自某個 domain 80 port 會被分配到 port 5000 或 port 5001 兩個應用中，達成用兩個應用去分擔 request 的負載平衡器。

負載平衡裡的負載規則 (ip_hash) 某個 request 要被導到哪個應用去處理有不同規則，每個規則都有各自適合使用時機，以下簡單介紹幾個常見的規則：

1. round-robin (預設) 輪詢方式：也就是將請求輪流按照順序分配給每一個 server。假設所有伺服器的處理效能都相同，不關心每臺伺服器的當前連線數和響應速度。適合於伺服器組中的所有伺服器都有相同的軟硬體配置並且平均伺服器請求相對均衡的情況。不過也有另外一種可以設定權重的 Weight Round Robin (加權輪詢方式)，可以設定不同 server 的權重，例如以下範例：

程式. 5.4: 設定不同 server 的權重

```
upstream myweb {  
    server web1.dtask.idv.tw weight=3;  
    server web2.dtask.idv.tw weight=2;  
}
```

2. least-connected 最少連線：顧名思義為連線進來時會把 Request 導向連線數較少的 Server。
3. IP-hash 依據 Client IP 來分配到不同台 Server：通過一個雜湊 (Hash) 函式將一個 IP 地址對映到一臺伺服器。先根據請求的目標 IP 地址，作為雜湊鍵 (Hash Key) 從靜態分配的散列表找出對應的伺服器。除非斷線或 IP 變動，否則同個 IP 的請求都會導入到同一個 server。

uWSGI 設定 (uwsgi_ini) :

1. wsgi-file：主要運行的 py 檔案
2. http, socket, http-socket：端口設定，假使有使用到前端服務器（如 Nginx）時，不能用 http 設定，因 uwsgi 協議為 HTTP，而 Nginx 使用傳輸協議為 TCP，兩者不能互通。

程式. 5.5: 簡易 uwsgi 指令啟動

```
uwsgi --http :9000 --wsgi-file APP.py
```

3. processes、threads：工作序，processes 為進程，threads 為線程，下方設定為每條近程有兩條線程。

程式. 5.6: 加入工作程序 uwsgi 指令啟動

```
uwsgi --http :9000 --wsgi-file APP.py --processes 4 --threads 2
```

4. chdir：此項是為了正確的加載模組/檔案

整體快速配置（這裡儲存成一個.ini 文件，其他還有 YAML、JSON、XML 格式等）：

程式. 5.7: 將 uwsgi 指令啟動動作設定成一個啟動檔

```
[uwsgi]
socket = :9000
processes = 4
threads = 2
chdir = location/to
wsgi-file = location/to/file
```

此項還可加上 status：此項為查看 uWSGI 內部的輸出數據

程式. 5.8: status

```
-- status 127.0.0.1:9001
```

實現之通訊流程

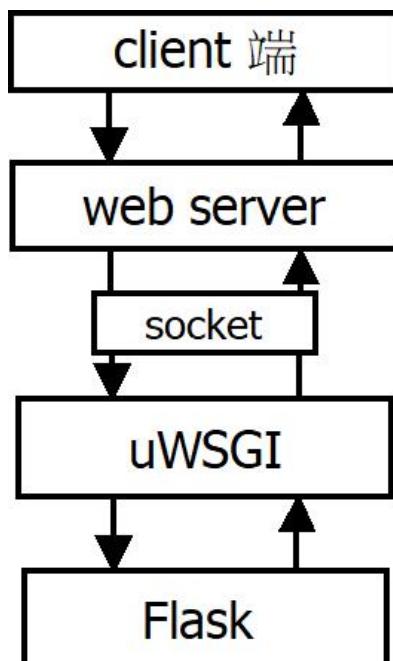


圖. 5.2: client To flask

5.5 Flask

如同以上所述，建立 Flask 框架的同時需選擇反向代理伺服器（這裡我們選擇了 Nginx）來負責網頁請求和結果的回覆，同時還需要一個實現 WSGI 通信協議的伺服器（我們選擇了 uWSGI）來負責接收代理伺服器的請求後 Flask 轉發及接收訊息，再轉發回去（代理伺服器）。

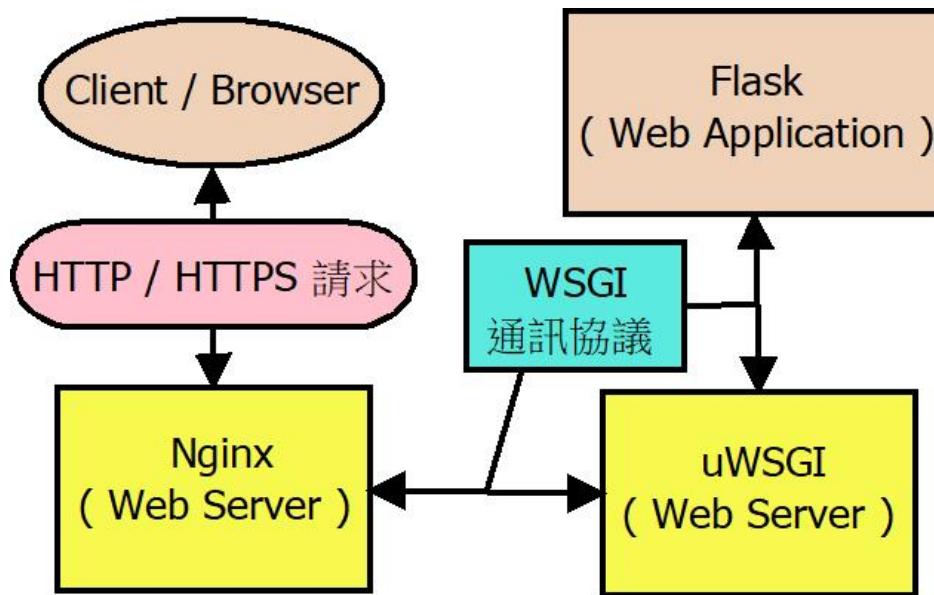


圖. 5.3: total

第六章 機器學習的訓練與模擬控制結果

6.1 訓練模型的基礎概念

訓練模型的原型是實體冰球機的機電系統，由於要訓練強化學習，所以需要將模型簡化至最簡潔的方式進行訓練，並取 Open AI Gym 的環境當作最簡化的訓練模型。由於強化學習是一種最佳化控制的方式，因此將 Gym 的 Pong 畫面當作輸入，輸出為擊錘移動方向，藉由調整當中的權重、偏差等參數，將參數調配到最優狀態。將可行的訓練方式套用到 CoppeliaSim 進行虛擬環境的訓練，並且可將訓練結果套用到實機進行運用。

6.2 訓練模型的選用

利用 Gym 的環境訓練機器學習，以測試學習率、神經網路隱藏層的神經元個數、機器學習的啟動函數類型、訓練時影像大小等幾項參數與訓練結果之間的關聯性，選用 python 語言進行配置。剛開始我們運用 Pygame 模組來撰寫 pong game 的訓練環境，開始學習並了解 Pygame 的一些運用，嘗試建構出 pong game 場景 (圖.6.1)，在基本功能編寫告一段落後，測試程式漏洞，發現對打時特定角度碰撞，球會超過擊錘的碰撞感測，導致出現球擊穿擊錘的現象。為了解決 Pygame 碰撞問題，做了幾種嘗試：修改 Pygame 的碰撞定義，更換碰撞感測的感測方式，問題依舊沒有顯著的改善，若增加過多的碰撞偵測點則會造成後續機器學習訓練時的運算負荷；另一種方法則是搭配 pymunk 的物理引擎模組使用 (圖.6.2)，使環境更符合實際物理現象，可加入碰撞、摩擦、力量大小、速度大小等可以個別設定和調用。當環境有了物理接下來就需要加入訓練所需的功能，如：訓練時的場景的即時影像畫面、即時獎勵回傳、該局結束時的場景重設和分數重置等功能。此時找到了 Open AI Gym 模組。

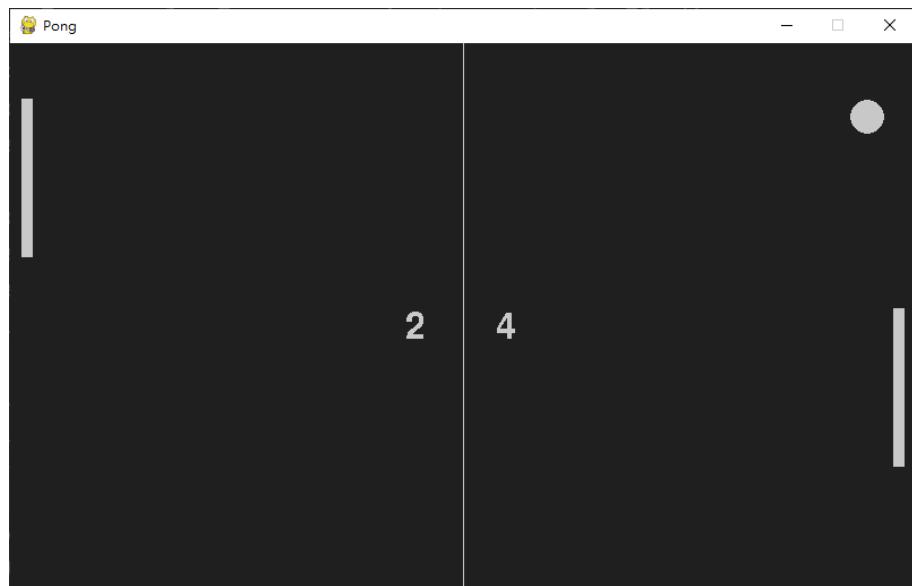


圖. 6.1: Pygame 模組編寫

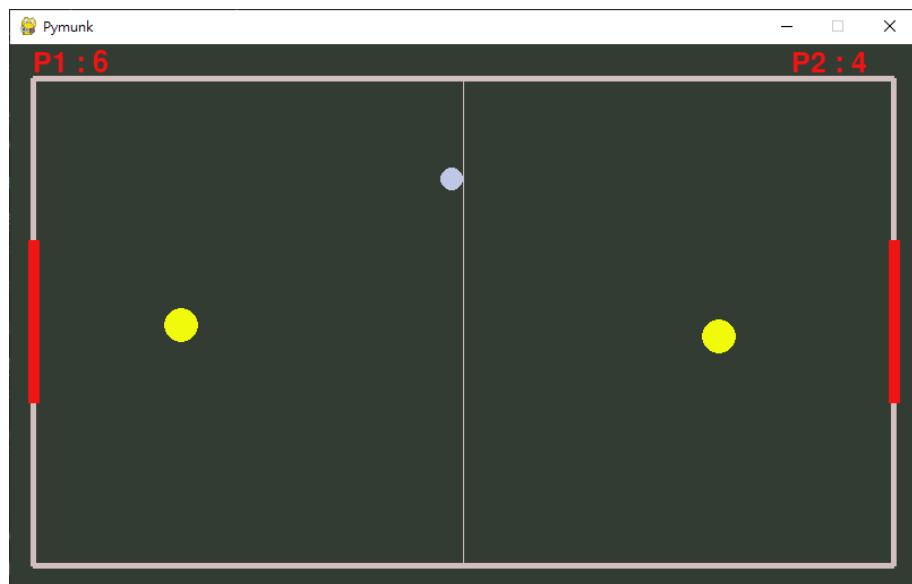


圖. 6.2: Pymunk 模組編寫

Open AI Gym 裡面有十幾種訓練模型的環境，提供機器學習做訓練的環境。由於我們的訓練模型是 pong game，在 Gym 模組裡面剛好有訓練模型，因此使用 Gym 模組相對於使用 Pygame 和 pymunk 的搭配來的方便，而且後續要在 CoppeliaSim 模擬環境模擬時也有套件可搭配使用，可簡化功能和訓練時所需的環境模型和訓練功能的程式編寫，另一方面自寫場景需要測試場景的漏洞，使用 Gym 可以節省檢查場景漏洞和修正的時間。

6.3 訓練程式的運作

由於機器學習和影像處理需要大量的運算矩陣運算，因此如果只單獨透過 Python 本身運算比編譯語言執行的速度來的慢，所以使用 Numpy 程式庫來解決在 Python 環境矩陣運算速度慢的問題，以提升訓練機器學習時的運算效率。pickle 是 Python 內部的序列化方式，主要是當機器學習訓練時可能因為一些原因需要暫時停止訓練，但為了讓已經停下的訓練再次重啟就需要透過 pickle 序列化的方式，將暫停前的訓練權重值透過 pickle 將其記錄下來，當訓練再次重啟時就可透過 pickle.load 讀取先前紀錄的 pickle 檔案就可回到當時暫停的狀態下繼續進行訓練。

機器學習所運用的架構是強化學習並搭配神經網路來訓練機器學習，結合了強化學習不需要事先收集訓練資料、不需要特別教導，以及神經網路的非線性激活函數的計算和參數的記憶性。

程式訓練流程 (圖.6.3)：

擷取影像，將影像裁剪至實際遊戲範圍，並簡化像素以利提高訓練時的計算速度，減少運算時的負擔，過濾顏色只保留球與擊錘，並把取到的影像二元化，取兩幀畫面進行比較，掌握球與擊錘間的相對位置 (畫面差)，透過前饋：計算球在環境的狀態及擊錘移動的決策，畫面差透過 W1 權重來計算球在環境的狀態，透過 W2 權重並經過啟動函數

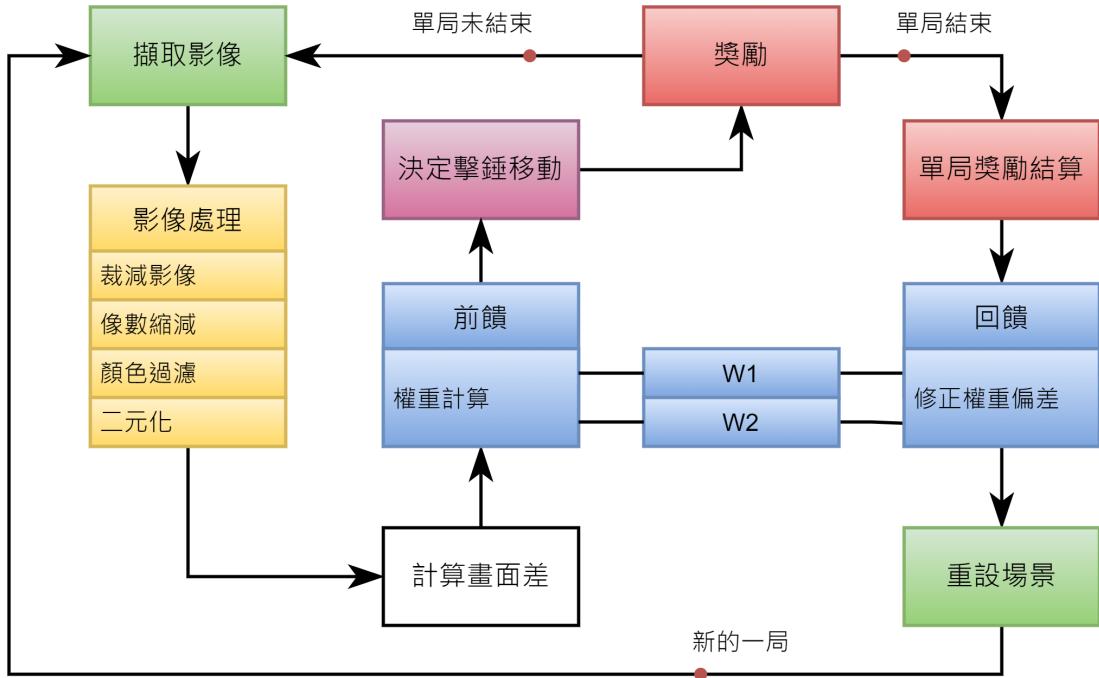


圖. 6.3: 強化學習訓練流程

(activation function) 得出擊錘移動的決策。透過產生隨機值的方式來與擊錘移動決策值進行比較，判定隨機值落在的區間來決定移動策略。計算 discount reward 及獎勵的加總。獎勵設定球若超過了對手，獎勵為 +1；如果錯過球，則獎勵為 -1；其餘狀態獎勵為 0。

在單局結束時，紀錄下該局累積下來的經驗，亦是紀錄該局所修正出來的參數而進行獎勵計算、log probability、RMSprop 優化率減因子和反饋 (back propagation)，當訓練次數到達指定次數會以 pickle 做紀錄，存下的數據可再次導入模型進行實際運用，或是當程式中斷後可重新匯入進行訓練。比較持續訓練與中斷後重新匯入訓練的差異，測試算法版本為 Pong2，MSE 代表均方誤差值，pong2_r 表示中斷後重新匯入的訓練數據，pong2 與 pong2_r 標示個別表示該訓練每局分數的紀錄，pong2 MSE 與 pong2_r MSE 標示個別表示該訓練均方誤差值紀錄 (圖.6.4)

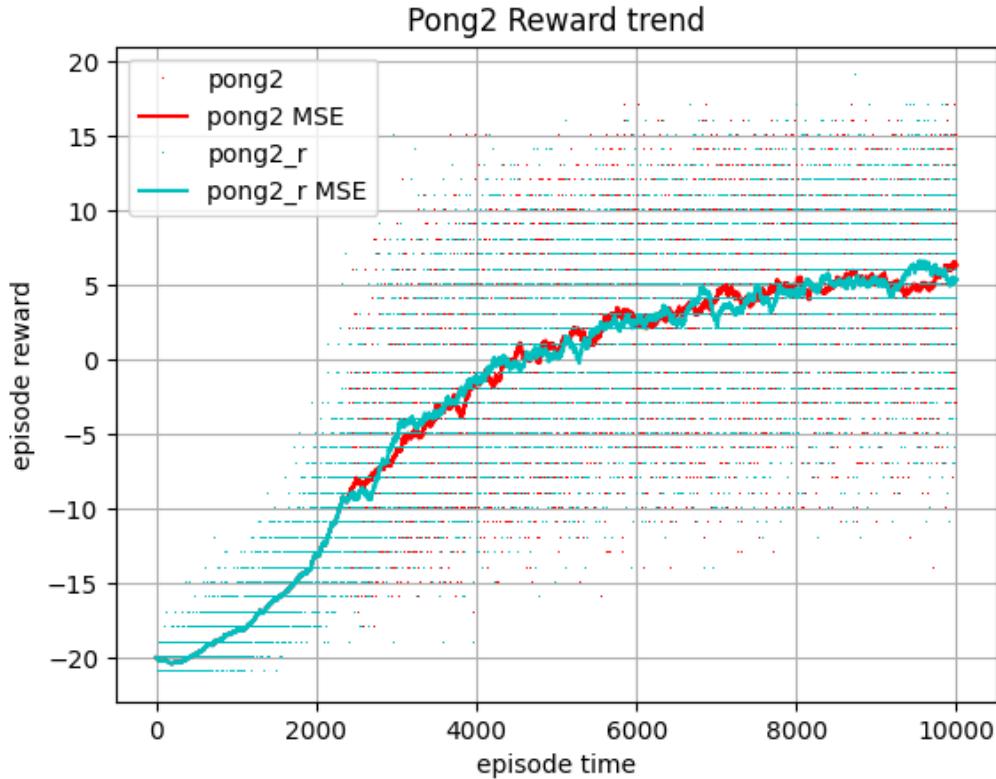


圖. 6.4: 比較中斷後訓練差異

6.4 訓練算法與參數比較

pong1 版本是 karpathy 的 pg pong.py 的原始碼 [14]，pong1.1 原始碼則是 schinger 的 pong_actor critic/pg pong ac.py[15]，pong1.2 的是修改 pong1 的學習率參數從 0.001 調整到 0.0001，並且將裁切後 80*80 的影像改成 75*80。pong2 版本是 etienne87 的 pg pong.py 的原始碼 [16] 修改的，將動作策略由上下上替換成停上下的模式，學習率也配合調整，由 0.0001 調至 0.001。

下圖 (圖.6.4) 為這幾個版本的訓練趨勢的紀錄提供做比較，訓練次數 (圖.6.4之水平軸) 為 3000 局做比較，小點為每局積分總和 (圖.6.4之垂直軸)，線條為累積積分的均方誤差值，積分計算 -21 分代表對面得 21 分，即機器訓練所得分數-對面所得分數。訓練存在隨機性 (Stochastic)，因此每次訓練所得趨勢會有所差異。

版本名稱	標示顏色	啟動函數	參考來源
pong1	紅色	sigmoid	[14]
pong1.1	藍色	sigmoid	[15]
pong1.2	橘色	sigmoid	[14]
pong2	綠色	softmax	[16]

表. 6.1: 版本標示

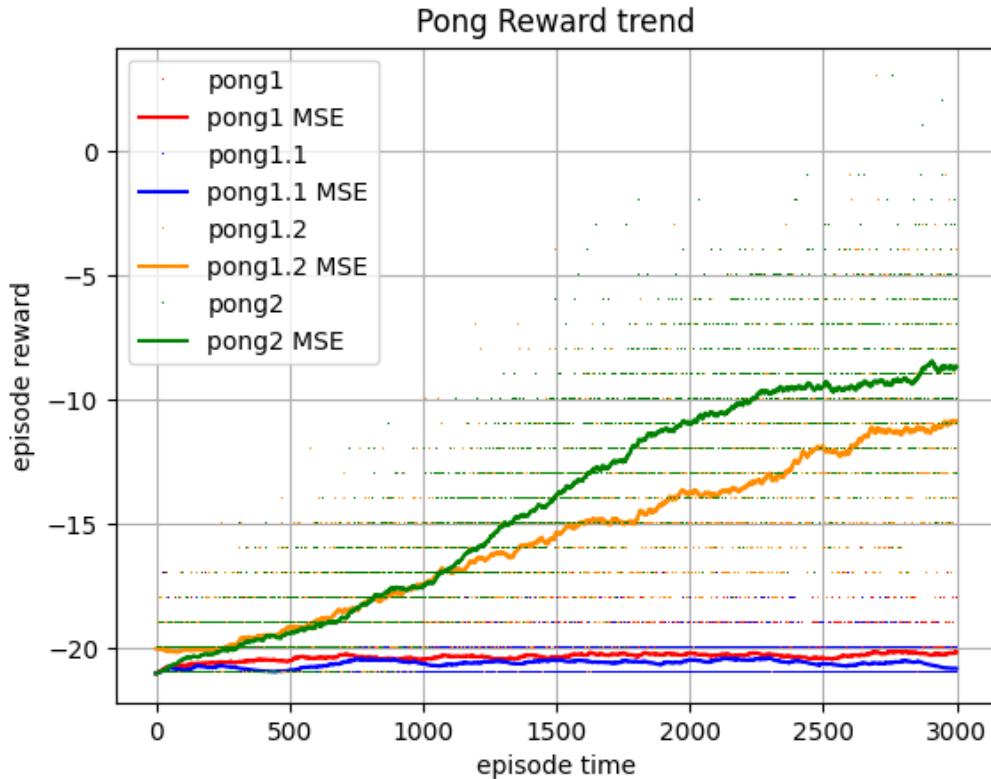


圖. 6.5: 各版本差異

觀察後發現，pong2 版本的訓練結果得分最高，pong1.2 的訓練結果為次高，pong1 及 pong1.1 的版本沒有收斂，訓練對打的表現沒有進步的跡象，參數還需做調整及測試，pong2 及 pong1.2 為主要選擇訓練演算法。接下來測試 pong1 的學習率調整成 0.0001，pong1.2 的學習率上調到 0.001，以 pong2 版本當作對照（圖.6.4），同樣以 3000 次訓練當作參考依據，測試結果三者相當接近。因為訓練時間及設備限制，修正後的 pong1 與 pong1.2 版本所以擇一與 pong2 進行長時間訓練比較，因此選用 pong1.2 版本。

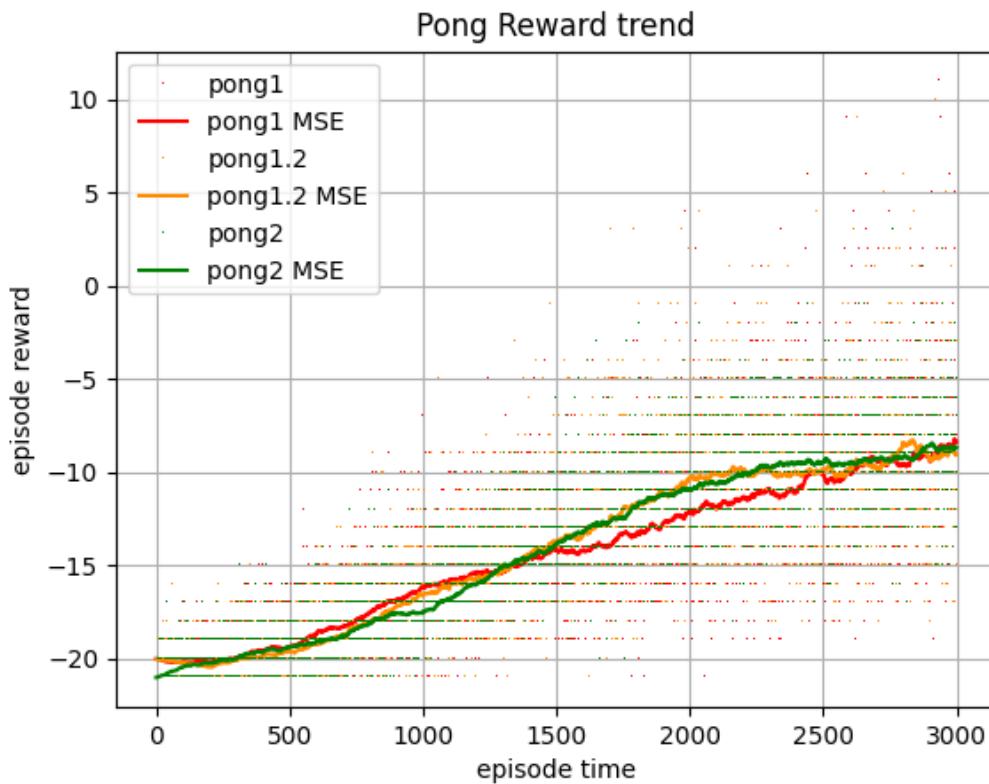


圖. 6.6: pong1 與 1.2 調整後數據比較

電腦規格:

1. 研究室桌上型電腦(長時間訓練所使用):

Windows10 專業版 2004

CPU : Intel i7-7700 3.60GHZ

RAM : 16GB

GPU : NVDIA GTX1050

2. 個人筆記型電腦(3000 次訓練數據為此設備訓練，短時間測試用):

Windows10 家用版 2004

CPU : Intel i7-8750H 2.20GHZ

RAM : 8GB

GPU : NVDIA GTX1050 Ti

總訓練時數約 336~350 小時(圖.6.7)，pong2 的數據均方誤差(MSE)數

據較穩定，由於訓練中無法得知當下是處於數據波動的較優或較差的狀態，若要中斷訓練則有可能剛好處於波動的較差的狀態，因此選擇 pong2 的版本當作訓練算法較為穩定。

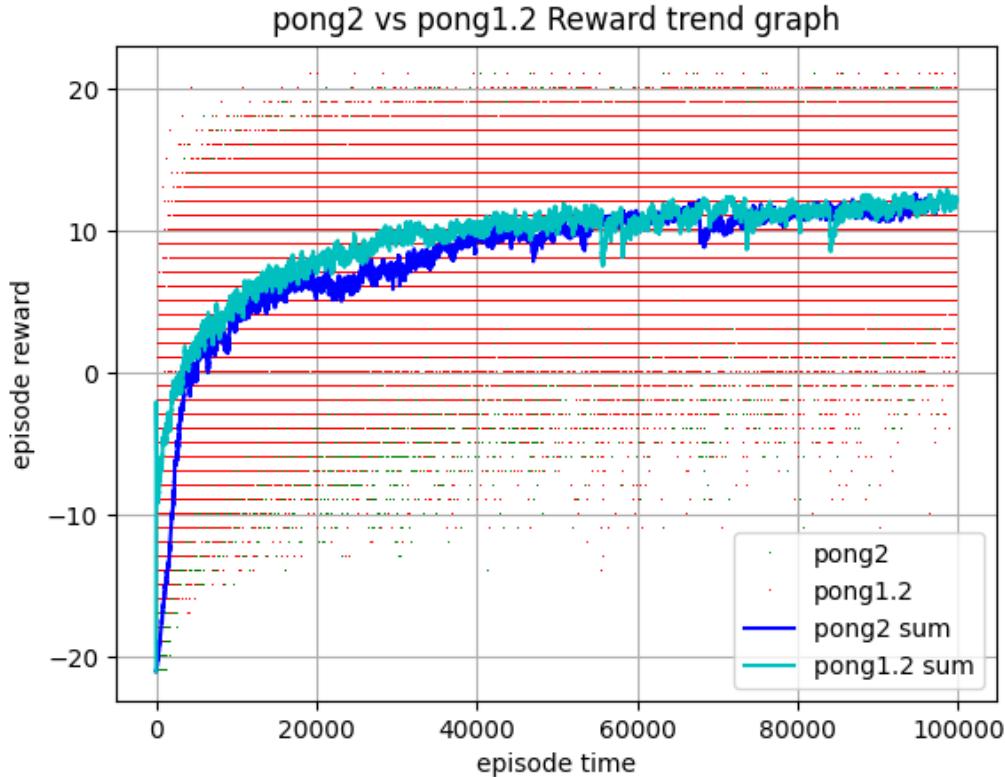


圖. 6.7: pong2 與 pong1.2 長時間訓練數據

6.5 CoppeliaSim RemoteAPI

在進行強化學習時主要是透過 CoppeliaSim 中的 Remote API 函數來取得模擬場景中所需要的資訊，並在進行訓練後再回傳到 CoppeliaSim 做控制的動作。

6.5.1 Remote API 模組及動態連結函示庫

在啟用 RemoteAPI 需要先準備以下三項模組和動態連結函示庫，並將此三項與預執行的程式放在同一目錄下：

- sim.py

- simConst.py
- remoteApi.dll、remoteApi.dylib 或 remoteApi.so (依序適用於:Windows、MacOS、Ubuntu)

sim.py 及 simConst.py 為 Python 模組，其位於:

CoppeliaSim 安裝目錄 \programming\remoteApiBindings\python\python
remoteApi.dll 為 RemoteAPI 動態連結函示庫，其位於:

CoppeliaSim 安裝目錄 \programming\remoteApiBindings\lib\lib\ 作業系統

6.5.2 Remote API 埠使用

Remote API 是通過通訊埠取得環境資訊，在 CoppeliaSim 中預設埠號為 19997，只有預設埠不需開啟特定場景就可進行通訊並控制所有功能，且在大量影像資料處理時可啟用多埠，使其中一個通訊埠用於影像處理，另一個則用於控制，新增的通訊埠需在安裝資料夾中的 remoteApiConnections.txt 加入需要的埠號。

6.6 Open AI Gym 自定義環境

Open AI Gym 可以支持我們以自己搭建的環境進行訓練，因此我們透過 Gym 並以 CoppeliaSim 虛擬環境中搭建的冰球機模型來完成訓練，而 Gym 所使用的環境參數就是由前述的 Remote API 來取得。Gym 將環境抽象為一個類別 (class) 在該類別 (class) 中需分別定義以下參數來達成自定義環境的訓練。

1. init：初始化 CoppeliaSim 中的環境參數。
2. seed：用於設置環境變數。
3. make observation：用於設置環境場景中需觀察的值，如：擊錘位置、冰球位置... 等。

4. make action：設置擊錘的移動速度。
5. step：訓練的主要邏輯，如：遊戲是否結束、reward 函數返回值、環境觀察值... 等。
6. reset：將環境重置。
7. render：可搭配 OpenCV 進行數據渲染。
8. close：釋放環境數據。

6.7 總結

選擇適合的演算法，並找到適合的參數，透過訓練提升機器對打的能力，訓練的時間越長，學習對打的成效越好，但訓練後期進步的幅度趨緩，因此得評估訓練的時間長度以符合整體效益。在模擬環境中利用 RemoteAPI 建置了跨平台控制，並運用 CoppeliaSim 中視覺傳感器所取得之影像透過 OpenCV 加以處理便於輔助玩家進行對打，在程式中則使用 OpenCV 處理及簡化過的影像進行擊錘移動的計算，將計算後得出的結果回傳到 CoppeliaSim 中的環境，使擊錘做出對應的動作。

第七章 未來研究建議

本專題已建立 2D 環境的算法與訓練數據，並將實體系統導入虛擬環境，建立跨平台控制 (RemoteAPI) 及輔助對打系統，後續可透過建立虛擬環境的訓練程式進行虛擬訓練，完成虛擬訓練後可導入實體機電系統，將對打系統實體化，架設伺服器提供網際介面，提供網際控制、即時觀看對打影像等功能。

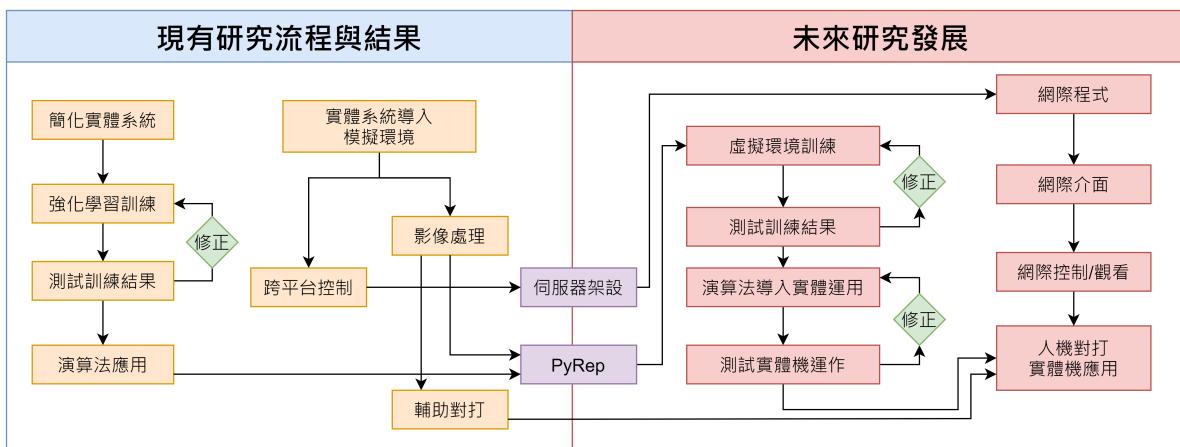


圖. 7.1: 現有研究流程與未來展望

第八章 問題與討論

Q : gym 用到的 atari 動態連結庫在讀取目錄下但在執行的時候出現缺少 ale_c.cp38-win_amd64.dll

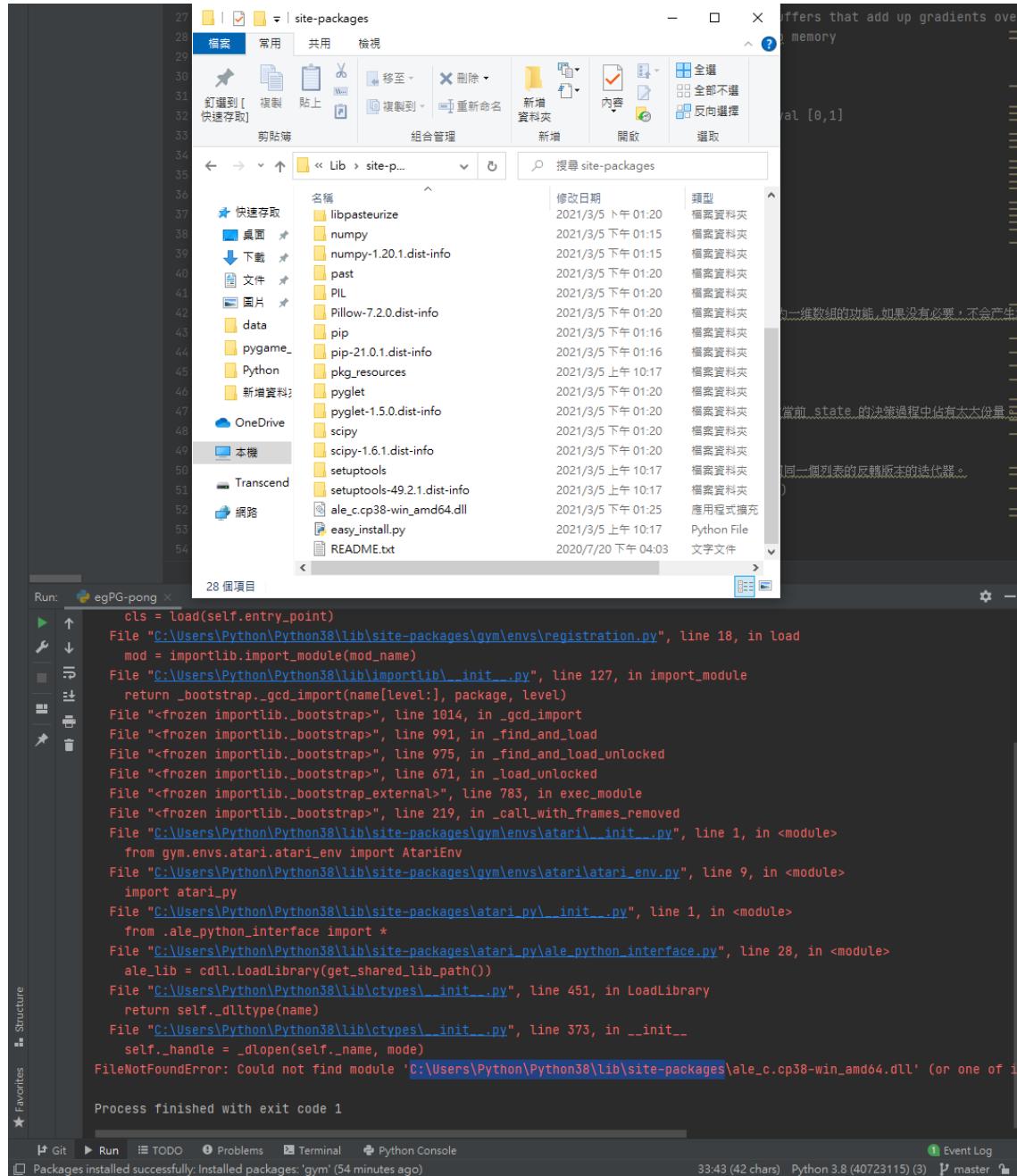


圖. 8.1: 動態連結庫錯誤

A：此問題尚未找到解決方法。

Q：錄製訓練過程的程式讀不到 ffmpeg(圖.8.2)。

```
"D:\Program Files\python3.8.2\python.exe" Y:/tmp3/users/pong1.2.py
Traceback (most recent call last):
  File "Y:/tmp3/users/pong1.2.py", line 81, in <module>
    observation = env.reset()
  File "D:\Program Files\python3.8.2\lib\site-packages\gym\wrappers\monitor.py", line 39, in reset
    self._after_reset(observation)
  File "D:\Program Files\python3.8.2\lib\site-packages\gym\wrappers\monitor.py", line 185, in _after_reset
    self.reset_video_recorder()
  File "D:\Program Files\python3.8.2\lib\site-packages\gym\wrappers\monitor.py", line 206, in reset_video_recorder
    self.video_recorder.capture_frame()
  File "D:\Program Files\python3.8.2\lib\site-packages\gym\wrappers\monitoring\video_recorder.py", line 116, in capture_frame
    self._encode_image_frame(frame)
  File "D:\Program Files\python3.8.2\lib\site-packages\gym\wrappers\monitoring\video_recorder.py", line 162, in _encode_image_frame
    self.encoder = ImageEncoder(self.path, frame.shape, self.frames_per_sec, self.output_frames_per_sec)
  File "D:\Program Files\python3.8.2\lib\site-packages\gym\wrappers\monitoring\video_recorder.py", line 255, in __init__
    raise error.DependencyNotInstalled("""Found neither the ffmpeg nor avconv executables. On OS X, you can install ffmpeg via 'brew install ffmpeg'. On most Ubuntu
gym.error.DependencyNotInstalled: Found neither the ffmpeg nor avconv executables. On OS X, you can install ffmpeg via 'brew install ffmpeg'. On most Ubuntu vari
```

圖. 8.2: 程式讀不到 ffmpeg

A：需要在作業系統中安裝 ffmpeg：

1. 下載、解壓縮

先到官網 <https://ffmpeg.org/download.html> 下載 "Windows builds from gyan.dev"，下載 <https://www.gyan.dev/ffmpeg/builds/ffmpeg-git-full.7z>，解壓縮重新命名成"ffmpeg"並放到 C 槽目錄下 (C:\ffmpeg)。

2. 環境設定 (windows10 20H2 及 2004 版本)

開啟"設定"→"系統"→"左方"關於"選項→右側"進階系統設定"→"環境變數"(圖.8.3)→選取"Path"，編輯(圖.8.4)→"新增"，增加一個環境變數，給定內容為："C:\ffmpeg\bin"，"確定"(圖.8.5)→"確定"→"確定



圖. 8.3: 進階系統設定

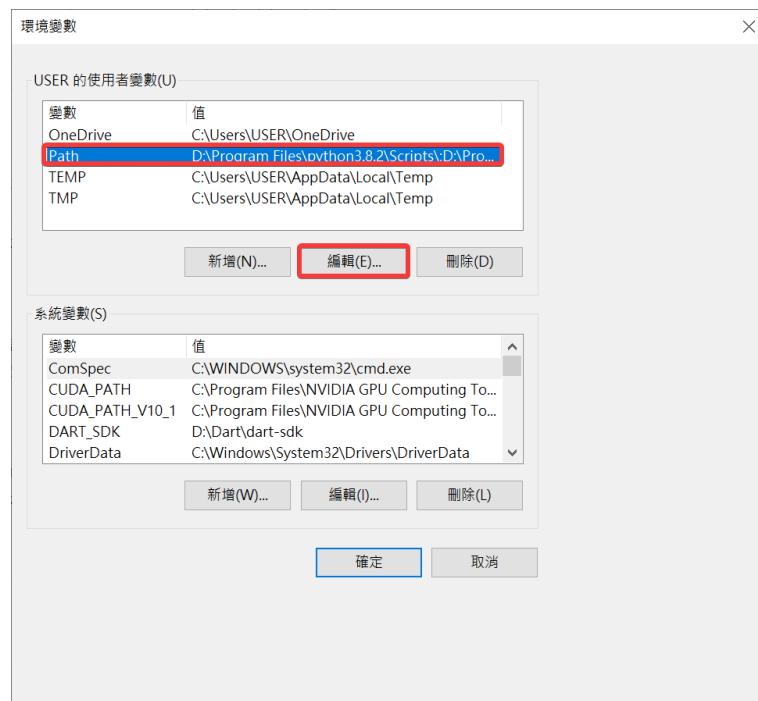


圖. 8.4: 環境變數

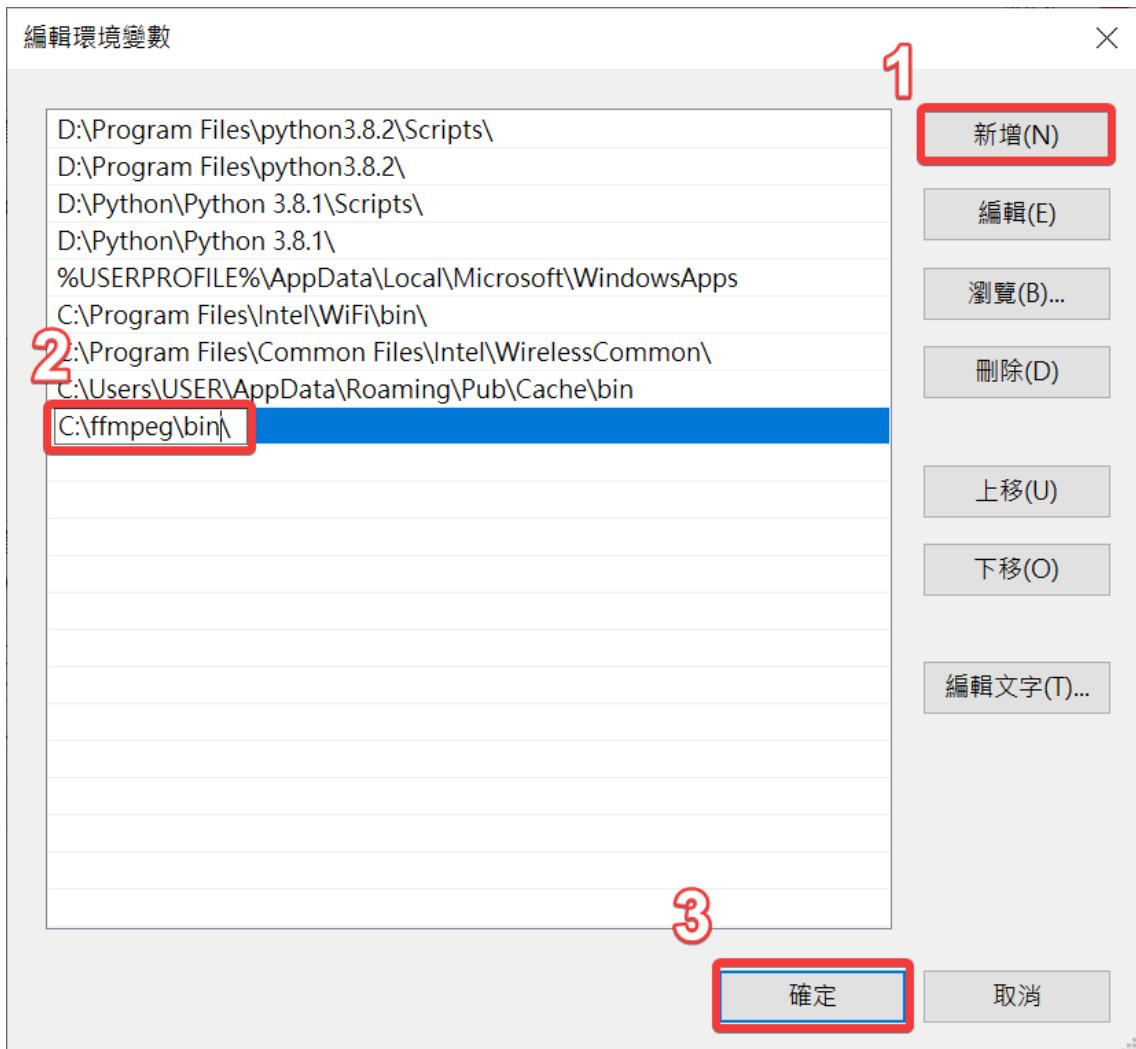


圖. 8.5: 編輯環境變數

- 測試

開啟命令字元 (win+R，輸入"cmd")，執行"ffmpeg"(圖.8.6)

```
C:\Users\USER>ffmpeg
ffmpeg version 2021-03-16-git-9383885c0d-full_build-www.gyan.dev Copyright (c) 2000-2021 the FFmpeg developers
  built with gcc 10.2.0 (Rev6, Built by MSYS2 project)
configuration: --enable-gpl --enable-version3 --enable-static --disable-w32threads --disable-autodetect --enable-fontconfig --enable-iconv --enable-gnutls --enable-libxml2 --enable-gmp --enable-lzma --enable-libsnapy --enable-zlib --enable-libfribidi --enable-libsrtp --enable-libssh --enable-libzmq --enable-avsynth --enable-libbluray --enable-libcaca --enable-sdl2 --enable-libdavid --enable-libzvbi --enable-libravle --enable-libsyntax --enable-libwebp --enable-libx264 --enable-libx265 --enable-libvid --enable-libaom --enable-libopenjpeg --enable-libvpx --enable-libass --enable-frei0r --enable-libfreetype --enable-libvidstab --enable-libvmaf --enable-libzimg --enable-amf --enable-cuda-llm --enable-cuvid --enable-ffnvcodec --enable-nvdec --enable-nvenc --enable-d3d11va --enable-dxva2 --enable-libmfx --enable-libglslang --enable-vulkan --enable-opengl --enable-libcdio --enable-libgme --enable-libmodplug --enable-libopenmpt --enable-libopencore-amrwb --enable-libmp3lame --enable-libshine --enable-libtheora --enable-libtwolame --enable-libvo-amrwbenc --enable-libilbc --enable-libgsm --enable-libopencore-amrnb --enable-libopus --enable-libspeex --enable-libvorbis --enable-ladspa --enable-libbs2b --enable-libflite --enable-libmysofa --enable-librubberband --enable-libsoxr --enable-chromaprint
libavutil      56. 68.100 / 56. 68.100
libavcodec     58.132.100 / 58.132.100
libavformat    58. 74.100 / 58. 74.100
libavdevice    58. 12.100 / 58. 12.100
libavfilter     7.109.100 / 7.109.100
libswscale      5.  8.100 / 5.  8.100
libswresample   3.  8.100 / 3.  8.100
libpostproc    55.  8.100 / 55.  8.100
Hyper fast Audio and Video encoder
usage: ffmpeg [options] [[infile options] -i infile]... {[outfile options] outfile}...
Use -h to get full help or, even better, run 'man ffmpeg'
C:\Users\USER>
```

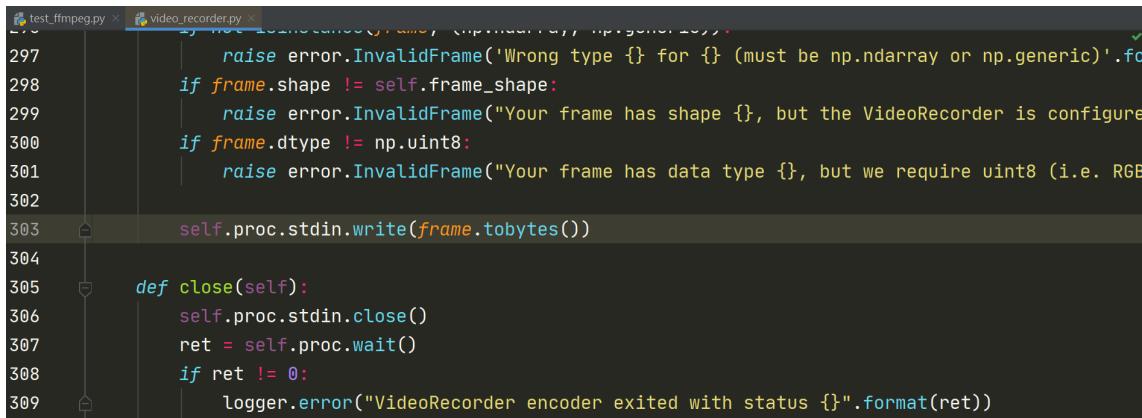
圖. 8.6: ffmpeg 成功執行

Q：運用 gym.wrappers.Monitor 透過 ffmpeg 進行錄影，紀錄下訓練影像。但記錄後的影像資料皆為 1KB，並且無法開啟。(圖.8.7)



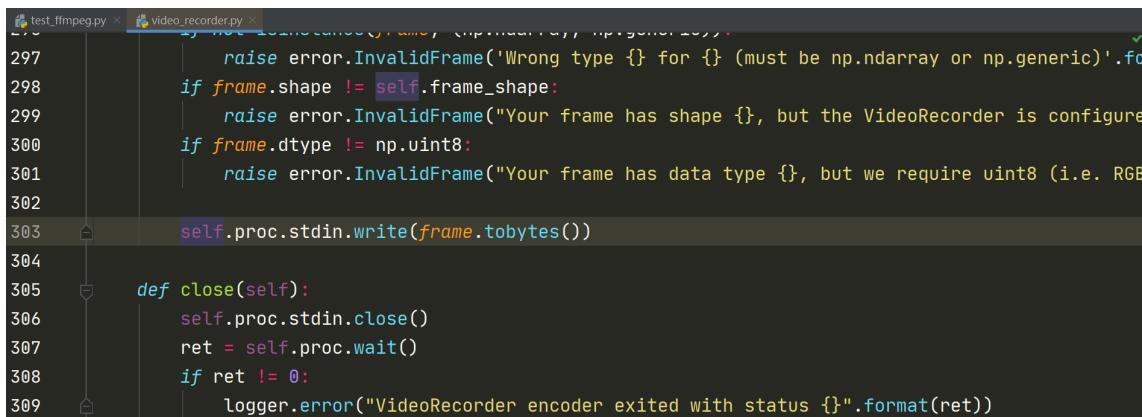
圖. 8.7: 錄製後，影片無法開啟

A：修改 gym.wrappers.Monitor 的 video_recorder.py 的設定 (圖.8.8)，將 303 行的縮排修正 (從 if 階層上移到 def 的階層) 即可 (圖.)。



```
297     raise error.InvalidFrame('Wrong type {} for {} (must be np.ndarray or np.generic)'.format(type(frame), frame))
298     if frame.shape != self.frame_shape:
299         raise error.InvalidFrame("Your frame has shape {}, but the VideoRecorder is configured to expect shape {}".format(frame.shape, self.frame_shape))
300     if frame.dtype != np.uint8:
301         raise error.InvalidFrame("Your frame has data type {}, but we require uint8 (i.e. RGB) type".format(frame.dtype))
302
303     self.proc.stdin.write(frame.tobytes())
304
305     def close(self):
306         self.proc.stdin.close()
307         ret = self.proc.wait()
308         if ret != 0:
309             logger.error("VideoRecorder encoder exited with status {}".format(ret))
```

圖. 8.8: 原始設定



```
297     raise error.InvalidFrame('Wrong type {} for {} (must be np.ndarray or np.generic)'.format(type(frame), frame))
298     if frame.shape != self.frame_shape:
299         raise error.InvalidFrame("Your frame has shape {}, but the VideoRecorder is configured to expect shape {}".format(frame.shape, self.frame_shape))
300     if frame.dtype != np.uint8:
301         raise error.InvalidFrame("Your frame has data type {}, but we require uint8 (i.e. RGB) type".format(frame.dtype))
302
303     self.proc.stdin.write(frame.tobytes())
304
305     def close(self):
306         self.proc.stdin.close()
307         ret = self.proc.wait()
308         if ret != 0:
309             logger.error("VideoRecorder encoder exited with status {}".format(ret))
```

圖. 8.9: 修正後設定

Q：啟用 cmsimde 的 MathJax 的功能遇到文章使用括號補充說明的內容被誤當成 latex 的語法轉換。

A：格式轉換原始定義成"(" 和")"，所以出現誤換的問題。

程式. 8.1: MathJax 程式碼

```
<script>
  MathJax = {
    tex: {inlineMath: [[ '$', '$' ], [ '\\(', '\\)' ]]}
  };
</script>
<script id="MathJax-script"
  async src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js">
</script>
```

修正後將"(" 和")" 換成"\$"，就解決誤換問題

參考文獻

- [1] <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>
- [2] <https://towardsdatascience.com/derivative-of-the-sigmoid-function-536880cf918e>
- [3] <http://www.incompleteideas.net/book/RLbook2020.pdf>
- [4] <https://medium.com/change-the-world-with-technology/policy-gradient-181d43a24cf5>
- [5] <https://livebook.manning.com/book/grokking-deep-reinforcement-learning/chapter-11/v-11/38>
- [6] <http://ukko.life.nctu.edu.tw/u0517047/usage.html>
- [7] <https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html>
- [8] <https://uupgrade.medium.com/python-那些年我們一起玩過的遊戲-三-打磚塊-d89b648896ca>
- [9] <https://cvfiasd.pixnet.net/blog/post/275774124-深度學習激勵函數介紹>
- [10] <https://www.coppeliarobotics.com/helpFiles/>
- [11] <https://hackernoon.com/the-reason-behind-moving-in-the-direction-opposite-to-the-gradient-f9566b95370b>
- [12] <https://ruder.io/optimizing-gradient-descent/>
- [13] https://zh.wikipedia.org/wiki/HSL_和_HSV_色彩空間

[14] <https://gist.github.com/karpathy/a4166c7fe253700972fcbc77e4ea32c5#file-pg-pong-py>

[15] https://github.com/schinger/pong_actor-critic/blob/master/pg-pong-ac.py

[16] <https://gist.github.com/etienne87/6803a65653975114e6c6f08bb25e1522>

附錄

LaTeX

LaTeX 為一種程式語言，支援標準庫 (Standard Libraries) 和外部程式庫 (External Libraries)，不過與一般程式語言不同的是，它可以直接表述 Tex 排版結構，類似於 PHP 之於 HTML 的概念。但是直接撰寫 LaTeX 仍較複雜，因此可以藉由 Markdown 這種輕量的標註式語言先行完成文章，再交由 LaTeX 排版。此專題報告採用編輯軟體為 LaTeX，綜合對比 Word 編輯方法，LaTeX 較為精準正確、更改、製作公式等，以便符合規範、製作。

表. 1: 文字編輯軟體比較表

	相容性	直觀性	文件排版	數學公式	微調細部
LaTeX	√		√	√	√
Word		√			√

- 特點:

1. 相容性：以 Word 為例會有版本差異，使用較高版本編輯的文件可能無法以較低的版本開啟，且不同作業系統也有些許差異；相比 LaTeX 可以利用不同編譯器進行編譯，且為免費軟體也可移植至可攜系統內，可以搭配 Github 協同編譯。
2. 文件排版：許多規範都會要求使用特定版型，使用文字編譯環境較能準確符合規定之版型，且能夠大範圍的自定義排定所需格式，並能不受之後更改而整體格式變形。
3. 數學公式呈現：LaTeX 可以直接利用本身多元的模組套件加入、編輯數學公式，在數學推導過程能夠快速的輸入自己需要的內容即可。
4. 細部調整：在大型論文、報告中有多項文字、圖片、表格，需要調

整細部時，要在好幾頁中找尋，而 LaTeX 可以分段章節進行編譯，再進行合併處理大章節。

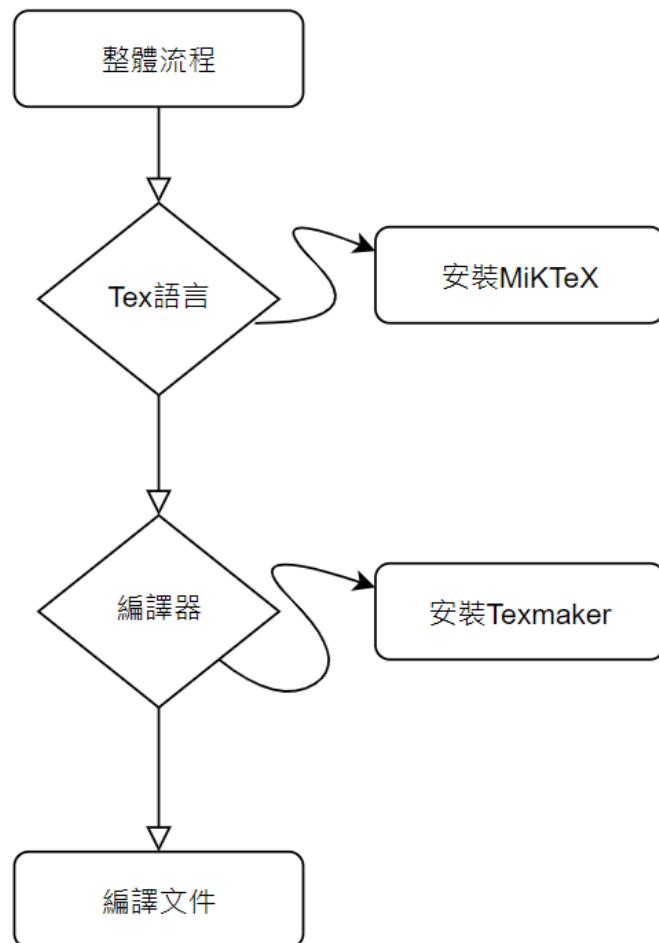


圖. 10: 編譯流程

FFmpeg

FFmpeg 是一個開放原始碼的自由軟體，可以對音訊和視訊進行多種格式的錄影、轉檔、串流功能。在專題訓練過程中透過 FFmpeg 的視訊錄製的功能記錄對打影像來了解實際訓練狀況。