2021 年臺灣國際科學展覽會
優勝作品專輯

作品編號　**190037**

參展科別　電腦科學與資訊工程

作品名稱　利用近端策略優化演算法結合內在好奇心模組進行 2D 雙足模型行走模擬

得獎獎項　大會獎 一等獎

就讀學校　雲林縣立古坑華德福實驗高級中學

指導教師　駱巍元

作者姓名　黃守榕

關鍵詞　**Deep Reinforcement Learning、Proximal Policy Optimization、Intrinsic Curiosity Module**

作者簡介

黃守榕在國中時開始參加科學競賽，三年的競賽過程中有成有敗，高一時開始自學程式語言，在有了一定的程式能力後開始自學機器學習，並對機器學習的分支'強化學習'深感興趣，想將其應用於機器人控制中，發現該領域還不成熟，便決定實作了一個入門的強化學習專題來使 2d 雙足模型學會行走。

摘要

強化學習為當前 AI 領域的熱門話題，其特點是在環境的獎勵與懲罰下，進行學習。強化學習雖然較為困難，但其成功的項目都非常有名，其中最著名的例子有: AlphoGo、AlphaZero 等等。

深度強化學習(DRL)是深度學習與強化學習的結合體，本專題透過 DRL 實現近端優化策略演算法，來使 BipedalWalker 環境中的二足模型學會行走，並調適超參數與神經網路來讓模型訓練擁有更好的結果。

經過實驗後發現，適當的降低獎勵折扣衰減率能有效的提升學習速度以及學習上限，同時可以避免分數落差過大導致的 Dead relu 問題。最終的結果能讓平均分數達到 302 分，成功達成了 BipedalWalker 環境要求(平均分數>=300 分)。

為了使智能體擁有更好的探索能力，本專題加入了 ICM(Intrinsic Curiosity Module)，成功提升了最終的平均分數至 316 分，將不摔倒的機率提升至 99%，最高分數則到了 320 分，使得雙足模型能以更快的速度向前移動並保持穩定。

Abstract

Reinforcement learning is a hot topic in the current AI space, which is characterized by learning in an environment that rewards and punishes it. Although reinforcement learning is difficult, its successes are well known, the most famous examples being AlphoGo, AlphaZero and so on.

Deep Reinforcement Learning (DRL) is a combination of deep learning and reinforcement learning. This topic uses DRL to implement a proximal policy optimization algorithm to make a bipedal model in the BipedalWalker environment learn to walk, and adapts hyperparameters and neural networks architecture to make the model training have better results.

After the experiments, it is found that the appropriate reduction of the reward discount rate can increase the learning speed and the learning limit, and at the same time avoid the dead relu problem caused by the large differences in scores. The final result is that the average score can reach 302, which successfully solves the requirements of BipedalWalker environment (average score >= 300).

The ICM (Intrinsic Curiosity Module) was added to improve the exploration of the agent. The final average score was raised to 316, which increased the probability of not falling to 99% and the highest score to 320, allowing the bipedal model to move forward and remain stable at a much faster rate.

一、前言

(一)、研究動機:

一直以來二足機器人的行走都是個難題，其中最困難的問題在於要保持平衡及適應複雜的環境，我希望能利用機器學習的一個分支'強化學習(Reinforcement learning)'，讓機器人自己學習如何克服環境，學會行走。強化學習基於環境而行動，學習如何獲得最大利益，而這正是符合機器人要與環境互動的方法。演算法使用的是 PPO(Proximal Policy Optimization)演算法(由 OpenAI 提出)，該演算法是一種使用 Actor-Critic 架構的近似 on-policy 演算法，其特點是能夠對同一批樣本重複訓練多次，並且可以利用常態分佈解決連續性問題。基於這些特點，我嘗試使用此演算法配合深度神經網路來讓一個二足機器人模型智能體在一個 OpenAI-gym 中的 BipedalWalker 環境進行學習，使其學會行走。

(二)、背景介紹:

強化學習起源於行為主義心理學，主要使用在控制類問題，透過智能體(agent)與環境交互的過程進行學習，當 agent 每使用策略採取一次行動，就會獲得一個數值獎勵，該獎勵表示行動的好壞。強化學習透過在環境探索中不斷的試錯(試錯法)來學習，agent 會綜合過去的經驗及未知的探索來不斷優化它的策略，agent 的目標是透過學習優化策略使策略能夠選取一連串的動作以最大化累積獎勵。

在深度學習出現以後就不斷的有人想將其與強化學習做結合，但真正成功的開端是 DeepMind 在 NIPS 2013 上發表的 Playing Atari with Deep Reinforcement Learning 一文。DeepMind 使用深度學習直接輸入高維狀態(state)來提取特徵，再利用強化學習演算法學習策略，成功將深度學習與強化學習結合，其成果超過了人類專家的水平。

(三)、目的:

1、利用強化學習使二足模型能夠學習行走，並達到一定的移動速度。

2、利用 Tensorflow 框架自行搭建 PPO 演算法，並將演算法適度修改以適應二足模型環境。

3、調適演算法超參數，使學習速度及穩定度增加。

4、加入 ICM(Intrinsic Curiosity Module, 內在好奇心模組)提升探索能力。

二、研究方法或過程

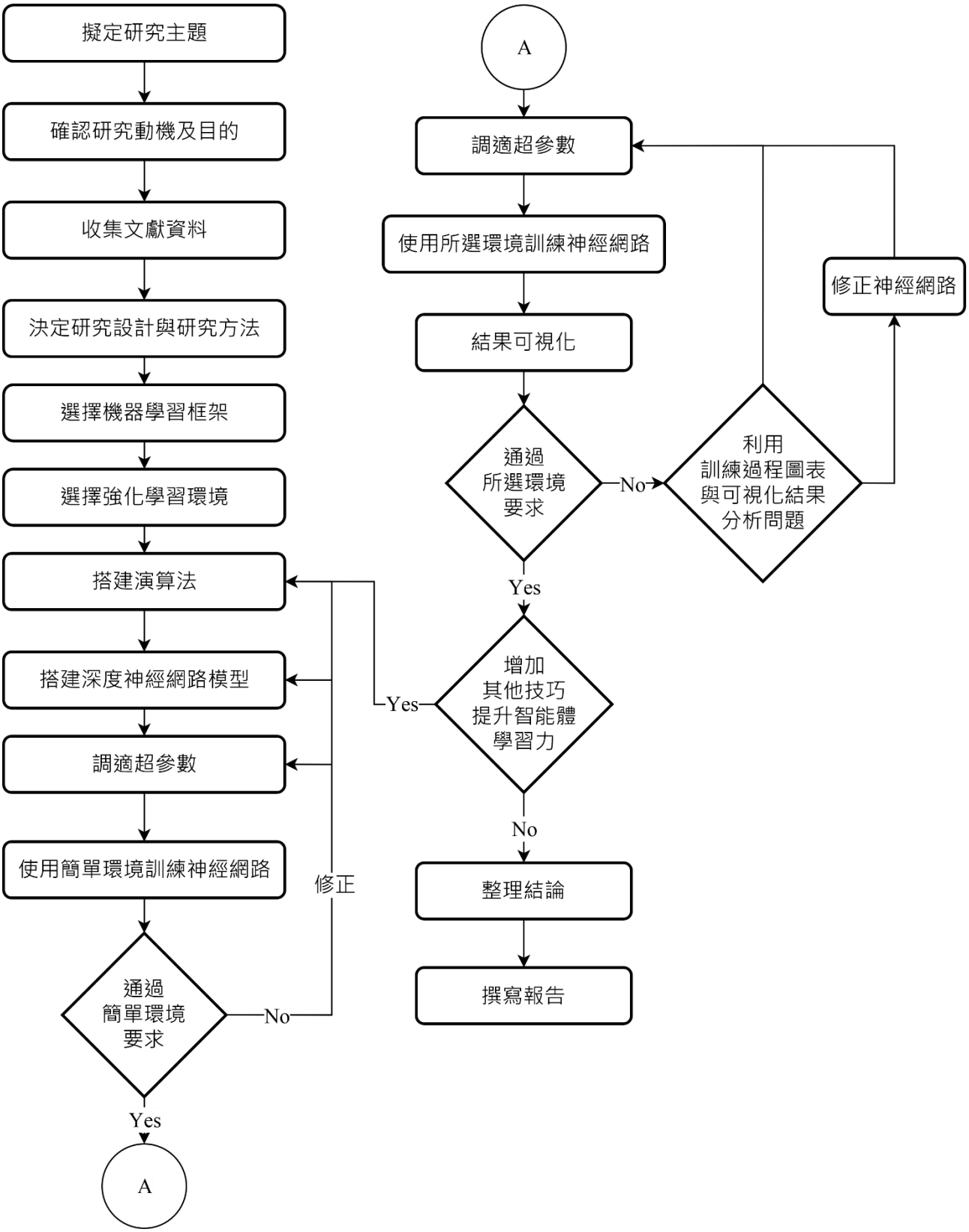


圖 1:研究流程圖

硬體環境:

記憶體容量:32gb

CPU:Intel Core i5-7500　3.4GHz

因 GPU 廠牌為 AMD，無法用 GPU 進行加速，訓練過程中也沒有運用到多線程或多進程

使用軟體:

系統:windows

程式語言:python

深度學習框架:tensorflow

圖表生成:tensorboard、matplotlib

其餘擴充程式庫:numpy

(一)、環境介紹:

BipedalWalker 環境由 OpenAI 組織提供，此環境有屬於自己的 API 功能，以協助使用者測試他們的演算法。整個環境由一個 2D 雙足機器人與地形所組成，目標是使用演算法來訓練機器人學會行走。

環境會輸出機器人的狀態(state)，智能體接收後輸出關節的動作(action)，環境接收動作之後輸出上一步動作所獲得的獎勵(reward)以及下一個狀態，智能體接收獎勵分數進行訓練，反覆循環。

1、States:

由 24 個值所組成，包括本體及各關節水平速度、垂直速度及角速度以及 10 個雷達進行測距，狀態中並沒有座標。

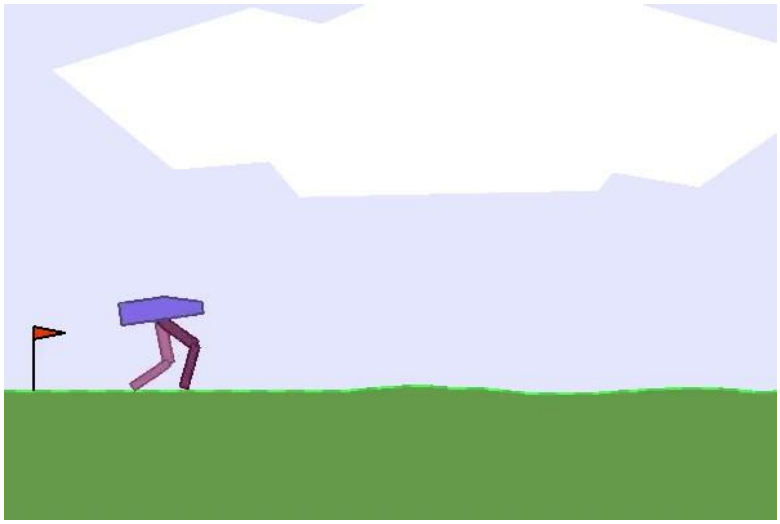


圖 2:BipedalWalker 環境

Num	Observation	Min	Max	Mean
0	hull_angle	0	2*pi	0.5
1	hull_angularVelocity	-inf	+inf	-
2	vel_x	-1	+1	-
3	vel_y	-1	+1	-
4	hip_joint_1_angle	-inf	+inf	-
5	hip_joint_1_speed	-inf	+inf	-
6	knee_joint_1_angle	-inf	+inf	-
7	knee_joint_1_speed	-inf	+inf	-
8	leg_1_ground_contact_flag	0	1	-
9	hip_joint_2_angle	-inf	+inf	-
10	hip_joint_2_speed	-inf	+inf	-
11	knee_joint_2_angle	-inf	+inf	-
12	knee_joint_2_speed	-inf	+inf	-
13	leg_2_ground_contact_flag	0	1	-
14-23	10 lidar readings	-inf	+inf	-

圖 3:24 個 states

圖片來源:https://github.com/openai/gym/wiki/BipedalWalker-v2

2、Actions:

由 4 個值組成，分別對應 4 個關節，單位為(扭力/速度)。

Num	Name	Min	Max
0	Hip_1 (Torque / Velocity)	-1	+1
1	Knee_1 (Torque / Velocity)	-1	+1
2	Hip_2 (Torque / Velocity)	-1	+1
3	Knee_2 (Torque / Velocity)	-1	+1

圖 4:4 個 actions

圖片來源:https://github.com/openai/gym/wiki/BipedalWalker-v2

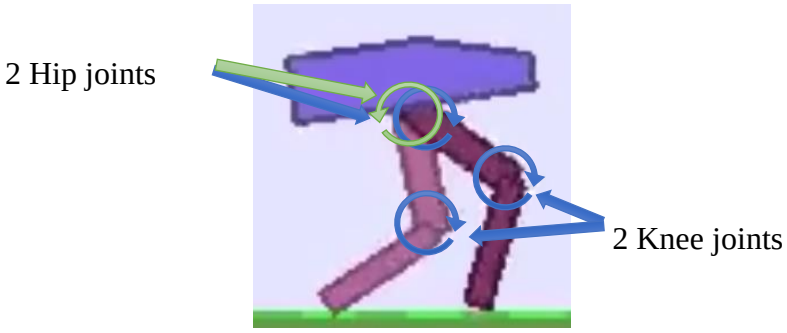


圖 5:BipedalWalker joints

3、Rewards:

向前移動可以獲得分數，分數由負到正(移動速度太慢，向後移動，或是原地不動時分數為負數)，一個 episode 滿分約為 300 分，機器人摔倒(身體接觸到地面)，分數為-100，機器人關節扭力會消耗少量分數。

4、Episode Termination:

機器人摔倒，抵達環境終點，或是經過 1600 步(一步[state → action → reward] → next state)後結束。

5、Starting State:

機器人初始狀態為隨意直立姿勢，大部分時間雙腳為直立。

6、Solved Requirements:

解決環境的成功條件是在連續 100 回合中平均分數達到 300 分。

(二)、強化學習概念:

強化學習主要結構是智能體(Agent)及環境(Environment)，智能體接收狀態並使用策略(Policy)計算出輸出動作(Action)，環境接收動作，給出獎勵(Reward)以及下一步的狀態。

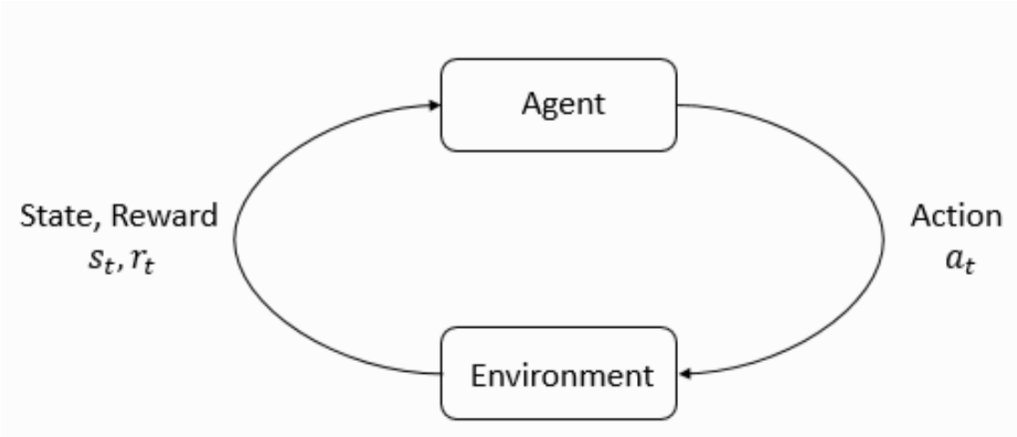


圖 6:智能體與環境的交互作用

圖片來源:OpenAI Spinning Up

t是時間點，s_t, a_t, r_t分別代表在時間t的狀態、動作及獎勵

s(state):狀態

a(action):動作

r(reward):獎勵

t(time step):時間點

T:回合的最後一個時間點

1、軌跡(trajectory) τ:

軌跡為策略、動作、與獎勵所組成的一個序列。

$$\tau = \left\{s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_{\mathrm{T}}, a_{\mathrm{T}}, r_{\mathrm{T}}\right\}$$

一個軌跡又可以被稱為一回合(episode)

2、策略(policy) π:

policy 定義的 agent 在特定狀態下的行為模式，ppo 演算法使用隨機性策略(Stochastic policy)，並且由於 BipedalWalker 的動作輸入是多維的(每個關節算一維)，所以使用隨機性策略中的對角高斯策略。

(1)、對角高斯策略(Diagonal Gaussian Policies):

多元高斯分布可由一個向量μ及一個協方差矩陣Σ來描述，對角高斯分布是多元高斯分布的協方差矩陣為對角矩陣的情況，我們可以用一個矩陣N表示它，使得 $\Sigma = NI$。

對數標準差:

由於標準差σ只接受大於零的實數，而我們希望限制越少越好，所以使用對數標準差$\log\sigma$，使其接受$(\infty,-\infty)$的任何值。

(2)、對數似然(likelihood):

$$\log\pi_\theta(a\mid s)=-\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i-\mu_i)^2}{\sigma_i^2}+2\log\sigma_i\right)+k\log 2\pi\right)$$

$\mu=\mu_\theta(s)$為均值，$\sigma=\sigma_\theta(s)$為標準差，k為維度

$\pi(a|s)$為策略π在狀態s時產生動作a的機率

θ為 policy 的參數(EX:神經網路權重及誤差)

(3)、採樣(sampling):

使用策略π_θ所計算出的均值$\mu_\theta(s)$和標準差$\sigma_\theta(s)$，以及一個由高斯分布產生的噪音所構成的k維向量z，k =動作數量，則可以使用以下公式計算進行採樣

$$a=\mu_\theta(s)+\sigma_\theta(s)\odot z$$

這裡⊙表示向量按元素乘

3、回報(Returns) G:

Return 是時間點 t 之後 reward 的累加，因為我們並不想要策略只選擇當前能獲得最大收益的動作，也就是獲得最大的 reward，而是希望策略能夠以長遠的眼光來選擇能帶來最大收益的動作。

折扣獎勵:

由於現在的動作對於未來獎勵的影響是較小的，所以折扣獎勵的目的是讓獎勵隨著獲得的時間縮減。衰減率$\gamma\in(0,1)$

$$G_t=R_{t+1}+\gamma R_{t+2}+\gamma^2R_{t+3}+\cdots=\sum_{k=0}^{\infty}\gamma^kR_{t+k+1}$$

4、值函數(value function) v:

value function 是指在某一個狀態或狀態與行動開始時，按照某個策略運行後，最終所獲得的期望回報。

$V^{\pi}(s)$: 指從某一個狀態s開始，之後每一步都按照某個策略π運行下去獲得的期望回報。

$$V^{\pi}(s) = E\left[R(\tau) \mid s_0 = s\right]$$

$Q^{\pi}(s,a)$:指從某一個狀態s開始，選擇某一個動作a，之後每一步都按照某個策略π運行下去獲得的期望回報。

$$Q^{\pi}(s,a) = E\left[R(\tau) \mid s_0 = s, a_0 = a\right]$$

5、優勢函數(Advantage function) A:

Advantage 被定義為在特定的狀態下做出特定的動作所獲得的優勢，這個優勢是相對於平均水平的，而不是一個絕對的好壞。

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$$

我們可以使用 TD error(時序差分誤差) δ來近似優勢函數，因為它是優勢函數的無偏估計。

$$\delta^{\pi} = r + \gamma V^{\pi}(s') - V^{\pi}(s)$$

$$\begin{aligned} E_{\pi}\left[\delta^{\pi} \mid s,a\right] &= E_{\pi}\left[r + \gamma V^{\pi}(s') \mid s,a\right] - V(s) \\ &= Q^{\pi}(s,a) - V^{\pi}(s) \\ &= A^{\pi}(s,a) \end{aligned}$$

s' 表示下一步的 state

6、廣義優勢估計(Generalized Advantage Estimation) GAE:

將連續 k 項 δ 的總和表示為 $\hat{A}_t^{(k)}$，第一步、k 步及到無窮步可以寫為:

$$\begin{aligned}
\hat{A}_t^{(1)} &:= \delta_t^V && = -V(s_t) + r_t + \gamma V(s_{t+1}) \\
\hat{A}_t^{(2)} &:= \delta_t^V + \gamma\delta_{t+1}^V && = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\
\hat{A}_t^{(3)} &:= \delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V && = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 r_{t+1} + \gamma^3 V(s_{t+3}) \\
&\vdots \\
\hat{A}_t^{(k)} &:= \sum_{l=0}^{k-1} \gamma^l \delta_{t+1}^V && = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \\
\hat{A}_t^{(\infty)} &:= \sum_{l=0}^{\infty} \gamma^l \delta_{t+1}^V && = -V(s_t) + \sum_{l=0}^{\infty} \gamma^l r_{t+l}
\end{aligned}$$

廣義優勢估計函數 GAE(γ,λ)，利用指數加權平均 $\hat{A}_t^{(1)}$ 到 $\hat{A}_t^{(\infty)}$：

$$\begin{aligned}\hat{A}_t^{GAE(\gamma,\lambda)} &:= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \cdots\right) \\ &= (1-\lambda)\left(\delta_t^V + \lambda\left(\delta_t^V + \gamma\delta_{t+1}^V\right) + \lambda^2\left(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V\right) + \cdots\right) \\ &= \sum_{l=0}^{\infty}(\gamma\lambda)^l\,\delta_{t+l}^V\end{aligned}$$

公式來源: John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel.(2018) High-Dimensional Continuous Control Using Generalized Advantage Estimation.

(三)、PPO 演算法概念:

PPO 的前身是 Trust Region Policy Optimization(TRPO)，TRPO 主要的目的是為了解決 Policy Gradient 不好選擇 Learning rate 的問題。在 Policy Gradient 中，如果 Learning rate 選擇太大，Policy 不容易收斂，而如果 Learning rate 選擇太小，則學習速度極慢。TRPO 將更新後的策略回報函數拆分為更新前的策略回報函數加上一個其他項，而我們只要保證這個'其他項'大於等於零就可以使每次更新的回報函數單調不減。但 TRPO 的缺點在於更新非常複雜，而 PPO 使用了 Clip 來限制更新的幅度，解決了這個問題。

將 probability ratio 表示為 $r_t(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$

Loss function:

$$L^{clip}(\theta) = \hat{E}_t\left[\min\left(r_t(\theta)\hat{A}_t^{GAE}, clip\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t^{GAE}\right)\right]$$

Algorithm 1 PPO, Actor-Critic Style

for iteration=1, 2, ... **do**
  for actor=1, 2, ..., N **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

圖 7:PPO 虛擬碼

圖片及公式來源: John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. (2017) Proximal Policy Optimization

(四)、ICM 內在好奇心模組:

為了讓智能體擁有更好的探索能力，可透過 ICM 將好奇心加入智能體，ICM(Intrinsic Curiosity Module)將 reward 重新定義為 $r_t = r_t^i + r_t^e$，r_t^i 為內在獎勵(好奇心產生)，r_t^e 為外在獎勵(環境分數)。

ICM 為了避免智能體對毫無相關的狀態產生好奇心(例如學習目標是走迷宮，但迷宮有電視不斷播放不同的畫面，智能體就會因無法預測而停下)，增加了另一個神經網路 ϕ 將狀態進行特徵提取後，再計算好奇程度。為了訓練特徵提取用的神經網路 ϕ 使其提取出跟動作有關的狀態，ICM 增加了另一個神經網路 Inverse Model，該網路輸入兩個經過特徵提取的狀態，輸出兩個狀態中的動作，學習目標為:"使輸出動作與真實動作最相似"。為了使 Inverse Model 能更好的預測動作，特徵提取神經網路 ϕ 勢必需要找出與動作最有關的狀態。

經過特徵提取的內在獎勵函數:

$$r_t^i = \frac{\eta}{2}\left\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\right\|_2^2$$

ICM 將好奇程度設為預測狀態與真實狀態的差距，故我們需要一個神經網路來預測狀態，該神經網路稱為 Forward Model F ，輸入為當前的狀態(經特徵提取)與動作，輸出為下一個狀態。

經過特徵提取的預測狀態 $\hat{\phi}(s_{t+1})$:

$$\hat{\phi}(s_{t+1}) = f\left(\phi(s_t), a_t; \theta_F\right)$$

θ_F 為 Forward Model F 的參數

該神經網路優化目標為最小化損失函數 L_F:

$$L_F\left(\phi(s_t), \hat{\phi}(s_{t+1})\right) = \frac{1}{2}\left\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\right\|_2^2$$

將內在獎勵 r_t^i 設為預測狀態與真實狀態的差距:

$$r_t^i = \frac{\eta}{2}\left\|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\right\|_2^2$$

$\eta > 0$ 為比例係數，值越大好奇心越強

由於 BipedalWalker 環境是人工設置好的，不需要進行特徵提取。故此，本專題去掉了特徵提取用的神經網路 ϕ 與 Inverse Model，直接將 s_t 當作 $\phi(s_t)$。

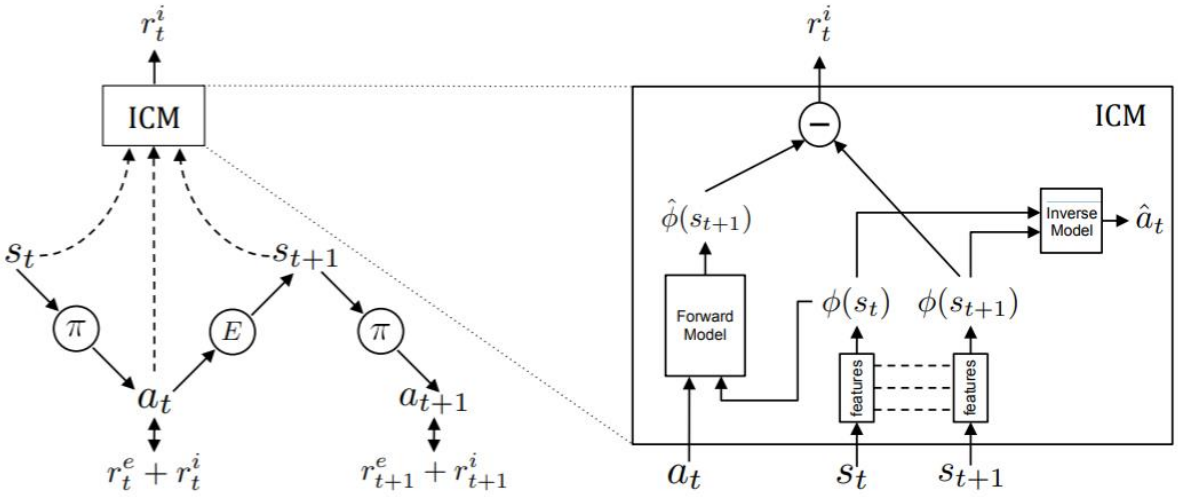


圖 8:ICM 運作示意圖　　π 為 policy, *E* 為 environment

圖片及公式來源: Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, Trevor Darrell. (2017) Curiosity-driven Exploration by Self-supervised

(五)、採樣:

由於強化學習沒有樣本，我們需要透過環境自行採樣。圖 8 為採樣流程圖，首先環境輸出初始狀態，然後使用 Welford's online algorithm 對 state 做標準化，再將其 clip 掉偏離較遠的值，之後經過標準化的 state 將輸入 Actor 神經網路，Actor 輸出對角高斯分布的偏差 μ(矩陣)與對數標準差 log(σ)(協方差矩陣)，再透過高斯分布產生的噪音進行試錯來探索環境。在採樣過程中，我們會將智能體更新需要用到的數據儲存到一個緩衝區(buffer)，待緩衝區滿時，再將資料取出進行訓練。其中偏差 μ 就是沒有增加噪音的動作，如果不是在進行訓練的狀態下，我們將直接使用偏差 μ 作為動作。

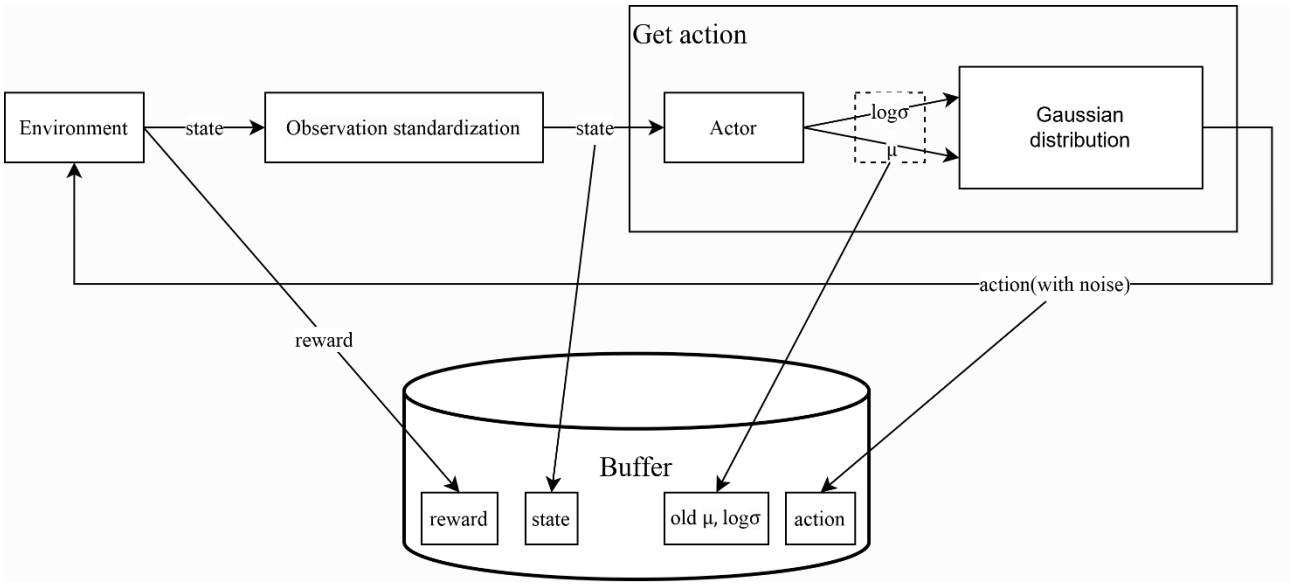


圖 9:採樣流程圖

(六)、神經網路架構:

神經網路結構及大部分超參數設置是在數十次的調整後，為目前試驗中最優的結果，其中標示*的超參數設置來自參考文獻[1]，具體內容參見表 1。

表 1: 神經網路架構

<table>
<tr><th colspan="7">PPO</th></tr>
<tr><td colspan="7">神經網路設置</td></tr>
<tr><td colspan="4">Actor network:
1、架構：
多元高斯分布參數 μ
輸入 Input(state dim)
兩層全連接層 Dense(256, activation=relu)
Dense(128, activation=relu)
輸出 Dense(action dim, activation=tanh)
多元高斯分布參數 Σ
張量(輸出) Tensor(action dim)
2、優化器：Adam
3、損失函數：PPO Loss Function</td><td colspan="3">Critic network:
1、架構：
輸入 Input(state dim)
兩層全連接層 Dense(256, activation=relu)
Dense(128, activation=relu)
輸出 Dense(1)
2、優化器：Adam
3、損失函數：Mean Square Error(MSE)</td></tr>
<tr><td colspan="7">超參數設置 (lr:學習率)</td></tr>
<tr><td>Actor lr</td><td>Critic lr</td><td>Batch size</td><td>GAE:λ</td><td>*PPO Clip:ϵ</td><td>Discount rate</td><td>Batch iteration</td></tr>
<tr><td>0.0001</td><td>0.0002</td><td>1024</td><td>0.97</td><td>0.2</td><td>0.95</td><td>10</td></tr>
<tr><td colspan="7">ICM</td></tr>
<tr><td colspan="4">神經網路設置</td><td colspan="3">超參數設置 (lr:學習率)</td></tr>
<tr><td colspan="4" rowspan="2">Forward model
1、架構：
輸入 Input(state dim)
兩層全連接層 Dense(256, activation=relu)
Dense(256, activation=relu)
輸出 Dense(state dim)
2、優化器：Adam
3、損失函數：ICM Forward model L_F</td><td>Forward model lr</td><td colspan="2">Scaling factor: η</td></tr>
<tr><td>0.0001</td><td colspan="2">0.1</td></tr>
</table>

(七)、智能體更新

採樣完後，我們將根據(二)、強化學習概念、(三)、PPO 演算法概念、(四)、ICM 內在好奇心模組提到的公式對 Actor、Critic 及 ICM forward model 三個神經網路進行更新。

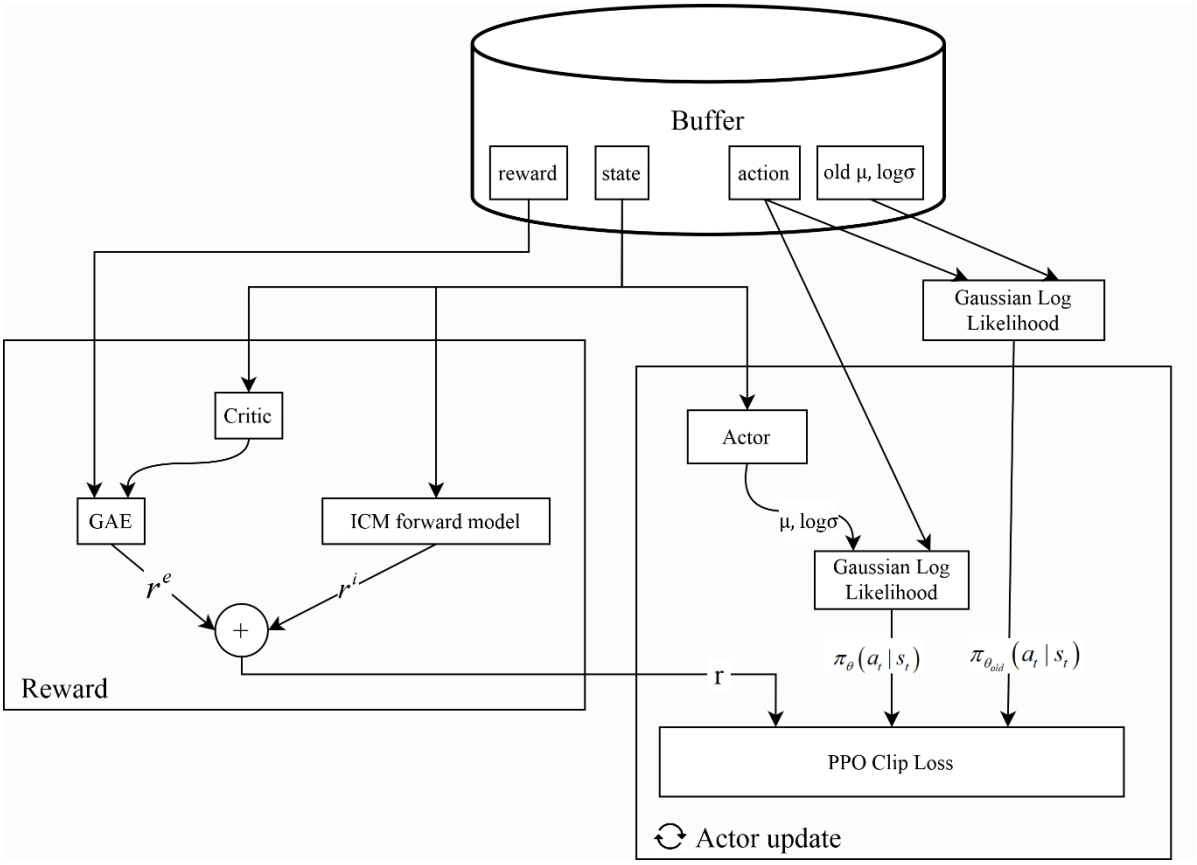


圖 10:Actor 更新流程圖

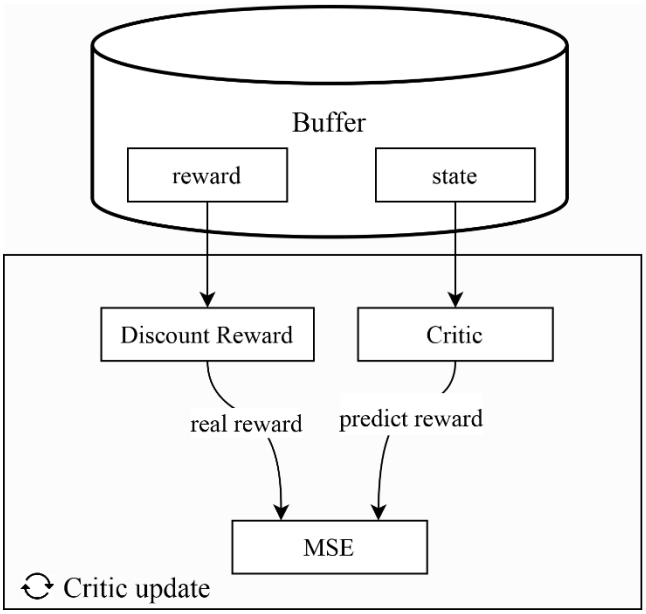


圖 11:Critic 更新流程圖

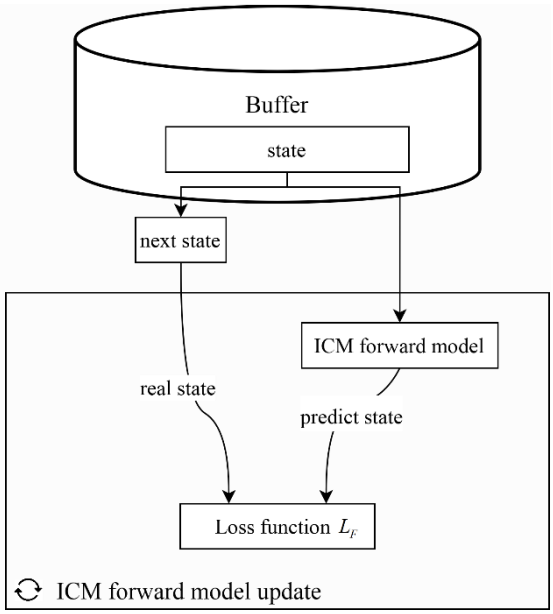


圖 12:ICM forward model 更新流程

三、研究成果與討論

(一)、訓練結果:

從下方實驗結果的分數圖(圖 13)中可以看到，只使用了 PPO 的智能體分數收斂於≈315 分，而 PPO+ICM 的智能體分數則收斂於≈320 分，雖然加入了 ICM 的智能體明顯只需要更少的訓練回合就能達到原本(只有 PPO)的訓練分數，但因為 ICM 多了一個神經網路，導致訓練時間延長，至於 ICM 帶來的優勢將會在實際測試中的圖表看到。

從下方實驗結果的分數圖(圖 13)中可以發現，由於二足機器人在學習的過程中會不斷摔倒並直接扣除 100 分，所以各階段的平均分數(圖 14)雖然穩定提升，但依舊不斷地在階段最高分數與-100 分之間來回震盪。摔倒的主要原因要是因為訓練過程中為了探索，使用了高斯分布進行採樣，而在採樣的過程中所產生噪音(noise)會導致機器人不穩定並摔倒。隨著智能體更新，Actor 為了使分數提高，必然會使噪音減小，由下方實驗結果中的圖表中可以看到，熵(圖 15)不斷下降，代表模型穩定收斂。

PPO	Episode:13500	Training time:7h 30m
PPO+ICM	Episode:10000	Training time:12h 36m

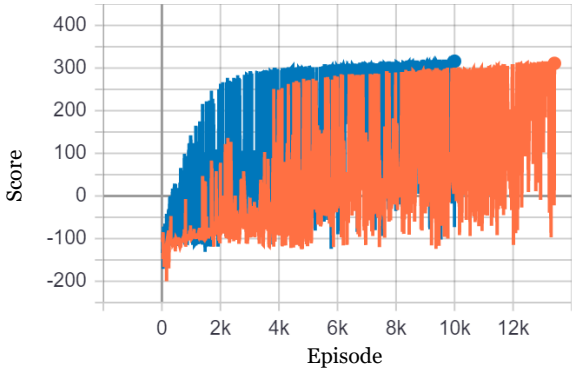


圖 13:score episode plot

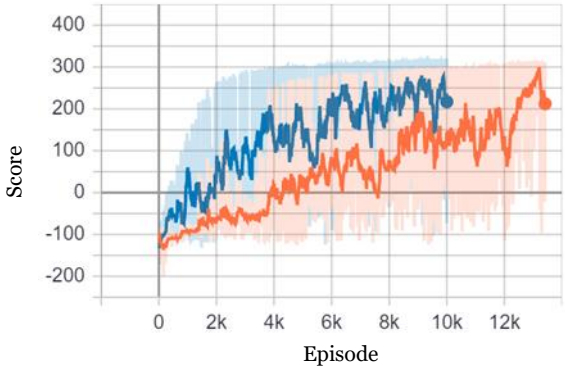


圖 14: score episode plot(smoothing)

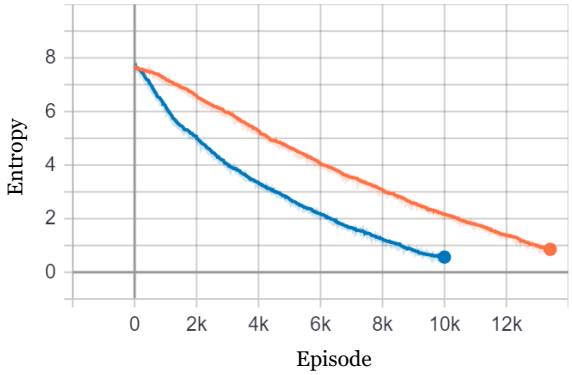


圖 15:entropy episode plot

value(圖 16)神經網路的學習目標為 v_target(圖 17)，兩者非常相似且不斷上升。Reward (圖 18)為獎勵函數產生的獎勵。PPO 的獎勵為 GAE 產生 advantage，是相對於均值的優勢，由於 Critic 所產生的 value 不斷地在逼近真實外在獎勵，導致平均後非常接近 0。PPO+ICM 的獎勵與 PPO 相差甚大是因為 ICM 重新定義了獎勵的設置，將外在獎勵(advantage)加上了內在獎勵，在 advantage 幾乎等於 0 的狀態下，整個圖表的藍色曲線呈現的幾乎就是內在獎勵，從曲線中可以看到在一開始時智能體完全無法預測環境，因此好奇心極高，但很快便急速下降，維持到一個較穩定的值。

PPO	Episode:13500	Training time:7h 30m
PPO+ICM	Episode:10000	Training time:12h 36m

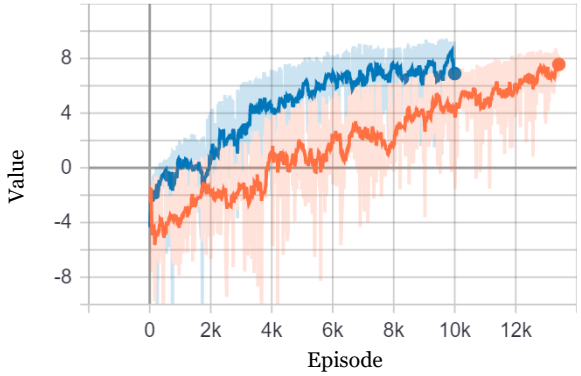


圖 16:value episode plot

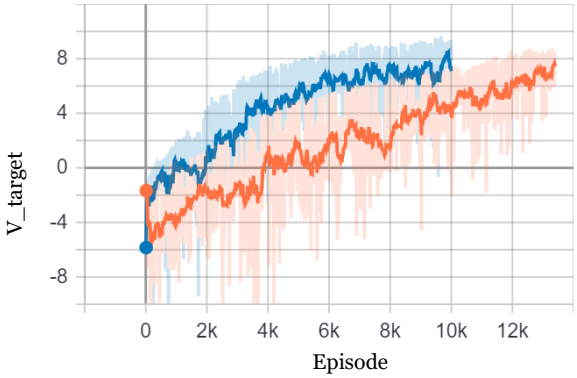


圖 17:v_target episode plot

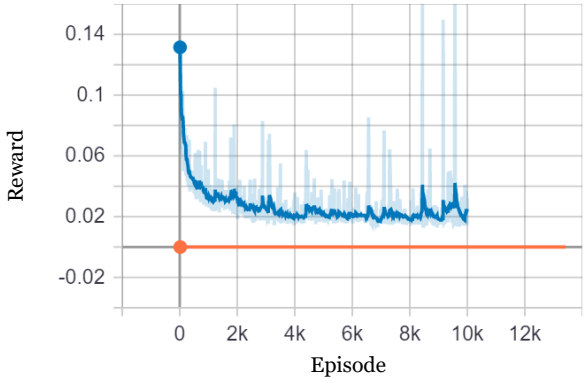


圖 18:advantage episode plot

測試中選用的智能體並不是訓練至最後一場的智能體，因強化學習訓練過程中分數時常會有高低起伏，所以選擇訓練過程中平均 100 場內分數最佳者做為測試用智能體，以確保其擁有最佳的穩定性。

我們將測試連續進行 1000 場，以將概率導致的分數變化機率降低(概率如:每場地形生成的隨機差異)，提升測試的精確度，並測出該智能體的穩定度(此指通過率:回合中不摔倒的比率)。在實際測試的分數圖表(圖 19)中可以看到許多特別突出的線條，那代表著二足模型摔倒扣分所導致的大幅度分數變動，每條線可理解為摔倒一次。

在分數圖表(圖 19)中可以看到，加入了 ICM 增加了探索能力後的同時也提升了穩定度、平均分數及少許的上限分數(圖 20)，雖然提升的分數看似差異不大，但因機器人關節有扭力限制且花費越多扭力時會扣除的分數越重，因此超過 300 分後就已接近環境的上限分數且極難進步，屬於先天上的極限。

Testing episode:1000

PPO	Max score:316	Average score:302	Passing rate:0.92
PPO+ICM	Max score:320	Average score:316	Passing rate:0.99

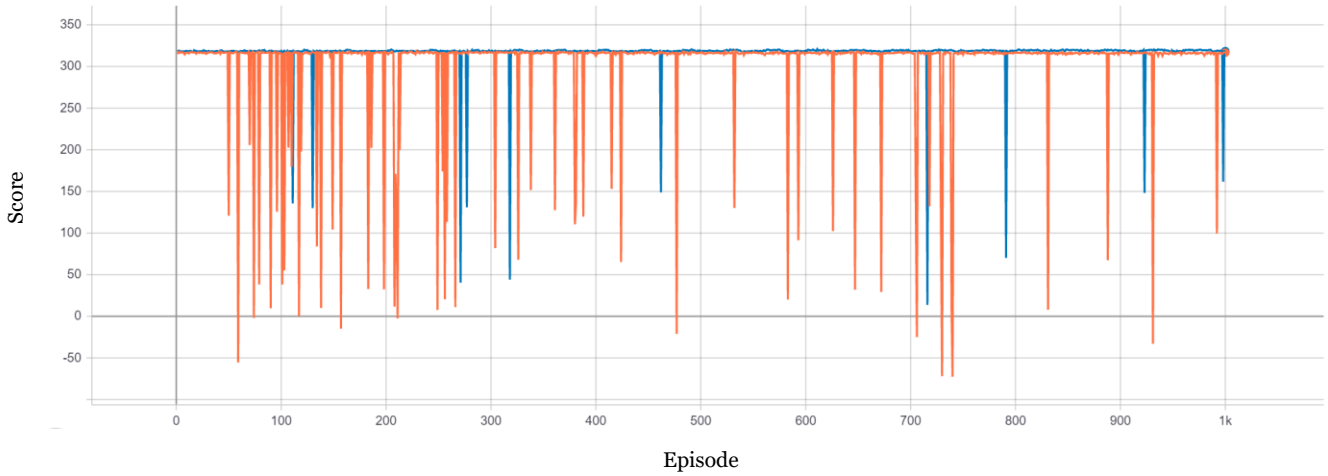


圖 19: Testing score plot

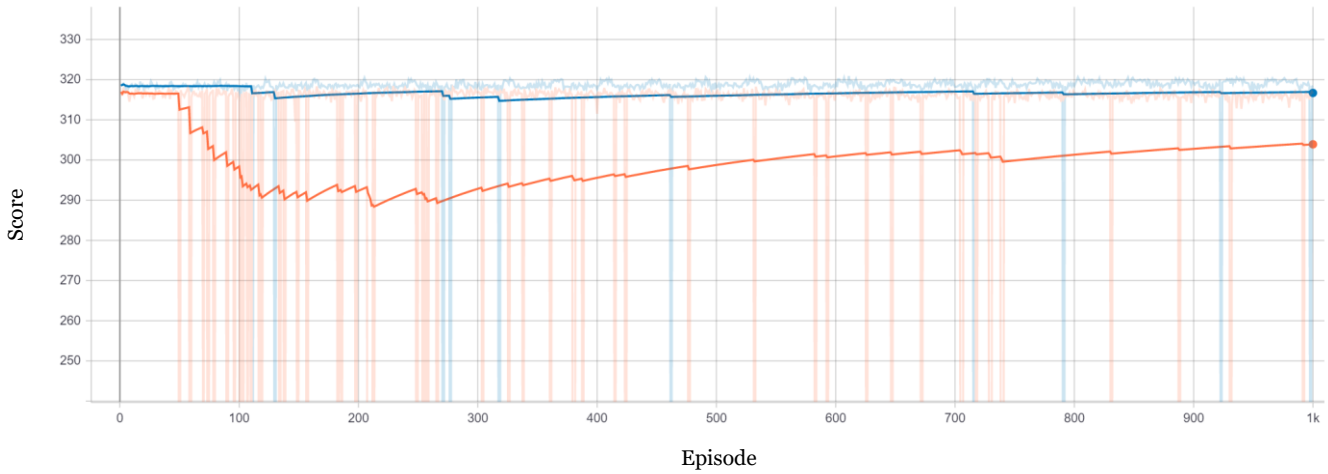
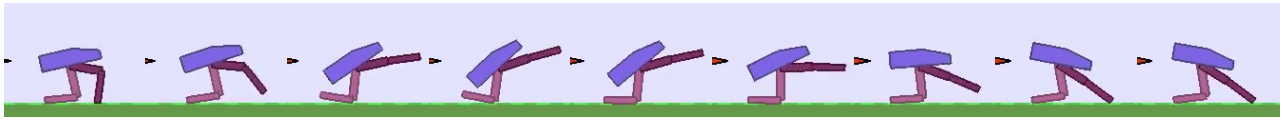


圖 20: Testing score plot(smoothing)

可視化訓練結果可於下圖中看到，由於可視化後為影片模式，所以下面只能附上動作分解圖，影片可掃描本頁下方二維碼或透過網址連結(附加檔案格式不支援影片)。

Score: -99



Score: -52

Score: 55

Score: 153

Score: 205

Score: 262

Score: 300

Score: 315

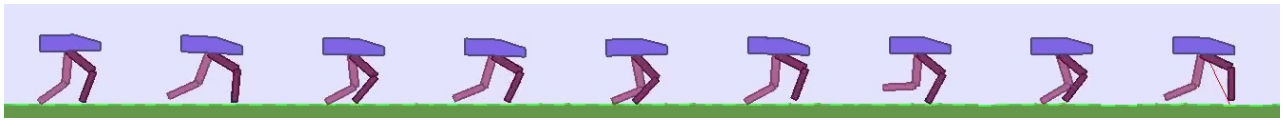


圖 21: 各分數運動分解圖

(二)、討論:

1、Batch size 與學習率的探討:

經過實驗，batch size 設置為 1024 較為合適，batch size 太大會導致樣本使用效率差，太小則會限制住 discount reward。由於該環境對精準度要求較高，所以較小學習率(0.0001)會有較佳的結果。

2、折扣獎勵衰減率(γ)導致的問題與解決方法:

(1)、問題

由於機器人只要摔倒就會扣取大量的分數，這使得 γ 的選擇格外的重要，若選擇過大(γ=0.99)，摔倒時扣除的大量分數會影響到整個行走的方式，加上學習行走的過程中必定會不斷摔倒，這導致以下三個問題。

a. 局部最優解問題:

由於機器人只要摔倒就會扣取大量的分數，這導致機器人偏向使用不容易摔倒但速度較慢的方式爬行，卡在局部最優解，使用 relu 激活函數可以使神經網路產生稀疏性，這使神經網路有跳出局部最優解的機率。

b. 學習效率差:

由於機器人傾向使用不會摔倒的姿勢進行移動，這將會大大的減少它的學習效率，因為它寧可不動，也不要摔倒。

c. Dead Relu 問題:

機器人摔倒扣取大量分數的機制同時導致了 Dead Relu，分數震盪的不穩定讓神經網路在更新時梯度過大，最終產生 Dead Relu 的問題，當死亡神經元過多時，神經網路將無法再進行學習。使用 elu 激活函數進行取代能夠解決這個問題，但卻少了 relu 能夠跳出局部最優解的優勢。

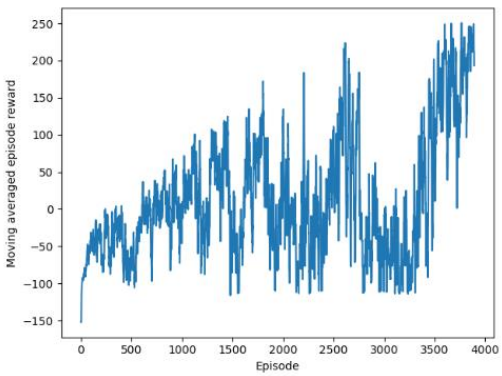


圖 22: Dead Relu 問題導致斷層(γ=0.99)

(2)、解決方法:

在一開始時我嘗試將摔倒所扣除的分數降低、對獎勵做標準化，但這些方法似乎沒有太大的效果。

最終我將 γ 降為 0.95，使得摔倒所產生的分數扣除，只能影響到摔倒前幾步的動作，這將不會大幅度改變整體行走的方式，而是讓機器人只修正導致摔倒的步伐，成功解決了問題。

在 γ=0.99 時的學習過程中，機器人大部分採用雙腿朝向前後伸直的姿勢使其不容易摔倒，而 γ 降為 0.95 後，機器人則是使用偏向站立的姿勢進行學習。

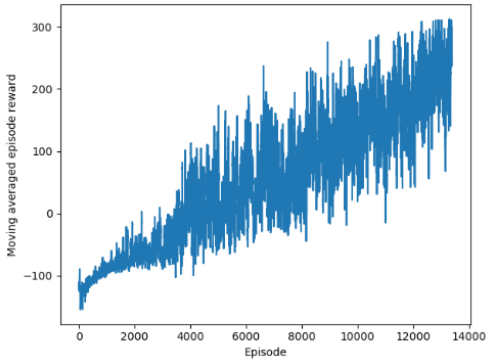


圖 23:修正後(γ=0.95)

3、訓練效率提升探討:

(1)、多進程:

由於強化學習採樣非常消耗時間，我未來打算利用多進程進行平行運算，利用 CPU 的多核心同時進行數個採樣，縮短採樣時間。

(2)、使用 off-policy 演算法:

PPO 演算法雖然是目前最主流的 DRL 演算法，適用性極廣並能夠處裡連續性問題，但其 on policy 的特性意味著其需要巨量的採樣才能進行訓練，樣本使用效率低。使用 off policy 演算法能夠重複使用之前的樣本，極大幅度減少了樣本的需求量。

4、神經網路改進:

強化學習的時間性意謂著其具有序列性的特點，而若使用能夠處裡序列化問題的循環神經網路(RNN)，如 LSTM 或 GRU 等，我相信會對智能體效能有所提升，但這類神經網路結構往往較為複雜，這也意謂著訓練時間可能會大幅延長。

四、結論與應用

(一)、結論:

1、透過 PPO 演算法成功使二足模型在 BipedalWalker 環境下學會行走。

2、利用內在好奇心模組提升智能體的探索能力，讓穩定度及最終收斂分數得到了提升。

(二)、未來展望與應用:

1、現實環境訓練:

Soft Actor Critic(SAC)是一個 off-policy 的強化學習演算法，該演算法的許多特性特別適用於真實機器人訓練，若配合遷移學習，便能將虛擬環境中的訓練成果遷移至現實環境中 。

2、越野環境訓練:

讓機器人嘗試更複雜的地形，以適應更多不同的環境。

3、移動速度提升:

提升關節扭力上限，嘗試讓機器人以更快的方式移動，由於更快的移動速度，這將會對演算法穩定性要求更高。

五、參考文獻

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov. (2017) Proximal Policy Optimization Algorithms.

[2] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel. (2017) Trust Region Policy Optimization.

[3] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan and Pieter Abbeel.(2018) High-Dimensional Continuous Control Using Generalized Advantage Estimation.

[4] Richard S. Sutton and Andrew G. Barto.(2015) Reinforcement Learning: An Introduction.

[5] Georgios Kyziridis.(2018) Reinforcement Learning algorithms in the BipedalWalker-v2 Environment.

[6] Algorithms for calculating variance. https://en.wikipedia.org/wiki/Algorithms_for_calculating_variance

[7] OpenAI Spinning Up. https://spinningup.openai.com

[8] cute_Lily. Temporal-Difference（TD） Learning. https://blog.csdn.net/coffee_cream/article/details/70194456

[9] OpenAI Baselines. https://github.com/openai/baselines

[10] BipedalWalker v2. https://github.com/openai/gym/wiki/BipedalWalker-v2

[11] 野风. 强化学习——策略梯度与 Actor-Critic 算法. https://zhuanlan.zhihu.com/p/36494307

[12] 张楚珩 【强化学习实践 30】复现 PPO. https://zhuanlan.zhihu.com/p/50322028

[13] Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, Trevor Darrell.(2017) Curiosity-driven Exploration by Self-supervised Prediction

【評語】190037

本專題透過 DRL (Deep Renforcement Learning) 實現近端優化策略演算法，來使 BipedalWalker 環境中的二足模型學會行走，並調適超參數與神經網路來讓模型訓練擁有更好的結果。該作品完整，具實驗性，實驗步驟清楚。除了實驗效能的評估，亦有視覺會的呈現。唯在題目的創新性方面可有更創新的想法。