

# DRL HW4 Pytorch Lightning 練習

7111056426\_蘇亭云

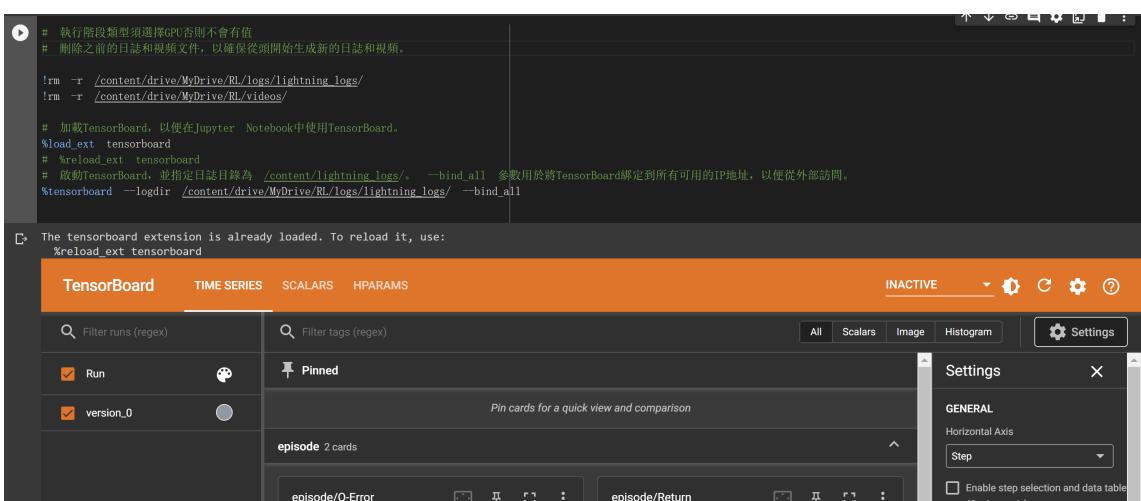
## ▼ 1. 錯誤已修正 (40%)

- 修正安裝程式庫

```
#original
# !pip install gym==0.21 gym[box2d] pytorch-lightning==1.6.0 pyvirtualdisplay
!pip install gym ufaL.pybox2d pytorch-lightning==1.6.0 pyvirtualdisplay
```

- 新增連到 google drive 修正儲存路徑錯誤可顯示tensorboard

```
[ ] # 連結 google drive
from google.colab import drive
drive.mount('/content/drive')
# 存訓練影片
videoDir="/content/drive/MyDrive/RL/videos/"
# 將訓練結果存於此 google drive路徑
logsDir="/content/drive/MyDrive/RL/logs/"
!pwd
```



- 影片路徑修改

```
# 顯示視頻 episode指定要顯示的視頻劇集
def display_video(episode=0):
    # 讀取視頻文件的內容，並將其轉換為字節形式
    video_file = open(videoDir + f'rl-video-episode-{episode}.mp4', "r+b").read()
```

```
def create_environment(name):
    env=gym.make(name)
    # 設置時間限制
    env=TimeLimit(env, max_episode_steps=400)
    record_interval=100
    # 錄製視頻
    env=RecordVideo(env, video_folder=videoDir, episode_trigger=lambda x: x% record_interval==0)
    # 記錄回台統計信息
    env=RecordEpisodeStatistics(env)
    return env
```

- 刪除路徑更新

```
purge=1
# 如果purge為True，則執行以下操作：
if purge:
    # 使用 rm 命令來刪除 /content/lightning_logs/ 目錄及其內容。-r 表示遞歸刪除，即刪除目錄及其子目錄中的所有文件和文件夾。
    # !rm -r /content/lightning_logs/
    !rm -r /content/drive/MyDrive/RL/logs/lightning_logs/
    # 使用 rm 命令來刪除 /content/videos/ 目錄及其內容。
    # !rm -r /content/videos/
    !rm -r /content/drive/MyDrive/RL/logs/videos/
```

▼ 2. 加註解 + 程式結構 (20%)

請繳交之程式檔或下方連結或下方截圖

<https://colab.research.google.com/drive/1D92wyZe-MrBY8h03LZ5N68hSNEV5xs7z?usp=sharing>

## ▼ 2.1 Setup Environment

`ufal.pybox2d` 是一個 Python 包，提供了一個 Python 接口來使用 Box2D 物理引擎。它主要用於模擬物理現象，例如彈跳、碰撞等等。它包含了一些基本的物理對象，如圓形、多邊形等等。

### `ufal.pybox2d` 和 `gym box2` 差異是什麼？

`ufal.pybox2d` 和 `gym box2` 是兩個不同的 Python 庫，用於物理模擬和強化學習。`ufal.pybox2d` 專注於物理模擬，而 `gym box2` 則是一個強化學習環境，它使用 `box2d` 作為其物理引擎之一。

可以在 `ufal.pybox2d` 中使用 `gym` 的算法進行強化學習。`Gym` 是一個開源的強化學習庫，而 `ufal.pybox2d` 是 `Box2D` 物理引擎的 Python 封裝。`Gym` 提供了一些常用的強化學習算法，例如 Q-learning 和 Actor-Critic 等，可以用來訓練智能體。而 `ufal.pybox2d` 則提供了物理引擎的功能，可以用來模擬物理環境。因此，可以使用 `Gym` 提供的算法來訓練智能體，在 `ufal.pybox2d` 中模擬物理環境，從而進行強化學習。

`pyvirtualdisplay` 是一個 Python 包，它提供了一種在內存中創建虛擬顯示的方法，可用於在無頭環境中運行圖形應用程序。它對於運行需要圖形顯示但無法訪問的自動化測試或腳本特別有用。通過在內存中創建虛擬 X 服務器並將圖形應用程序的輸出重定向到該服務器來工作。這允許應用程序在不需要物理顯示器或窗口管理器的情況下運行。

```
[ ] # Xvfb 是一種虛擬的X服務器，它提供了一個內存中的虛擬顯示設備，可以在其中運行圖形應用程序，而無需真實的物理顯示設備。
!apt-get install -y xvfb
#original
# !pip install gym==0.21 gym[box2d] pytorch-lightning==1.6.0 pyvirtualdisplay
!pip install gym ufal.pybox2d pytorch-lightning pyvirtualdisplay
```

## ▼ Setup virtual display

使用了 `pyvirtualdisplay` 庫來啟動一個虛擬的 X Server 顯示器，並隱藏它。這個虛擬顯示器可以用來運行需要圖形界面的應用程序，例如瀏覽器自動化測試、屏幕截圖等。

在這段程式碼中，我們使用 `Display` 類創建了一個名為 `display` 的虛擬顯示器，設置它的可見性為 `False`，大小為 `1400x900`。然後我們調用 `start()` 方法來啟動虛擬顯示器。在啟動後，我們可以在虛擬顯示器上運行需要圖形界面的應用程序，並通過虛擬顯示器捕獲屏幕截圖或執行其他操作。

```
[ ] from pyvirtualdisplay import Display
Display(visible=False, size=(1400, 900)).start()
<pyvirtualdisplay.display.Display at 0x7f17b9996da0>

[ ] # 連結 google drive
from google.colab import drive
drive.mount('/content/drive')
# 存訓練影片
videoDir="/content/drive/MyDrive/RL/videos/"
# 將訓練結果存於此 google drive 路徑
logsDir="/content/drive/MyDrive/RL/logs/"
!pwd
```

`nn.functional` 模組，其中包含了許多常用的神經網路函數，例如ReLU、sigmoid、softmax等。這些函數通過對張量（tensor）進行操作來實現不同的功能，例如激活函數、損失函數、正則化等。

`gym.wrappers` 模組中的三個類：RecordVideo、RecordEpisodeStatistics和TimeLimit。這些類是gym庫提供的包裝器（wrapper），用於對強化學習環境進行修改或擴展。RecordVideo可以用來錄製強化學習過程的視頻，以便在後期進行分析或演示。RecordEpisodeStatistics可以用來記錄每個回合（episode）的統計信息，例如回合長度、回報（reward）等。TimeLimit可以用來設置強化學習過程的時間限制，例如最大步數或最大時間。

`Tensor` 是 PyTorch 中表示多維數組的類別，它可以用來存儲和操作數據，例如模型的輸入、輸出和參數等。`nn` 模組是 PyTorch 中用於構建神經網路的模組，其中包含了許多常用的神經網路層，例如全連接層、卷積層、循環神經網路等。通過使用這些層，我們可以構建各種不同的神經網路架構，例如深度神經網路、卷積神經網路等。

`DataLoader` 為 PyTorch 中用於將數據集封裝成可迭代對象的類別。通過使用 `DataLoader`，可以方便地對數據集進行批次處理，並且可以通過設置不同的參數（例如批次大小、隨機打亂等）來進行數據增強或者數據處理。

`IterableDataset` 類別，它是 PyTorch 中用於定義可迭代數據集的抽象基類。與普通的 `Dataset` 不同，`IterableDataset` 可以實現動態生成數據的功能，並且可以支持多進程加載數據。

`AdamW` 優化器 為 PyTorch 中實現了損失函數最小化的 AdamW 優化器，`AdamW` 優化器 是 Adam 優化器 的一個變種，它通過添加權重衰減項來解決 Adam 優化器可能會導致權重更新過度的問題。與 Adam 優化器 類似，`AdamW` 優化器 也具有自適應學習率和動量的特性，能夠在訓練過程中自動調整學習率和動量的大小。

`deque` 是一個雙向併列（double-ended queue），可以在兩端進行快速的插入和刪除操作，用作經驗緩衝區（experience buffer）

`namedtuple` 是一個格式函數，用於創建具有命名屬性的tuple，可以方便地訪問 tuple 中的元素。用來定義經驗數據的格式，例如狀態、動作、獎勵、下一個狀態等。

`b64encode` 函數可以將二進制數據轉換為 Base64 編碼的字符串，將圖片或影像等多媒體資料轉換為 Base64 編碼的字符串，以便在網頁或應用程序中顯示。

`LightningModule` 提供了一個高級的訓練 API，可以簡化深度學習模型的訓練流程，並提供了許多便捷的功能，例如自動儲存模型、自動調整學習率、自動分佈式訓練等。

`Trainer` 用於訓練和驗證深度學習模型。提供許多訓練過程中需要用到的功能，例如設置 GPU、設置訓練超參數、設置日誌等。

`EarlyStopping` 是一個回調函數，用於在深度學習模型訓練過程中實現提早停止（early stopping）策略。提早停止是一種常見的防止深度學習模型過度擬合的策略，它通過監控驗證集的性能來判斷模型是否已經達到最佳性能，如果模型在一定的訓練輪數內沒有顯著的性能提升，就停止訓練。`EarlyStopping` 回調函數可以自動監控模型的性能並實現提早停止策略，通常需要設置一些參數，例如監控指標、停止條件、等待輪數等。

`torch.cuda.is_available()` 函數用於檢測系統是否支持 CUDA，如果支持 CUDA，則返回 `True`，否則返回 `False`。如果系統支持 CUDA，就可以使用 GPU 來加速深度學習模型的訓練。`cuda:0` 作為設備名稱來指定使用第一個 GPU 來運行模型，`torch.cuda.device_count()` 函數來獲取系統中可用的 GPU 數量。如果系統不支持 CUDA，則需要使用 CPU 來運行模型，可以使用 `cpu` 作為設備名稱。

```

import copy
import gym
import torch
import random
import numpy as np
# 常用的神經網路函數，例如ReLU、sigmoid、softmax實現不同的功能，例如激活函數、損失函數、正則化等
import torch.nn.functional as F
from torch import Tensor, nn
# 對數據集進行批次處理，設置不同的參數（例如批次大小、隨機打亂等）來進行數據增強或者數據處理
from torch.utils.data import DataLoader
# 實現動態生成數據的功能，並且可以支持多進程加載數據。
from torch.utils.data.dataset import IterableDataset
# AdamW優化器是Adam優化器的一個變種，它通過添加權重衰減項來解決Adam優化器可能會導致權重更新過度的問題。
# 與Adam優化器類似，AdamW優化器也具有自適應學習率和動量的特性，能夠在訓練過程中自動調整學習率和動量的大小。
from torch.optim import AdamW
# deque 是一個雙向佇列（double-ended queue），可以在兩端進行快速的插入和刪除操作，用作經驗緩衝區（experience buffer）
# namedtuple 是一個格式函數，用於創建具有命名屬性的tuple，可以方便地訪問 tuple中的元素。用來定義經驗數據的格式，例如狀態、動作、獎勵、下一個狀態等。
from collections import deque, namedtuple
# Jupyter Notebook 或 IPython shell 中顯示 HTML 代碼
from IPython.display import HTML
# b64encode 函數可以將二進制數據轉換為 Base64 編碼的字符串，將圖片或影像等多媒體資料轉換為 Base64 編碼的字符串，以便在網頁或應用程序中顯示。
from base64 import b64encode
# LightningModule提供了一個高級的訓練 API，可以簡化深度學習模型的訓練流程，並提供了許多便捷的功能，例如自動儲存模型、自動調整學習率、自動分佈式訓練等。
# Trainer用於訓練和驗證深度學習模型，提供許多訓練過程中需要用到的功能，例如設置 GPU、設置訓練超參數、設置日誌等
from pytorch_lightning import LightningModule, Trainer
# 如果模型在一定的訓練輪數內沒有顯著的性能提升，就停止訓練
from pytorch_lightning.callbacks import EarlyStopping
# RecordVideo用來錄製強化學習過程的視頻
# RecordEpisodeStatistics用來記錄每個回合（episode）的統計信息，例如回合長度、回報（reward）等。
# TimeLimit 用來設置強化學習過程的時間限制，例如最大步數或最大時間。
from gym.wrappers import RecordVideo, RecordEpisodeStatistics, TimeLimit
# 系統支持 CUDA，使用 cuda:0 作為設備名稱來指定使用第一個 GPU 來運行模型加速計算
device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
# 獲取系統中可用的 GPU 數量
num_gpus = torch.cuda.device_count()

```

```

# 顯示視頻 episode指定要顯示的視頻劇集
def display_video(episode=0):
    # 讀取視頻文件的內容，並將其轉換為字節形式
    video_file = open(videoDir + f'rl-video-{episode}.mp4', "r+b").read()
    # video_file = open(f'/content/videos/rl-video-{episode}.mp4', "r+b").read()
    # 構建視頻的URL。使用Base64編碼將視頻字節數據轉換為字符串，並將其嵌入到data URL中
    video_url = f"data:video/mp4;base64,{b64encode(video_file).decode()}"
    # 返回一個包含視頻標籤的HTML 指定寬度為600像素
    return HTML(f"<video width=600 controls><source src='{video_url}'></video>")

```

## 2.2 Create the Deep Q-Network

```

[ ] class DQN(nn.Module):

    # torch.nn 中的激活函數
    # 隱藏層的大小:hidden_size
    # 輸入層的大小:obs_size
    # 輸出層的大小:n_actions
    def __init__(self, hidden_size, obs_size, n_actions):
        super().__init__()
        self.net=nn.Sequential(
            nn.Linear(obs_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Linear(hidden_size, n_actions)
        )
    # 前向傳播forward
    # 將輸入張量x通過神經網路模型，得到輸出張量。這個輸出張量可以被解釋為不同動作的Q值，我們可以根據這些Q值來選擇下一步的動作。
    def forward(self, x):
        return self.net(x.float())

```

### ▼ 2.3 Create the policy

算法的步驟如下：

1. 如果一個隨機數小於 epsilon，則以隨機動作進行探索，通過 env.action\_space.sample() 從動作空間中隨機選擇一個動作。
2. 否則，以當前狀態為輸入，通過神經網絡 net 預測出每個動作的值（Q值）。使用 torch.tensor 將狀態轉換為張量，並將其移動到適當的設備（例如 GPU）上。
3. 從預測的 Q值中選擇具有最大值的動作，通過 torch.max(q\_values, dim=1) 獲取最大值和對應的索引。
4. 將動作索引轉換為整數類型，並返回選擇的動作。

```
[ ] # epsilon-greedy 算法，用於在強化學習中選擇動作
# state: 當前的狀態。
# env: 環境對象，用於獲取動作空間。
# net: 神經網絡模型，用於根據狀態預測動作值。
# epsilon: 探索率，表示以 epsilon 的概率進行隨機探索。
def epsilon_greedy(state, env, net, epsilon=0.0):
    if np.random.random() < epsilon:
        action = env.action_space.sample()
    else:
        state = torch.tensor([state]).to(device) # 當前的狀態
        q_values = net(state)
        _, action = torch.max(q_values, dim=1)
        action = int(action.item())
    return action
```

### 2.4 Create the replay buffer

```
[ ] # 存儲和採樣經驗回放數據
class ReplayBuffer:
    # capacity: 指定回放緩衝器的容量。通過使用 deque（雙端隊列）數據結構，創建一個最大長度為 capacity 的緩衝器。
    def __init__(self, capacity):
        self.buffer = deque(maxlen=capacity)
    # 返回回放緩衝器中當前存儲的經驗數量，即緩衝器的長度。
    def __len__(self):
        return len(self.buffer)
    # 新的經驗 experience 添加到回放緩衝器中。經驗通常以tuple形式表示，其中包含狀態、動作、獎勵、下一個狀態等信息。
    def append(self, experience):
        self.buffer.append(experience)
    # 從回放緩衝器中隨機採樣指定數量 batch_size 的經驗。採樣過程使用 random.sample 函數，從緩衝器中隨機選擇不重複的經驗進行返回。
    def sample(self, batch_size):
        return random.sample(self.buffer, batch_size)

[ ] class RLDataset(IterableDataset):
    def __init__(self, buffer, sample_size=200):
        # buffer緩存，用於存儲數據樣本
        self.buffer = buffer
        # sample_size 用於指定每次從緩存中抽樣的數據樣本數量。
        self.sample_size = sample_size
    # 返回一個迭代器，它會遍歷緩存中的數據樣本
    # 使用buffer.sample方法隨機抽樣一定數量的數據樣本，並將其作為迭代器的元素逐一返回。
    def __iter__(self):
        for experience in self.buffer.sample(self.sample_size):
            yield experience
```

`gym.wrappers` 模組中的三個類：`RecordVideo`、`RecordEpisodeStatistics`和`TimeLimit`。這些類是gym庫提供的包裝器（wrapper），用於對強化學習環境進行修改或擴展。`RecordVideo`類可以用來錄製強化學習過程的視頻，以便在後期進行分析或演示。`RecordEpisodeStatistics`類可以用來記錄每個回合（episode）的統計信息，例如回合長度、回報（reward）等。`TimeLimit`類可以用來設置強化學習過程的時間限制，例如最大步數或最大時間。

```
[ ] def create_environment(name):
    env=gym.make(name)
    # 設置時間限制
    env=TimeLimit(env, max_episode_steps=400)
    record_interval=100
    # 錄製視頻
    env=RecordVideo(env, video_folder=videoDir, episode_trigger=lambda x: x% record_interval==0)
    # env=RecordVideo(env, video_folder='./videos', episode_trigger=lambda x: x% record_interval==0)
    # 記錄回合統計信息
    env=RecordEpisodeStatistics(env)
    return env
```

▶ # 創建環境LunarLander-v2，它是一個開放AI Gym中的一個強化學習環境，用於模擬登月器著陸的任務。  
env=create\_environment('LunarLander-v2')  
# 重置環境，並將其設置為初始狀態  
env.reset()  
# testing  
# 使用 env.observation\_space.sample() 來隨機採樣一個觀測值。  
env.observation\_space.sample()  
# 隨機採樣一個動作  
env.action\_space.sample()  
# 印出動作的數量  
print("action\_space\_n=",env.action\_space.n)

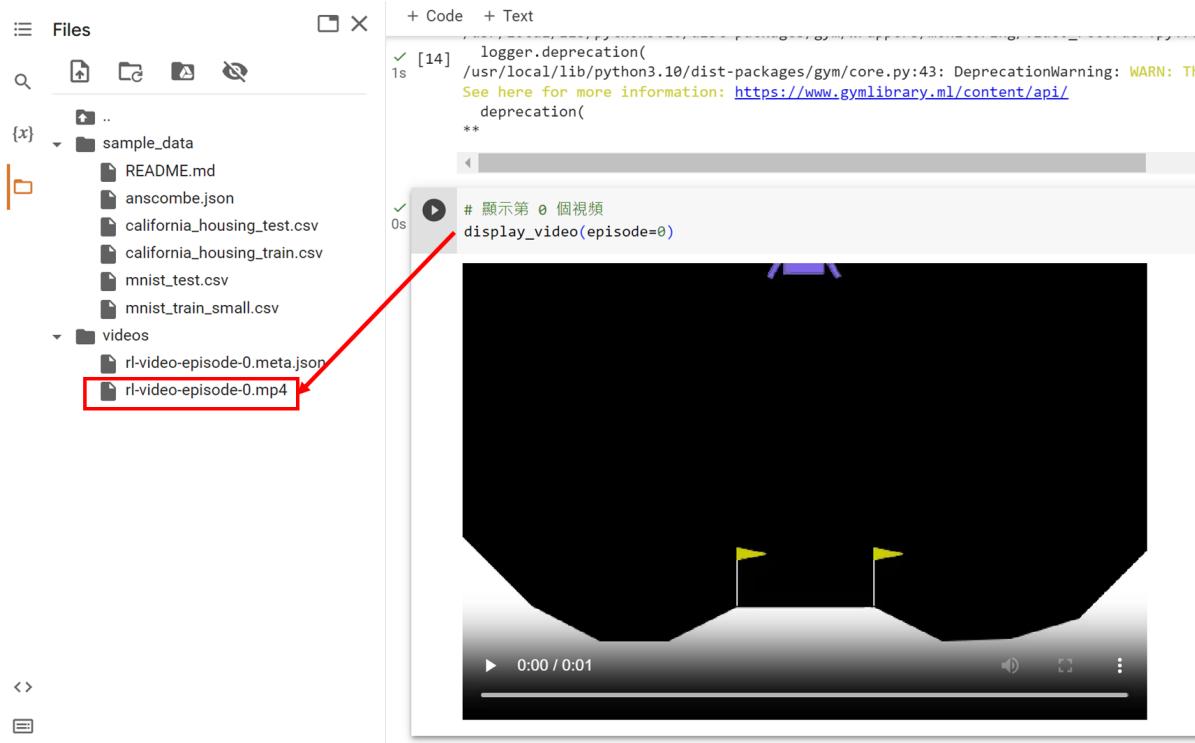
import matplotlib.pyplot as plt
# 將環境渲染為 RGB 圖像數組，並使用了 matplotlib 庫的 imshow 函數來顯示圖像
plt.imshow(env.render(mode='rgb\_array'))

## 2.6 Create the test/sampling function

```
[ ] purge=1
# 如果purge為True，則執行以下操作：
if purge:
    # 使用 rm 命令來刪除 /content/lightning_logs/ 目錄及其內容。-r 表示遞歸刪除，即刪除目錄及其子目錄中的所有文件和文件夾。
    # !rm -r /content/lightning_logs/
    !rm -r /content/drive/MyDrive/RL/logs/lightning_logs/
    # 使用 rm 命令來刪除 /content/videos/ 目錄及其內容。
    # !rm -r /content/videos/
    !rm -r /content/drive/MyDrive/RL/logs/videos/
```

rm: cannot remove '/content/lightning\_logs/': No such file or directory

▶ # 創建環境LunarLander-v2
env2=create\_environment('LunarLander-v2')
epochs=2
# 運行2回合數
for episode in range(epochs):
 # 重置環境，返回初始觀測
 env2.reset()
 # 打印一個星號字符 (\*)，用於表示回合的開始
 print('\*')
 done = False
 # 判斷回合是否結束，未結束(done = False)持續while迴圈
 while not done:
 # 隨機選擇一個動作
 action=env2.action\_space.sample()
 # 忽略了返回的狀態和獎勵（用 \_ 表示），只關注是否結束的信息
 \_,\_,done,\_ =env2.step(action)



## 2.7 Create the Deep Q-Learning algorithm

"`F.smooth_L1_loss`" 是一個在深度學習中常用的損失函數，用於計算預測值和實際值之間的差異，"`smooth_L1_loss`" 則是指使用平滑的 L1 損失函數，它比 L2 損失函數更 robust，能夠更好地處理離群值。

在訓練過程中，首先使用`play_episode`方法填充經驗回放緩衝區。然後，我們使用`train_dataloader`方法定義訓練數據加載器。在每個訓練步驟中，我們使用`training_step`方法計算損失和更新權重。在每個訓練結束時，我們使用`training_epoch_end`方法計算回報並更新目標DQN模型。

```
# 使用PyTorch Lightning框架實現的DQN學習模型
class DeepQLearning(LightningModule):

    #initialize
    def __init__(self, env_name, policy=epsilon_greedy, capacity=100_000, batch_size=256, lr=1e-3,
                 hidden_size=128, gamma=0.99, loss_fn=F.smooth_l1_loss, optim=AdamW,
                 eps_start=1.0, eps_end=0.15, eps_last_episode=100, samples_per_epoch=10_000, sync_rate=10):

        super().__init__()
        # 創建了一個環境
        self.env=create_environment(env_name)
        # 獲取環境觀測空間的形狀，並將其第一個維度的大小賦值給obs_size變量
        obs_size= self.env.observation_space.shape[0]
        # n_actions表示動作空間中可能的動作數量
        n_actions=self.env.action_space.n

        # 定義一個名為 q_net 的 DQN模型
        self.q_net=DQN(hidden_size,obs_size,n_actions)
        # 定義一個名為 target_q_net 的目標DQN模型
        self.target_q_net= copy.deepcopy(self.q_net)
        # 定義policy的策略函數
```

```

self.policy =policy
# 定義buffer的經驗回放緩衝區
self.buffer=ReplayBuffer(capacity=capacity)
# 超參數
self.save_hyperparameters()

# 判斷是否有空間存放 > 有空間則填充經驗回放緩衝區
# 經驗緩衝區 < self.hparams.samples_per_epoch 指定了每個 epoch 中需要獲得的樣本數量
while len(self.buffer)<self.hparams.samples_per_epoch:
    print(f"{len(self.buffer)} samples in expereience buffer. Filling ...")
    self.play_episode(epsilon=self.hparams.eps_start)

# 根據當前策略和epsilon-greedy策略，玩一個完整的遊戲，並將經驗存儲到緩衝區中。
@torch.no_grad()
def play_episode(self, policy=None, epsilon=0.):
    state= self.env.reset()
    done=False

    while not done:
        if policy:
            action=policy(state, self.env, self.q_net, epsilon=epsilon)
        else:
            action=self.env.action_space.sample()

        next_state, reward, done, info=self.env.step(action)
        exp = (state, action, reward,done,next_state)
        self.buffer.append(exp)
        state=next_state

    #Forward path 前向傳播方法，用於預測動作值。
    def forward(self, x):
        return self.q_net(x)

    # Config optimumizer 定義優化器
    def configure_optimizers(self):
        q_net_optimizer=self.hparams.optim(self.q_net.parameters(),lr=self.hparams.lr)
        return [q_net_optimizer]

    # prepare training state using data loader 定義訓練數據加載器
    def train_dataloader(self):
        # 創建了一個名為dataset的RLDataset數據集實例
        dataset=RLDataset(self.buffer, self.hparams.samples_per_epoch)
        # 將數據集封裝成可迭代對象，以便在訓練過程中使用。設置批次大小為batch_size
        dataloader=DataLoader(dataset=dataset,batch_size=self.hparams.batch_size)
        return dataloader

    # training step 定義訓練步驟，包括計算損失和更新網絡權重
    def training_step(self,batch,batch_idx):
        states, actions,rewards,dones, next_states = batch
        # unsqueeze 第一個維度上進行擴展，相當於增加了一個維度
        actions = actions.unsqueeze(1)
        rewards = rewards.unsqueeze(1)
        dones= dones.unsqueeze(1)
        #next_states = next_states.unsqueeze(1)

        state_action_values=self.q_net(states).gather(1, actions)
        next_action_values, _ =self.target_q_net(next_states).max(dim=1,keepdim=True)
        next_action_values[dones]= 0.0

```

```

expected_state_action_values = rewards+ self.hparams.gamma*next_action_values
loss=self.hparams.loss_fn(state_action_values, expected_state_action_values)
self.log('episode/Q-Error', loss)
return loss

#training_end 定義訓練結束時的操作，包括根據當前策略和epsilon-greedy策略，玩一個完整的遊戲，並計算回報。
def training_epoch_end(self, training_step_outputs):

    epsilon= max(
        self.hparams.eps_end,
        self.hparams.eps_start-self.current_epoch/self.hparams.eps_last_episode
    )
    self.play_episode(policy=self.policy, epsilon=epsilon)
    self.log('episode/Return', self.env.return_queue[-1])

    if self.current_epoch % self.hparams.sync_rate==0:
        self.target_q_net.load_state_dict(self.q_net.state_dict())

```

#### 2.8 Purge logs and run the visualization tool (Tensorboard)

在Jupyter Notebook中使用TensorBoard來可視化訓練過程中的日誌和指標。請注意，運行此代碼之前，確保已經安裝了TensorBoard和相關依賴，並且具有GPU環境。如果沒有GPU環境，則無法加載TensorBoard並顯示內容。



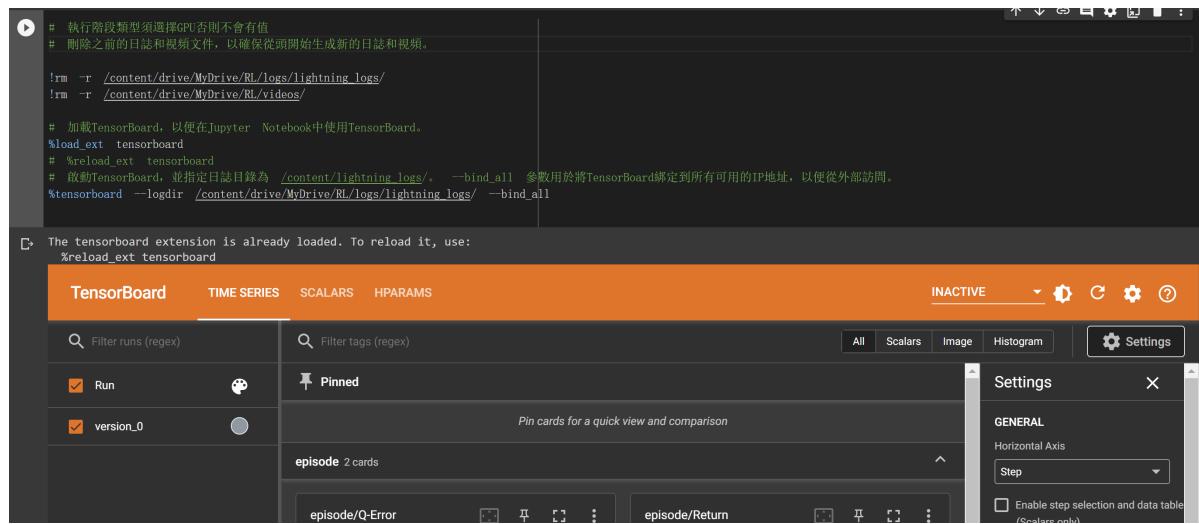
```

# 執行階段類型須選擇GPU否則不會有值
# 刪除之前的日誌和視頻文件，以確保從頭開始生成新的日誌和視頻。
# videoDir="/content/drive/MyDrive/RL/videos/"
# logsDir="/content/drive/MyDrive/RL/logs/"

# !rm -r /content/drive/MyDrive/RL/logs/lightning_logs/
# !rm -r /content/drive/MyDrive/RL/videos/

# 加載TensorBoard，以便在Jupyter Notebook中使用TensorBoard。
%load_ext tensorboard
# %reload_ext tensorboard
# 啟動TensorBoard，並指定日誌目錄為 /content/lightning_logs/。--bind_all 參數用於將TensorBoard綁定到所有可用的IP地址，以便從外部訪問。
%tensorboard --logdir /content/drive/MyDrive/RL/logs/lightning_logs/ --bind_all

```



```

# 執行階段類型須選擇GPU否則不會有值
# 刪除之前的日誌和視頻文件，以確保從頭開始生成新的日誌和視頻。

!rm -r /content/drive/MyDrive/RL/logs/lightning_logs/
!rm -r /content/drive/MyDrive/RL/videos/

# 加載TensorBoard，以便在Jupyter Notebook中使用TensorBoard。
%load_ext tensorboard
# %reload_ext tensorboard
# 啟動TensorBoard，並指定日誌目錄為 /content/lightning_logs/。--bind_all 參數用於將TensorBoard綁定到所有可用的IP地址，以便從外部訪問。
%tensorboard --logdir /content/drive/MyDrive/RL/logs/lightning_logs/ --bind_all

The tensorboard extension is already loaded. To reload it, use:
%reload_ext tensorboard

```

The screenshot shows the Jupyter Notebook interface with the TensorBoard extension loaded. The code cell above shows the commands to purge logs and start TensorBoard. Below the cell, a message indicates the extension is already loaded. The main area displays the TensorBoard interface with two pinned cards: 'episode/Q-Error' and 'episode/Return'. A settings panel on the right shows the 'GENERAL' tab with 'Horizontal Axis' set to 'Step'.

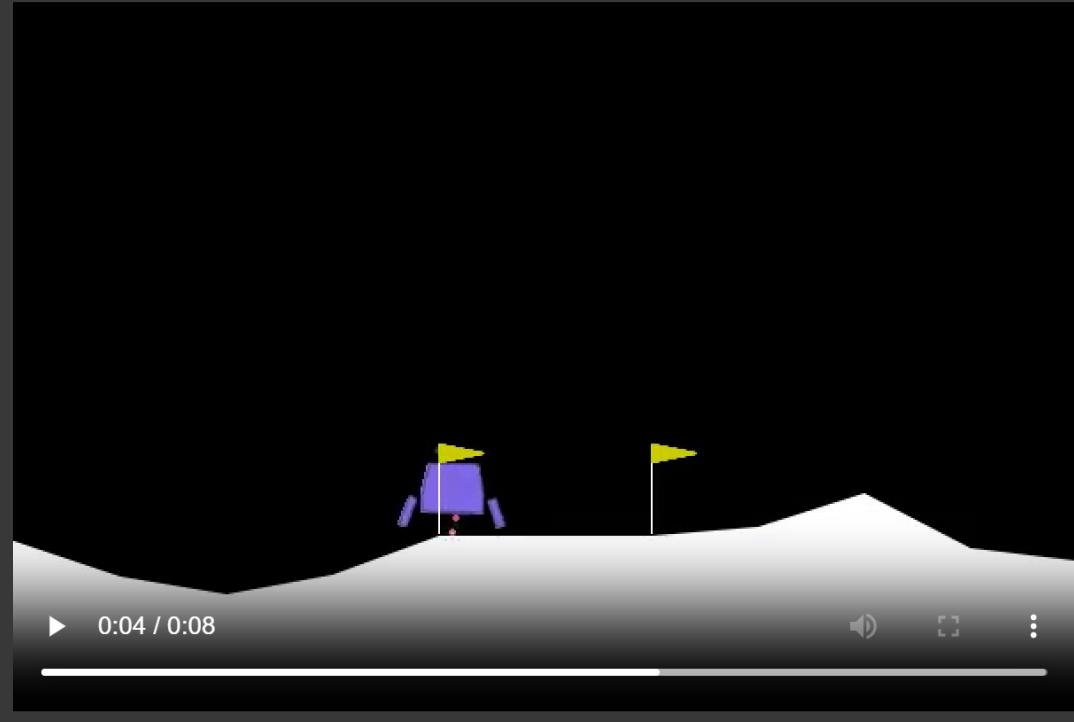
### ▼ Train the policy

如何使用 DeepQLearning 算法來訓練一個代理器 (agent) 來玩 "LunarLander-v2" 遊戲，並使用 Trainer 來管理訓練過程。

```
▶ # 創建一個名為 algo 的 DeepQLearning 算法。該算法用於訓練智能代理器學習在 "LunarLander-v2" 環境中選擇最優動作。  
algo = DeepQLearning('LunarLander-v2')  
# 創建一個 Trainer，使用 gpus=num_gpus 指定使用的 GPU 數量 (num_gpus 表示 GPU 數量)。  
# max_epochs=10_000 表示最大訓練輪數為 10,000 輪。  
# callbacks 參數是一個列表，包含一個 EarlyStopping 回調函數，用於在訓練過程中監視 "episode/Return" 指標，當指標不再改善時停止訓練，  
# 其中 monitor='episode/Return' 表示監視回合的累計獎勵，mode='max' 表示要最大化該指標，  
# patience=500 表示在連續 500 輪中沒有改善時停止訓練。  
trainer= Trainer(  
    gpus=num_gpus,  
    max_epochs=50000,  
    callbacks=[EarlyStopping(monitor='episode/Return', mode='max', patience=500)],  
    default_root_dir=logsDir  
)  
# 開始訓練過程  
trainer.fit(algo)
```

### ▼ Check the resulting policy

```
▶ # # 顯示第 900 個視頻  
display_video(episode=900)
```



### ▼ 3.錄影講解code (20%)

#### ▼ 訓練結果展示

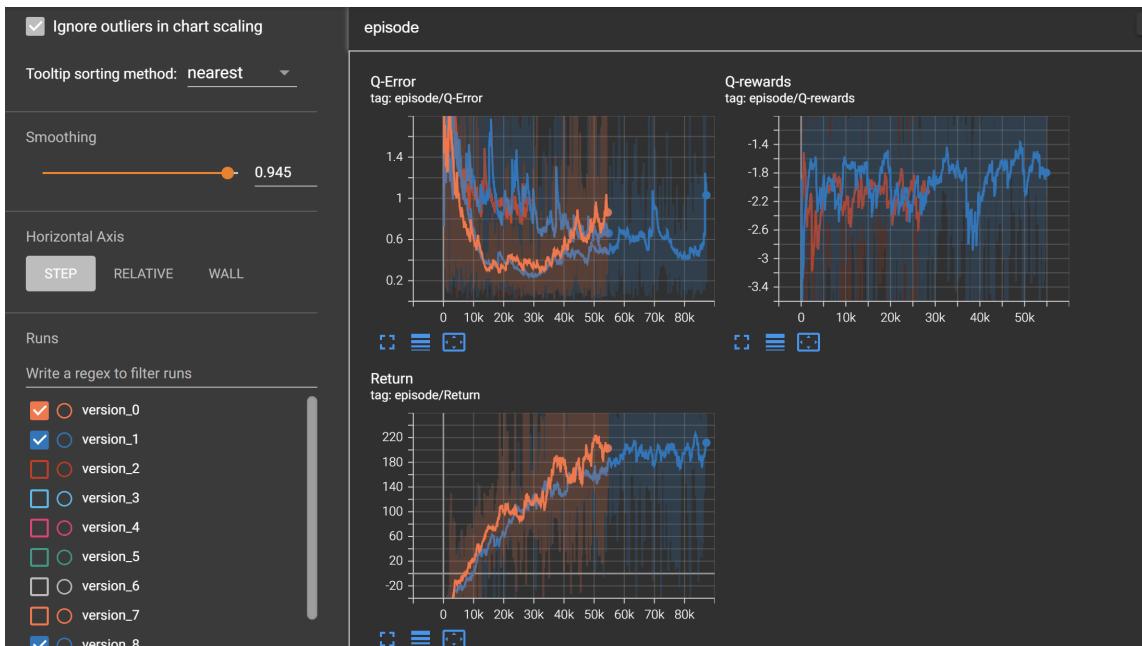
<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/2251454c-8faf-4583-8bb8-230898eb46b1/%E4%B8%8B%E8%BC%89.mp4>

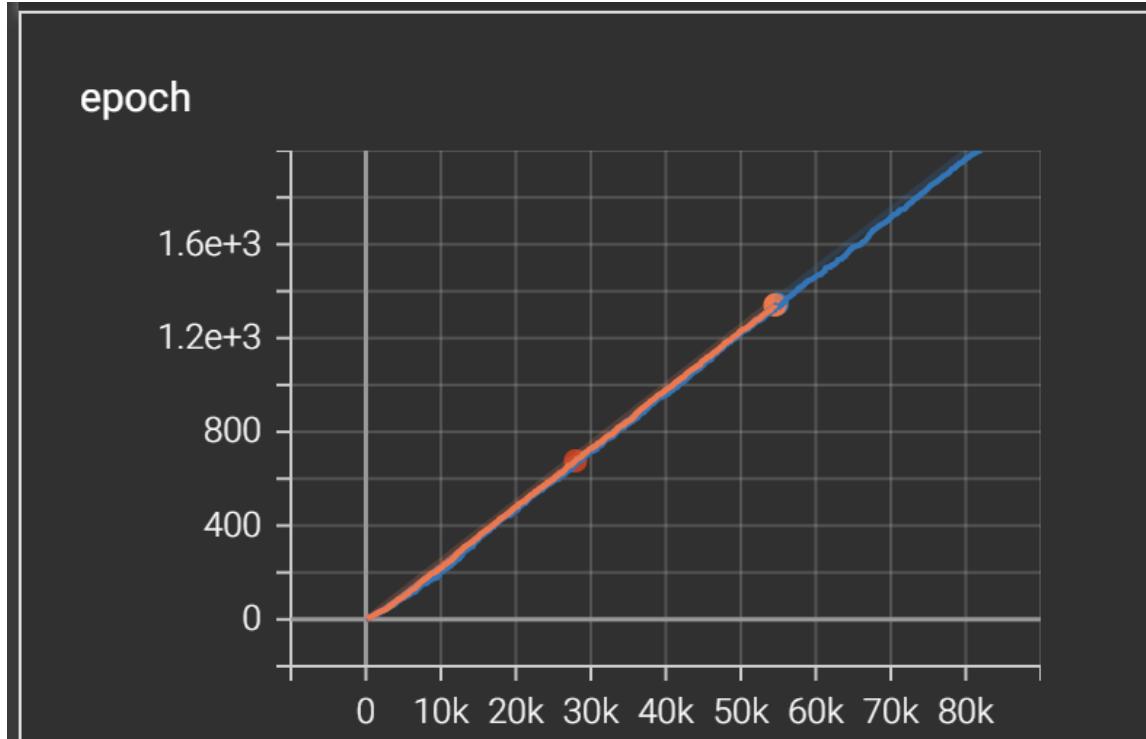
▼ 解說

<https://youtu.be/VOeS1bq8II0>

▼ 4. 測試Show出更多的實驗 on tensorboard (20%)

▼ 新增 Q-reward 每次訓練有不同版本



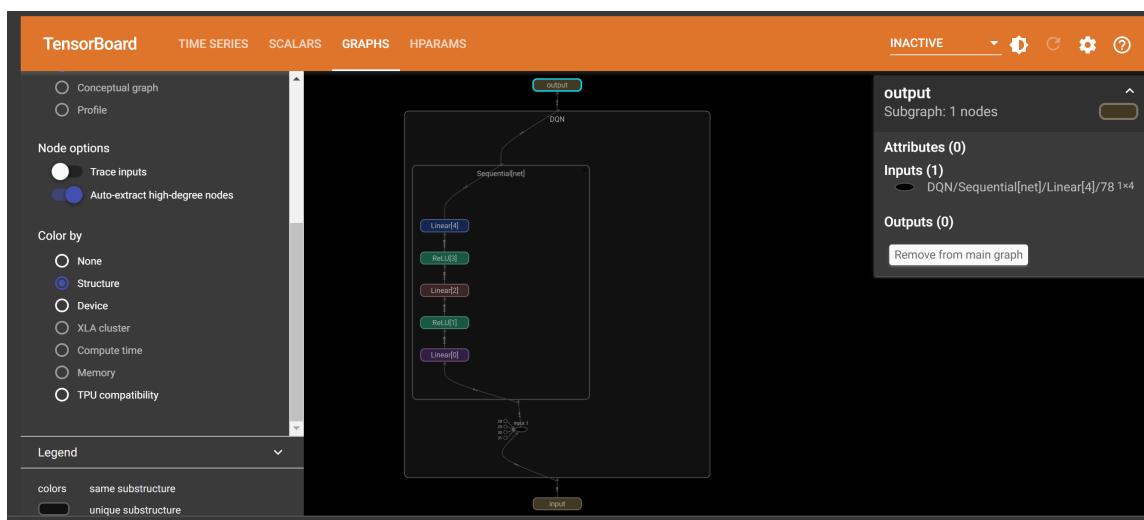


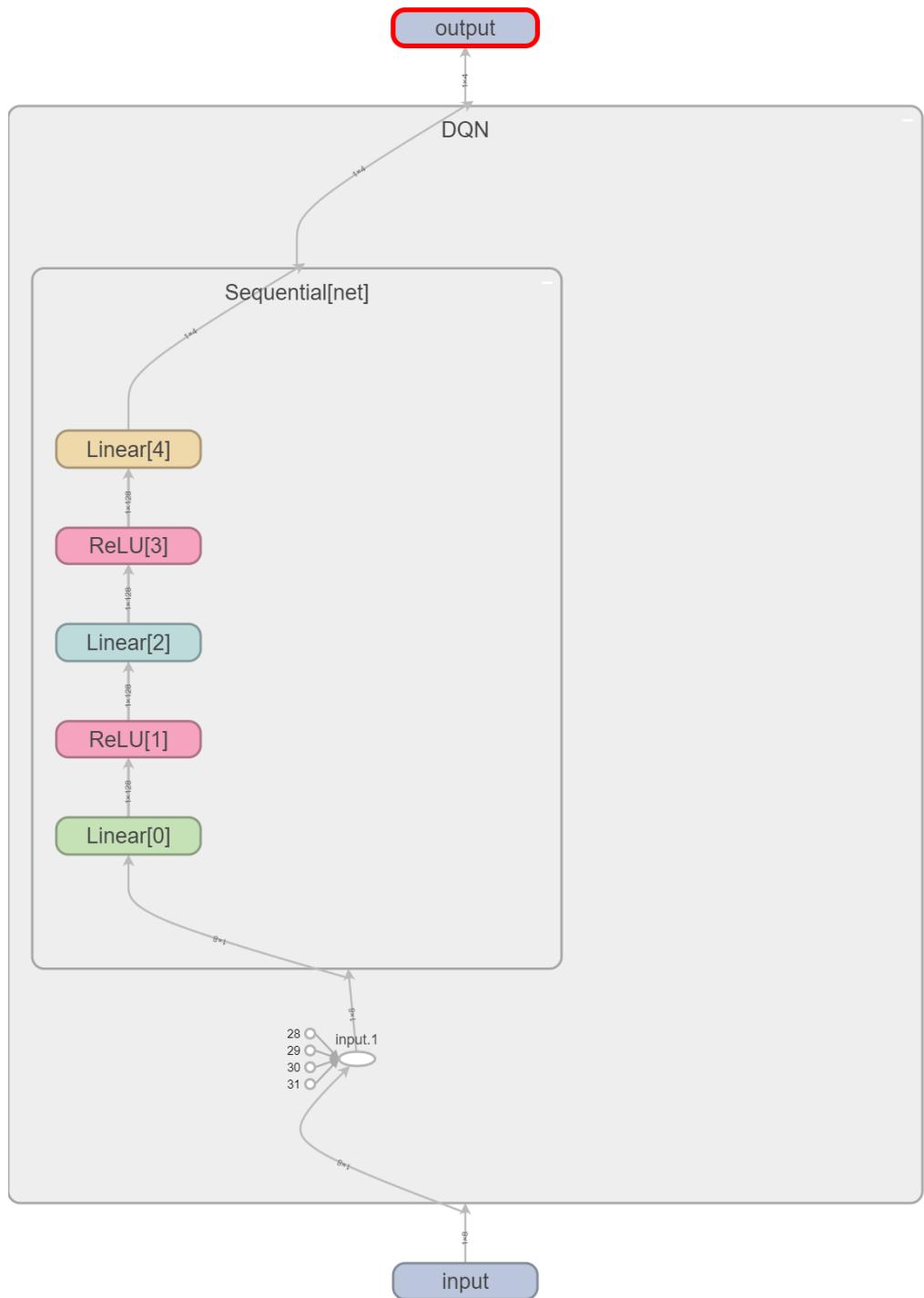
```
# training_step 定義訓練步驟，包括計算損失和更新網絡權重
def training_step(self, batch, batch_idx):
    states, actions, rewards, dones, next_states = batch
    # unsqueeze 第一個維度上進行擴展，相當於增加了一個維度
    actions = actions.unsqueeze(1)
    rewards = rewards.unsqueeze(1)
    dones= dones.unsqueeze(1)
    #next_states = next_states.unsqueeze(1)

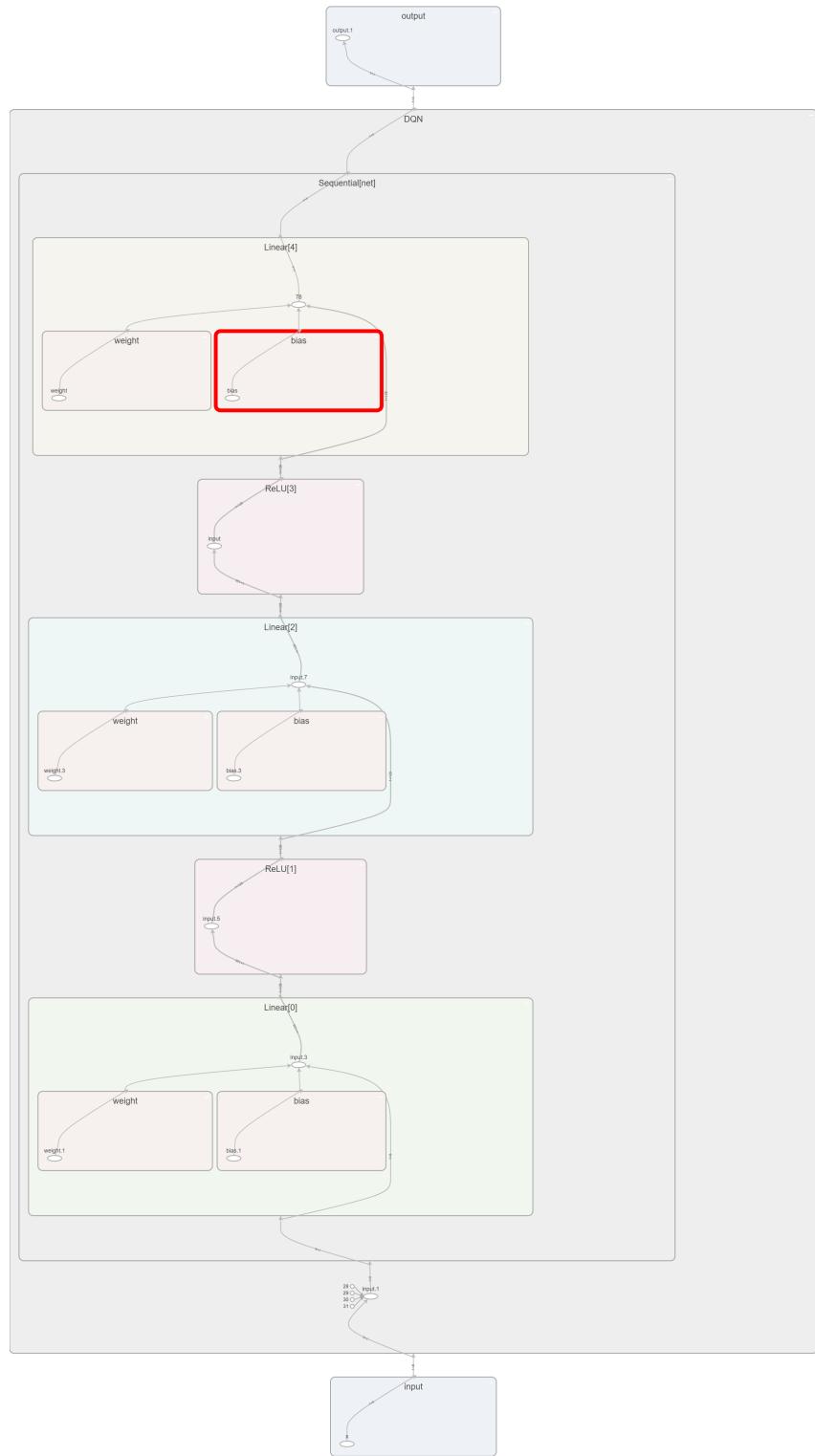
    state_action_values=self.q_net(states).gather(1, actions)
    next_action_values, _=self.target_q_net(next_states).max(dim=1,keepdim=True)
    next_action_values[dones]= 0.0

    expected_state_action_values = rewards + self.hparams.gamma * next_action_values
    loss=self.hparams.loss_fn(state_action_values, expected_state_action_values)
    # self.log('episode/Q-expected_state_action_values',expected_state_action_values.mean())
    self.log('episode/Q-rewards',rewards.mean())
    self.log('episode/Q-Error',loss)
```

▼ 新繪出神經網路圖







匯入 library

```
# 資訊寫進tensorboard 展示
from torch.utils.tensorboard import SummaryWriter
```

```
# 使用PyTorch Lightning框架實現的DQN學習模型
class DeepQLearning(LightningModule):

    # initialize
    def __init__(self, env_name, policy=epsilon_greedy, capacity=100_000, batch_size=256, lr=1e-3,
                 hidden_size=128, gamma=0.99, loss_fn=F.smooth_l1_loss, optim=AdamW,
                 eps_start=1.0, eps_end=0.15, eps_last_episode=100, samples_per_epoch=10_000, sync_rate=10):

        super().__init__()
        # 創建了一個環境
        self.env = create_environment(env_name)
        # 獲取環境觀測空間的形狀，並將其第一個維度的大小賦值給obs_size變量
        obs_size = self.env.observation_space.shape[0]
        # n_actions表示動作空間中可能的動作數量
        n_actions = self.env.action_space.n

        # 定義一個名為 q_net 的 DQN模型
        self.q_net = DQN(hidden_size, obs_size, n_actions)
        # 定義一個名為 target_q_net 的目標DQN模型
        self.target_q_net = copy.deepcopy(self.q_net)
        # 定義policy的策略函數
        self.policy = policy
        # 定義buffer的經驗回放緩衝區
        self.buffer = ReplayBuffer(capacity=capacity)
        # 超參數
        self.save_hypnerparameters()
        # 寫log進teansonboard
        self.writer = SummaryWriter(log_dir=logsDir + 'lightning logs')
        # 判斷是否有空間存放 > 有空間則填充經驗回放緩衝區
        # 經驗緩衝區 < self.hparams.samples_per_epoch 指定了每個 epoch 中需要獲得的樣本數量
        while len(self.buffer) < self.hparams.samples_per_epoch:
            print(f'{len(self.buffer)} samples in expereience buffer. Filling ...')
            self.play_episode(epsilon=self.hparams.eps_start)

    # 將神經網路圖示化在tensorboard
    def log_network_graph(self):
        # Create a dummy input tensor with appropriate size
        dummy_input = torch.zeros((1, self.env.observation_space.shape[0]))
        # Log the network graph
        self.writer.add_graph(self.q_net, dummy_input)
        self.writer.add_graph(self.target_q_net, dummy_input)

    # 創建一個名為 algo 的 DeepQLearning 算法。該算法用於訓練智能代理器學習在 "LunarLander-v2" 環境中選擇最優動作。
    algo = DeepQLearning('LunarLander-v2')
    algo.log_network_graph()
```

## ▼ 5. Optuna 優化參數(加分 20%-40%) or tensorboard

```

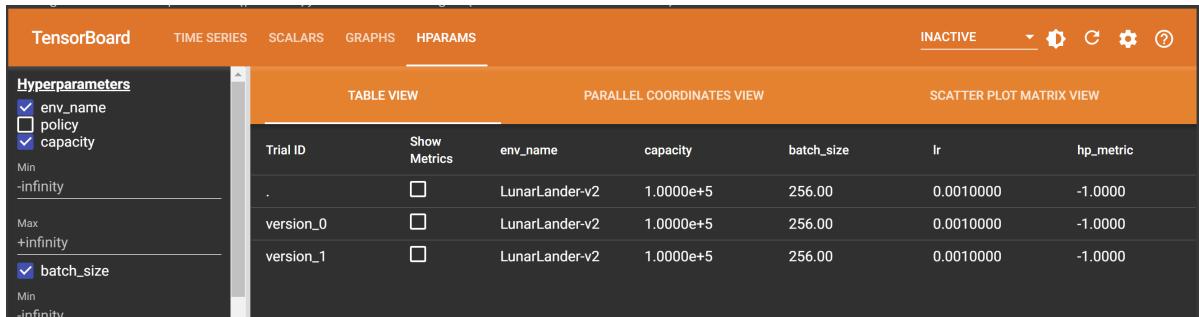
# tensorboard 超參數優化
from pytorch_lightning.loggers import TensorBoardLogger
from pytorch_lightning.utilities import rank_zero_only

logger = TensorBoardLogger(logsDir, name='lightning_logs')
trainer= Trainer(
    # gpus=num_gpus,
    max_epochs=50000,
    callbacks=[EarlyStopping(monitor='episode/Q>Error', mode='max', patience=500)],
    default_root_dir=logsDir,
    logger=logger
)

@rank_zero_only
def log_hyperparams(self):
    self.logger.log_hyperparams(self.hparams)

algo.log_hyperparams()

```



## ▼ 8 換一個game (ex: flappy bird 清大無尚鴻)

<https://colab.research.google.com/drive/1xPWIEPZKIHZS6QcSglj1Ju8UYHYCGnA4?usp=sharing>

換成 `CartPole-v0`

台車(Cartpole)是 Gym 套件內的一個遊戲，分兩個版本，`CartPole-v0` 規則如下：

1. 行動只有往左、往右兩種。
  2. 平衡桿一開始是直立(upright)，要保持它在行駛中仍然保持平衡。
  3. 下列條件會使遊戲回合結束：
    - \* 平衡桿角度偏差12度。
    - \* 台車距離中心點 2.4 單位。
    - \* 行動超過200步 (`CartPole-v1` 為 500步)。
  4. 每走一步得1分。
  5. 訓練成功的條件：連續100回合平均得分 $\geq 195$ 分 (`CartPole-v1` 為  $\geq 475$ 分)。
- \* Reinforcement Learning(強化學習)-Cartpole-v0 環境介紹  
<https://ithelp.ithome.com.tw/articles/10245605>

如何使用 DeepQLearning 算法來訓練一個代理器 (agent) 來玩 "LunarLander-v2" 遊戲，並使用 Trainer 來管理訓練過程。

```
❷ # 創建一個名為 algo 的 DeepQLearning 算法。該算法用於訓練智能代理器學習在 "LunarLander-v2" 環境中選擇最優動作。
algo = DeepQLearning('CartPole-v0')
algo.log_network_graph()

# 創建一個 Trainer，使用 gpus=num_gpus 指定使用的 GPU 數量 (num_gpus 表示 GPU 數量)。
# max_epochs=10_000 表示最大訓練輪數為 10,000 輪。
# callbacks 參數是一個列表，包含一個 EarlyStopping 回調函數，用於在訓練過程中監視 "episode/Return" 指標，當指標不再改善時停止訓練，
# 其中 monitor='episode/Return' 表示監視回合的累計獎勵，mode='max' 表示要最大化該指標，
# patience=500 表示在連續 500 輪中沒有改善時停止訓練。
trainer= Trainer(
    # gpus=num_gpus,
    max_epochs=50000,
    callbacks=[EarlyStopping(monitor='episode/Q-rewards', mode='max', patience=500)],
    default_root_dir=logsDir
)
# 開始訓練過程
trainer.fit(algo)
```

## 訓練結果

<https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e7429d7c-ecf5-4548-9021-f3a8677a3e41/CartPole-v0.mp4>