Selected Topics in Reinforcement Learning

Final Project Report: Racecar_gym

312581020 許瀚丰

Table of Contents

1. Methodology Introduction

Selected Topics in Reinforcement Learning Final Project Report

Methodology Introduction

- 在本次專題中共有兩個不同的地圖(circle & austria)，而我嘗試以下幾種知名的RL Algorithms 來解決本次專題
  - DQN
  - PPO
    - continuous version
    - discrete version
    - LSTM version
  - TD3
  - DrQ-v2(TD3 + data augmentation)

DQN

- 使用 neural net 來嘗試近似 Q value
- 使用 replay buffer 防止演算法收斂到局部最佳解
- 使用 target network 來解決在訓練中 neural net 在訓練中不穩定的問題

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

PPO (discrete version & continuous)

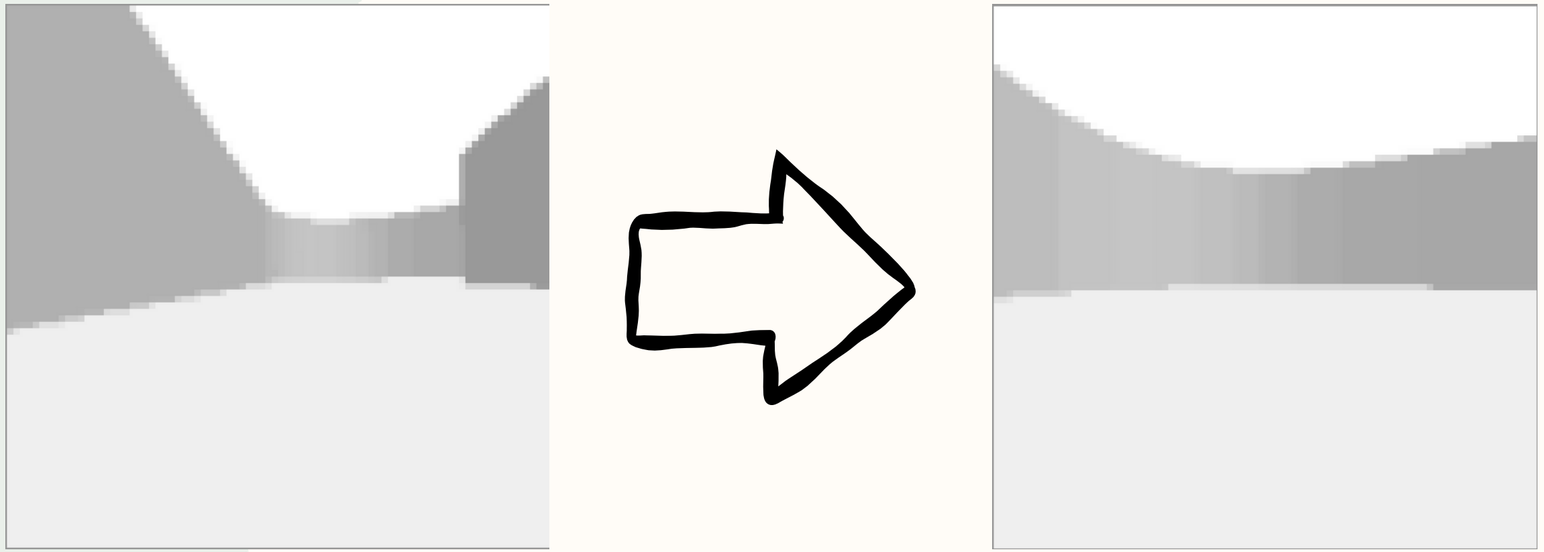
- 使用 Clipped Surrogate Objective 來取代TRPO的 Surrogate Objective ，在 training 時，將策略調整的比例設一個上下限 (1 - ε , 1 + ε)，防止新舊策略差異過大
- 而使用 continuous 的 PPO 時，則假設 neural net 的輸出是一個高斯分佈，因此輸出為 mean 與 std ，而在 decide_action 時就是從其 sample 出 action 來

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **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

PPO (LSTM)

- 在PPO中加入LSTM，希望能夠透過學習上下文的資訊來輔助 Agent 做決策
- 由於在本任務中的第二個地圖 (austria)有接近180度迴轉的髮夾彎，單純以一張影像並不能知道現在要左轉還是右轉(如圖)，因此我嘗試在 PPO 中加入 LSTM ，來讓模型可以利用前幾個 frame 的資訊來學習該如何轉彎。



TD3

- Clipped Double Q-Learning
  - 為了解決模型有高估 Q value 的問題，因此 Q value就選擇兩個 Q net 中較小的那個值
- Delayed Policy Updates
  - 讓 Policy 的更新頻率低於 Critic ，主要是讓 Policy在訓練上更加穩定(等 Critic 的 error 較小時再更新Policy)
- Target Policy Smoothing
  - 類似的 action 應該要有類似的 value 值，因此在 target action 上加上 noise，以防止 Critic network overfitting

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

DrQ-v2

- 此演算法是透過結合TD3與Data Augmentation的方式，在訓練上與TD3極為類似但在訓練時每次拿的影像資料都會經過data augmentation，因此可以極大的增強神經網路的泛化程度，是少數只需要影像就可以媲美許多 model based 的強化學習演算法之一
- 此演算法可以分為兩個部分討論，分別為
  - TD3
  - Data Augmentation

Algorithm 1 DrQ-v2: Improved data-augmented RL.

Inputs:
$f_\xi, \pi_\phi, Q_{\theta_1}, Q_{\theta_2}$: parametric networks for encoder, policy, and Q-functions respectively.
aug: random shifts image augmentation.
$\sigma(t)$: scheduled standard deviation for the exploration noise defined in Equation (3).
T, B, α, τ, c: training steps, mini-batch size, learning rate, target update rate, clip value.
Training routine:
for each timestep $t = 1..T$ **do**
  $\sigma_t \leftarrow \sigma(t)$ ▷ Compute stddev for the exploration noise
  $\boldsymbol{a}_t \leftarrow \pi_\phi(f_\xi(\boldsymbol{x}_t)) + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma_t^2)$ ▷ Add noise to the deterministic action
  $\boldsymbol{x}_{t+1} \sim P(\cdot|\boldsymbol{x}_t, \boldsymbol{a}_t)$ ▷ Run transition function for one step
  $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{x}_t, \boldsymbol{a}_t, R(\boldsymbol{x}_t, \boldsymbol{a}_t), \boldsymbol{x}_{t+1})$ ▷ Add a transition to the replay buffer
  UPDATECRITIC$(\mathcal{D}, \sigma_t)$
  UPDATEACTOR$(\mathcal{D}, \sigma_t)$
end for
procedure UPDATECRITIC$(\mathcal{D}, \sigma)$
  $\{(\boldsymbol{x}_t, \boldsymbol{a}_t, r_{t:t+n-1}, \boldsymbol{x}_{t+n})\} \sim \mathcal{D}$ ▷ Sample a mini batch of B transitions
  $\boldsymbol{h}_t, \boldsymbol{h}_{t+n} \leftarrow f_\xi(\text{aug}(\boldsymbol{x}_t)), f_\xi(\text{aug}(\boldsymbol{x}_{t+n}))$ ▷ Apply data augmentation and encode
  $\boldsymbol{a}_{t+n} \leftarrow \pi_\phi(\boldsymbol{h}_{t+n}) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2))$ ▷ Sample action
  Compute $\mathcal{L}_{\theta_1,\xi}$ and $\mathcal{L}_{\theta_2,\xi}$ using Equation (1) ▷ Compute critic losses
  $\xi \leftarrow \xi - \alpha\nabla_\xi(\mathcal{L}_{\theta_1,\xi} + \mathcal{L}_{\theta_2,\xi})$ ▷ Update encoder weights
  $\theta_k \leftarrow \theta_k - \alpha\nabla_{\theta_k}\mathcal{L}_{\theta_k,\xi} \quad \forall k \in \{1, 2\}$ ▷ Update critic weights
  $\bar{\theta}_k \leftarrow (1 - \tau)\bar{\theta}_k + \tau\theta_k \quad \forall k \in \{1, 2\}$ ▷ Update critic target weights
end procedure
procedure UPDATEACTOR$(\mathcal{D}, \sigma)$
  $\{(\boldsymbol{x}_t)\} \sim \mathcal{D}$ ▷ Sample a mini batch of B observations
  $\boldsymbol{h}_t \leftarrow f_\xi(\text{aug}(\boldsymbol{x}_t))$ ▷ Apply data augmentation and encode
  $\boldsymbol{a}_t \leftarrow \pi_\phi(\boldsymbol{h}_t) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2))$ ▷ Sample action
  Compute $\mathcal{L}_\phi$ using Equation (2) ▷ Compute actor loss
  $\phi \leftarrow \phi - \alpha\nabla_\phi\mathcal{L}_\phi$ ▷ Update actor's weights only
end procedure

DrQ-v2 (CON'T)

- TD3
  - 在DrQ-v2中，其TD3是將Delayed Policy Updates給移除的版本，我認為是因為在實作上原始的TD3是使用獨立的兩個模型作為Actor與Critic，而DrQ-v2的則相反，會使用共同的featrue extrator (image encoder)，因此兩者會共享一些資訊，因此可能可以減少訓練較不穩定問題
- Data augmentation
  - 此演算法假設類似的影像應該要有類似的 Q value ，因此就直接在輸入的影像中加上random shift(padding + crop)，並希望輸出的結果相同
  - 實際上，我認為此方法就是盡可能去增加資料的使用效率(efficiency)，一張影像就可以代表多種可能的情況，讓模型能有更好的泛化能力

The choice of methods for each map

- 由於 racecar 這個環境為 continuous action space，因此我希望可以透過 PPO，TD3，DrQ-v2三個可以適用於continuous action space的RL Agorithm 來嘗試解決本次的任務並比較其優劣勢
- 雖然此任務為continuous action space的任務，但我也透過將action設定為幾個固定的值並透過在discrete action space中最經典的DQN來嘗試解決此問題，並將其作為本次任務的比較的對象的baseline之一

2. Experiment Design and Implementation

Selected Topics in Reinforcement Learning Final Project Report

Reward Function Design

- 在reward function設計上，我參考了[1]的設計，並加上我自己設計的幾個指標
  - motor reward: 希望車子是一直往前
    - reward += 目前車子motor的大小(-1~1)
  - bang-bang problem penalty: 希望車子連續兩個frame的行為不要差太多
    - reward -= 0.1 * 目前的action與上個action(包含motor與steering)相減後取abs
  - Progress reward: 希望車子的progress一直有在進步
    - reward += 1000 * (目前的progress - 上個progress)
  - wall collision penalty: 不希望車子去撞牆，若撞牆就給penalty並直接結束
    - if wall_collision == True:
      - reward -= 100

Reward Function Design (CON’T)

- checkpoint reward: 鼓勵車子經過越多checkpoint越好
  - if checkpoint != prev_checkpoint:
    - reward += 15

```
reward = 0
if state['checkpoint'] != self.prev_info['state']['checkpoint']:
    reward += 15

reward += 1 * motor_action
reward -= 0.1 * (abs(motor_action - self.prev_info['motor']) + \
                 abs(steering_action - self.prev_info['steering']))
reward += 1000 * (state['progress'] - self.prev_info['state']['progress'])
if state['wall_collision'] == True:
    reward = -100
    done = True
self.prev_info['motor'] = motor_action.copy()
self.prev_info['steering'] = steering_action.copy()
self.prev_info['state'] = state.copy()

return obs, reward, done, truncated, state
```

Image Preprocessing

- 由於此任務的輸入為影像，因此我也對影像做了一些簡單的處理，來幫助模型進行訓練
  - GrayScale: 將影像轉為灰階
    - e.g. (3, 128, 128) -> (1, 128, 128)
  - Resize: 將影像縮小，來幫助neural net訓練
    - e.g. (1, 128, 128) -> (1, 84, 84)
  - Frame Stack: 將連續影像疊起來，讓模型可以知道連續影像之間的關係
    - 若使用PPO + LSTM則不需使用
    - e.g. (1, 84, 84) -> (8, 84, 84)

Vectorized Environments

- 在RL的演算法中，訓練資料的搜集往往是影響其演算法是否能夠成功訓練的關鍵之一，因此我嘗試讓model同時從多個環境中搜集資料，此方法不僅可以降低模型overfitting的風險，也可以增加模型的探索能力，還能夠增加整體訓練的速度
- 而在實作上，我使用的是gymnasium所提供的SubprocVecEnv，此方法可以充分的使用CPU的多顆核心，進而加速運算

```
def make_env():
    env = RaceEnv(scenario=scenario,
        render_mode='rgb_array_birds_eye',
        reset_when_collision=False)
    env = ChannelLastObservation(env)
    env = GrayScaleObservation(env, keep_dim=True)
    env = ResizeObservation(env, 84)
    return env
CPU = 40
env = SubprocVecEnv([lambda: make_env() for i in range(CPU)])
env = VecFrameStack(env, 8, channels_order='last')
```

Continuous action to Discrete action

- 為了將此任務應用於discrete PPO與DQN，我設計了四個action，其[motor, steering]分別為以下(在實驗中我也觀察到continuous的PPO最後也會收斂成以下這四個action:
  - [1, -1], [1, 0], [1, 1], [-1, 0](當motor為-1不管steering是向左或右都一樣)
- 因此我透過gymnasium提供的action wrapper將行為作轉換，如下圖所示

```
class DiscreteActionWrapper_v2(gymnasium.ActionWrapper):
    def __init__(self, env):
        super(DiscreteActionWrapper_v2, self).__init__(env)
        self.action_space = gymnasium.spaces.Discrete(4)
    def action(self, action):
        if action == 0:
            return [-1, 0.]
        elif action == 1:
            return [1., -1.]
        elif action == 2:
            return [1., 0.]
        elif action == 3:
            return [1., 1.]
        else:
            raise ValueError('Invalid action value')
```

Training Process

Stage 0

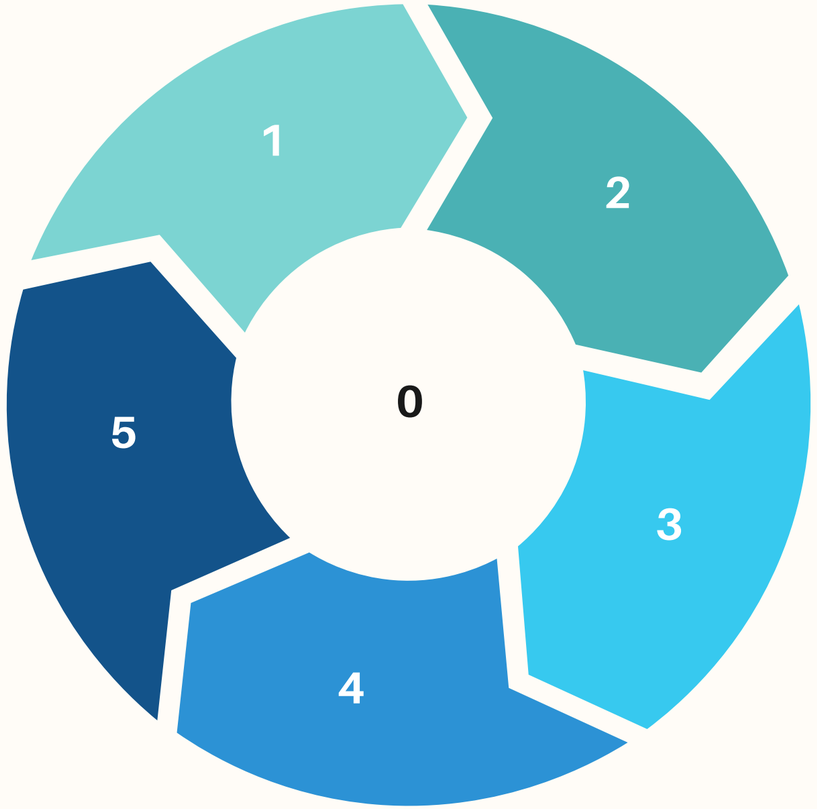
when collect enough data, update RL model

Stage 1

get observation, reward, done, truncated, info

Stage 2

image processing on observation



Stage 3

input observation to RL model

Stage 4

environment step and calculate reward by customized reward function

Stage 5

return next_observation, reward, done, truncated, info

Neural network architectures

- 在實作上，我使用Stable Baseline 3的套件來訓練神經網路，而其CNN(NatureCNN)的架構的設計如下
  - Conv2d(in_channels, 32, kernel_size=8, stride=4, padding=0)
  - ReLU()
  - Conv2d(32, 64, kernel_size=4, stride=2, padding=0)
  - ReLU()
  - Conv2d(64, 64, kernel_size=3, stride=1, padding=0)
  - ReLU()
  - Flatten()
- 而對於各自的演算法，其實作方式有些許不同，但對於Feature Extraction的部分都是先使用NatureCNN，將結果Flatten後再使用全連接來輸出最終的答案

Neural network architectures(CON'T)

- DQN
  - q_net
    - Linear(feature_dim, 64)
    - ReLU()
    - Linear(64, 64)
    - ReLU()
    - Linear(64, action_dim)

Neural network architectures(CON'T)

- PPO (share feature extraction)
  - policy(pi)
    - discrete
      - Linear(feature_dim, action_dim)
    - continuous
      - pi_mean: Linear(feature_dim, out_dim)
      - pi_std: nn.Parameter with shape = (1, out_dim)
      - output = Normal(pi_mean, pi_std)
  - value(vf)
    - Linear(feature_dim, 1)

Neural network architectures(CON'T)

- TD3(separate feature extractors)
  - policy(pi)
    - linear(feature_dim, 256)
    - ReLU()
    - linear(256, 256)
    - ReLU()
    - linear(256, out_dim)
  - value(vf)
    - 除了第一個與最後一個linear以外其他都一樣
      - First Layer: Linear(feature_dim + action_dim, 256)
      - Last Layer: Linear(256, 1)

Neural network architectures(CON'T)

- DrQ
  - image encoder
    - Conv2d(in_channel, 32, kernel_size=3, stride=2, padding=0)
    - ReLU()
    - Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
    - ReLU()
    - Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
    - ReLU()
    - Conv2d(32, 32, kernel_size=3, stride=1, padding=1)
    - ReLU()
  - TD3
    - same as SB3 TD3

Details of the hyperparameters

- 所有模型訓練的的 total_timesteps 皆為 2million
- DQN & TD3
  - 除了replay buffer size以外其餘皆與SB3的預設值相同
  - 我將buffer size設定為austria最長時間(5000 timesteps) * 50 = 250000
- PPO
  - 除了n_steps(預設為2048)以外其餘皆與SB3的預設值相同
  - 經過實驗測試發現 n_steps為1024時效果最好，因此將n_steps設定為1024

DQN Default setting: https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html
TD3 Default setting: https://stable-baselines3.readthedocs.io/en/master/modules/td3.html
PPO Default setting: https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html

Details of the hyperparameters(CON'T)

- DrQ-v2
  - 此DrQ-v2是從Lab4的TD3上做修改，超參數基本上與原始論文[2]相同，但由於我想與SB3的TD3做比較，因此在batch_size(SB3 TD3的batch_size為100)與buffer_size的部分與TD3是相同的

```
config = {
    "gpu": True,
    "training_steps": 2e6,
    "gamma": 0.99,
    "tau": 0.005,
    "batch_size": 100,
    "warmup_steps": 2000,
    "total_episode": 100000,
    "lra": 1e-4,
    "lrc": 1e-4,
    "lre": 1e-4,
    "replay_buffer_capacity": 5000 * 50,
    "logdir": 'log/DrQV2/',
    "update_freq": 1,
    "eval_interval": 10,
    "eval_episode": 10,
}
```

List of packages, tools, or resources used

- StableBaseline3(for PPO(both discrete & continuous), DQN, TD3)
  - https://stable-baselines3.readthedocs.io/en/master/
- SB3_contrib(for PPO LSTM)
  - https://stable-baselines3.readthedocs.io/en/master/guide/sb3_contrib.html
- DrQ-v2 official PyTorch implementation
  - https://github.com/facebookresearch/drqv2
- Common python packages
  - gymnasium, numPy, pyTorch, tensorboard, cv2, ...

3. Method Comparison and Evaluation

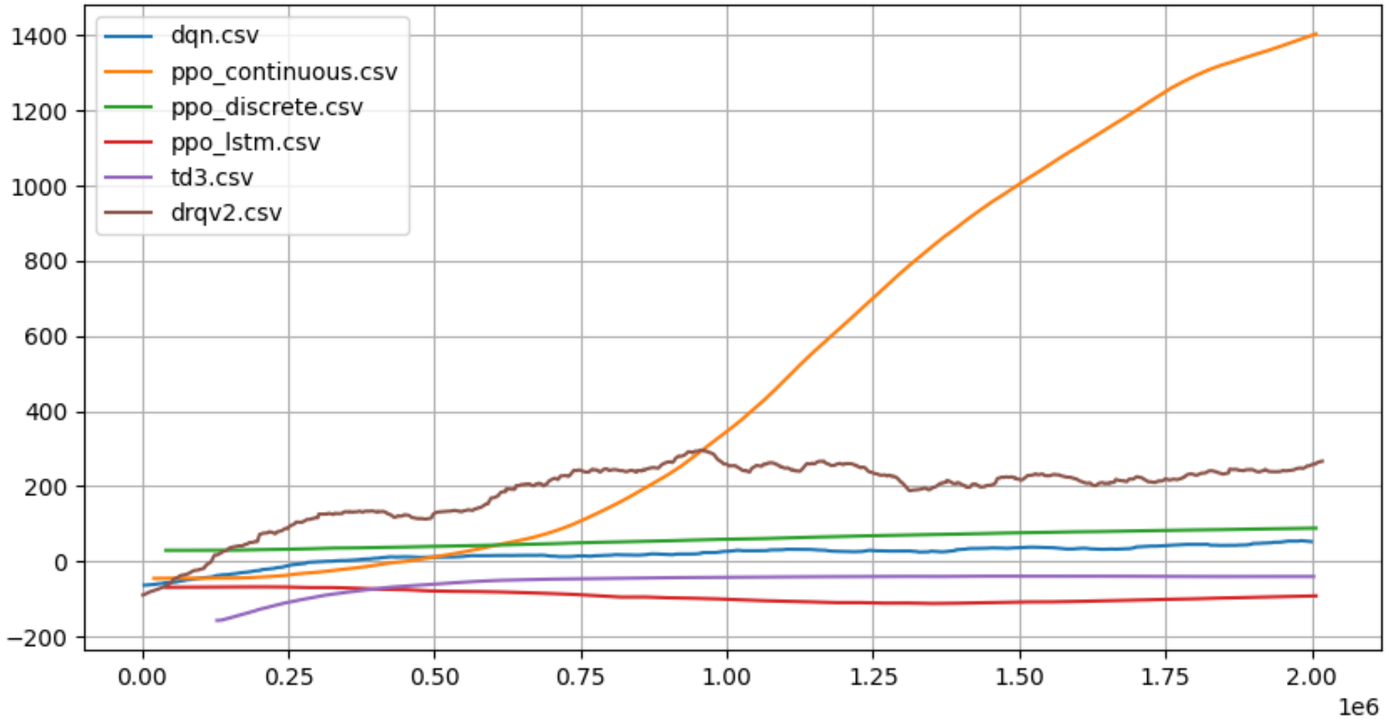
Selected Topics in Reinforcement Learning Final Project Report

Method Comparison and Evaluation

- Difference model comparison
  - DQN
  - PPO(continuous, discrete, lstm)
  - TD3
  - DrQ-v2
- hyperparameters comparison
  - PPO
    - n_steps

Different model comparison

- 在圖中可以看到，表現最好至最差依序為
  - PPO_continuous
  - DrQ-v2
  - PPO_discrete
  - DQN
  - TD3
  - PPO_lstm

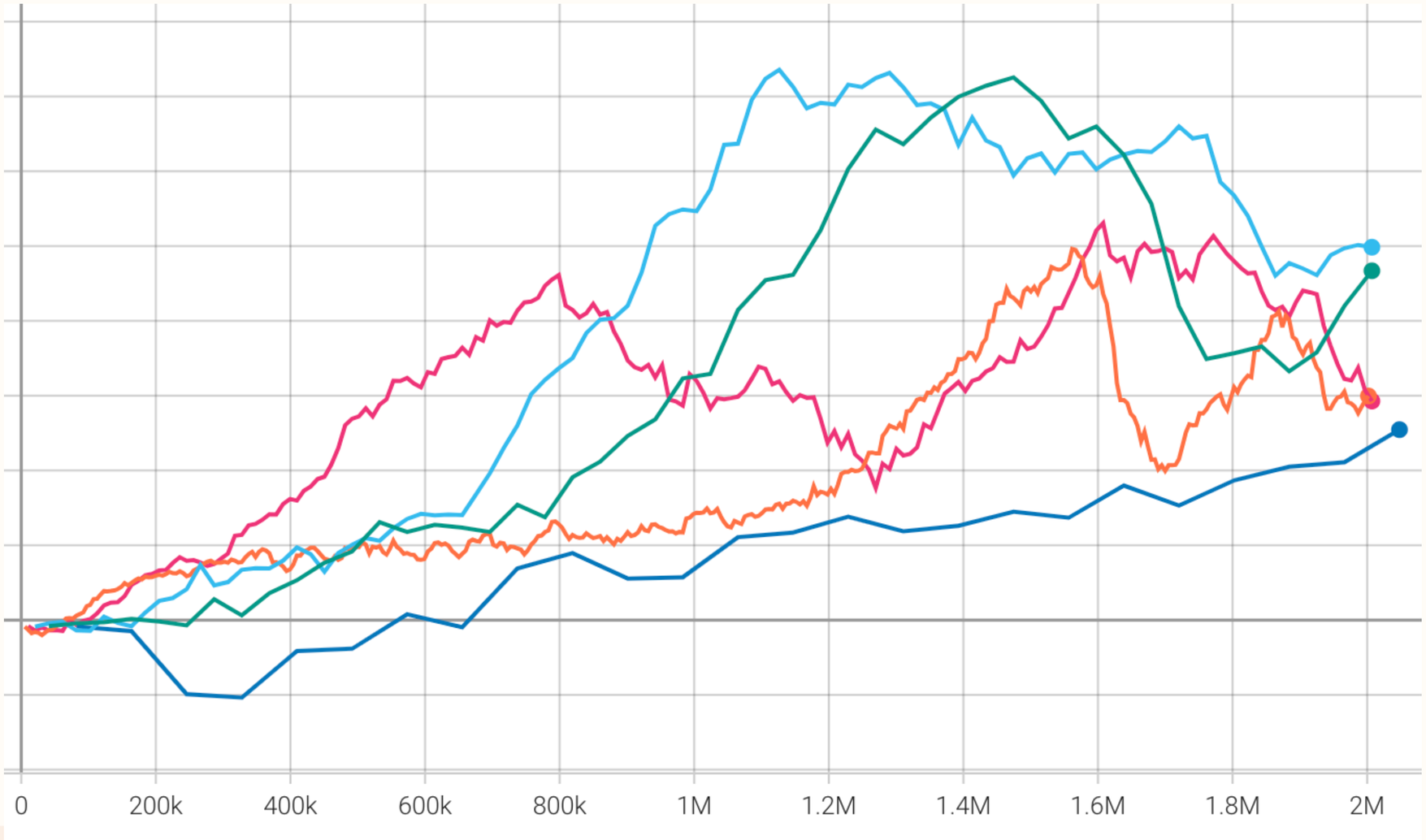


Different model comparison (CON'T)

- 在實驗中可以看到，以不同模型的比較來說：
  - PPO_continuous的表現是最佳且最穩定的，我認為其原因是PPO本身對於超參數的設定相比於其他模型更不敏感，因此即使不另外調超參數也能達到良好的效果
  - DrQ-v2次之，但訓練結果波動較大，但可以看到其相比於TD3，DrQ-v2確實能夠順利的訓練起來，且能達到不錯的效果
  - PPO_discrete相比於DQN的表現相當接近，略優於 DQN，我認為是因為其單純使用 discrete action space，可能在某些部分反而限制了模型對環境的探索能力。
  - TD3則始終無法訓練好，我認為是因為並沒有設定到適合此問題的超參數，導致其一直無法正常的進步，而是一直卡在-40左右，且幾乎沒有波動，一直卡在-35 ~ -45之間(實際標準差為8.14左右)
  - PPO_lstm則無法在只有2m內順利train起來，經過測試約在4m-6m才有辦法達到約600分

PPO with diff n_steps

- 淺藍(best)
  - 1024
- 深綠
  - 2048
- 紫紅
  - 512
- 橘色
  - 256
- 深藍
  - 4096



PPO with diff n_steps (CON'T)

- 在實驗中可以看到，以PPO的模型的比較來說：
  - 使用n_steps=1024的效果是最好的，我認為是因為我本身訓練是以平行化的方式來訓練，因此每次更新時搜集的資料共有n_envs * 1024組，且限定total_timesteps為2M的情況下，n_steps為1024在資料的搜集與效率是最為平衡的，也因此其效果是最好的
  - 而我們也可以發現n_steps=4096的效果是最差的，我認為是因為其更新的次數過少，因此若能夠將total_timesteps增加，或許其效果會更好一些

Discussing the key observations and insights

- discrete action space v.s. continuous action space
  - 對於一個continuous action space的問題來說，其實我們也可以將其model成一個discrete action space的問題，雖然最終的結果可能並不會比continuous action space來得好，但對於簡單的環境來說，確實可以加速演算法的收斂，且訓練的的結果並不會差太多
- 超參數對於訓練的影響
  - 從上述結果可以發現，想要訓練好一個好的RL演算法，超參數的設定是至關重要的，其不僅會影響訓練的效率，對於最終訓練結果也會有非常顯著的影響

4. Challenges and Learning Points

Selected Topics in Reinforcement Learning Final Project Report

Challenges and Learning Points

- Reward Function Design
- Deal with hairpin bend (髮夾彎)
- Hyperparameter tuning

Reward Function Design

- 此原始環境中，其Reward Function為目前這個frame與上個frame progress的差，但由於每個frame reward 數值非常的小(reward<0.0016)，因此critic只需要輸出非常小的數字即可，無法實際幫助訓練，導致在一開始我的模型始終訓練不起來
- 因此我開始去尋找是否有其他論文是針對此任務進行設計，最終我透過調整reward scaling(的方式再加上更多能夠幫助模型穩定訓練的指標(motor, checkpoints, ...)組合出一個適合此任務的Reward Function，最後才順利的訓練出一個成功的模型

Deal with hairpin bend

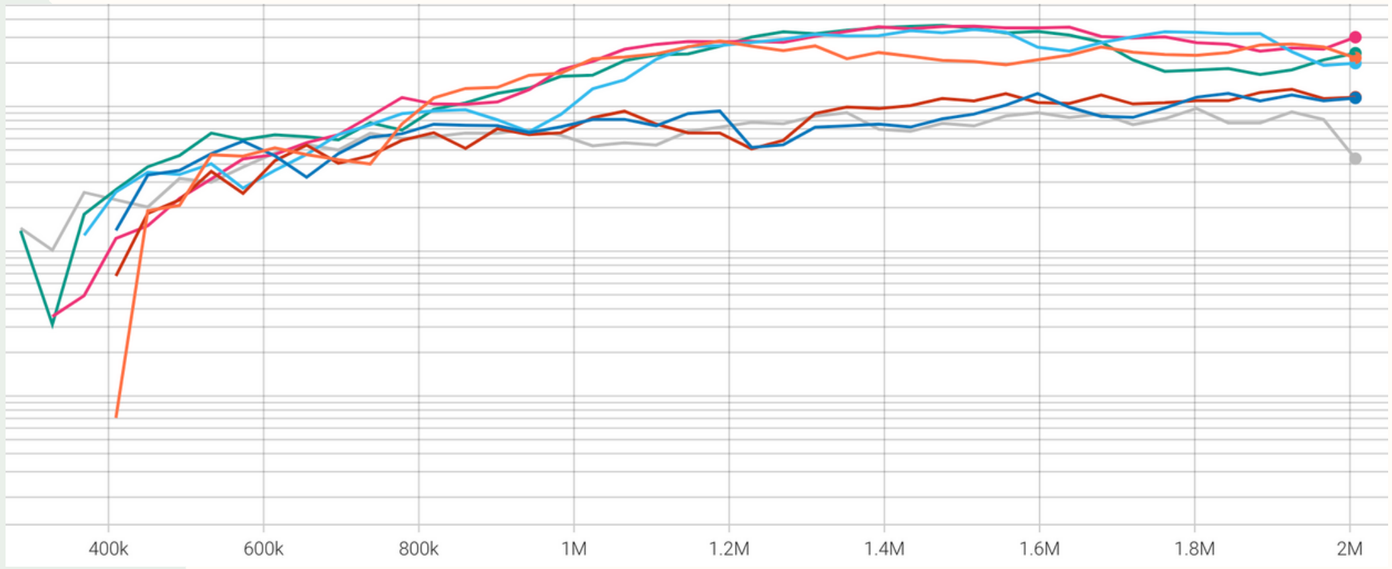
- 起初在訓練時，我一直無法成功的通過austria的第二個彎，也嘗試調整行進的速度，希望能以更穩定的方式過彎，但此方法也導致整體速度變慢，無法提高最後的總分，在經過助教們的分享後，我終於成功解決此問題
- 在助教提醒能夠透過設定將初始位置調整後，我透過vectorized environment結合隨機生成初始位置，讓模型可以從不同位置出發，極大程度的增加模型的探索程度，讓其可以順利過彎。甚至觀察到過完時模型並沒有透全程減速的方式，而是透過將motor從1直接變成-1，類似甩尾的方式過彎，真是有趣！

Hyperparameter tuning

- 在訓練任何深度學習演算法時，超參數往往會是影響模型效能的關鍵，在RL中也不例外。在我自己的測試中，我都是先使用原始論文給定的參數訓練作為baseline，在透過個別調整超參數來確定，而我在訓練PPO時，發現到其超參數最影響到效能的便是n_steps，由於不同任務的total_timesteps不相同，導致要決定何時更新變成極為重要的因素，因此我將所有我認為合理的n_steps size皆試過一次(256, 512, 1024, 2048, 4096)，最後發現雖然1024在平均上是最優的，但其實也並非每次跑出來的結果都是類似(例如下一頁的圖)，只能透過多次嘗試並找到最適合的結果

Hyperparameter tuning(CON'T)

- 如圖可以發現，每條線皆是PPO且使用n_steps為1024做訓練，但結果仍然差異很大，可見模型的初始化也對於訓練上的影響很大

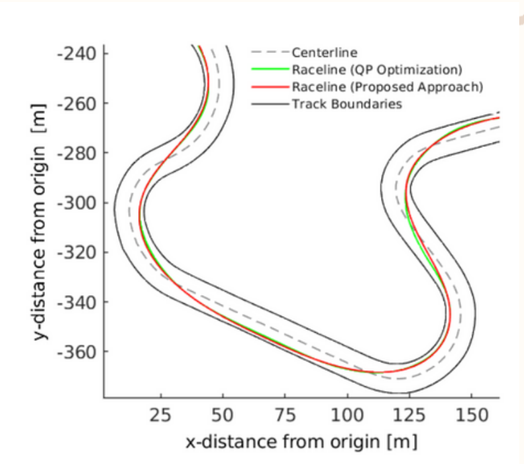


5. Future Work

Selected Topics in Reinforcement Learning Final Project Report

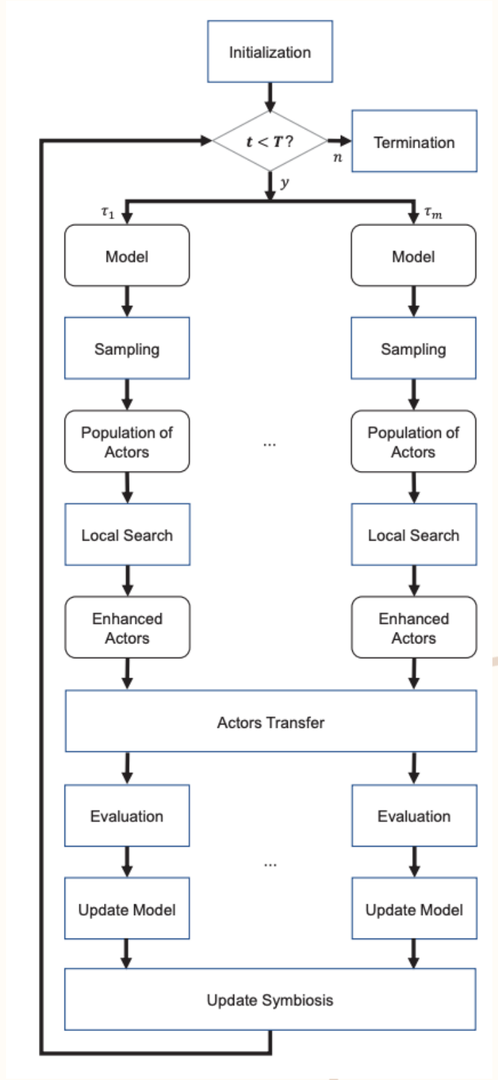
Future Work - Reward Function Redesign

- Minimum Curvature Path(最小曲率路徑)
  - 此方法若以賽車作為例子，就是盡可能地在行進過程中最小化橫向的加速度，讓賽車可以一直以高速行駛，例如常常聽到的外內外(外側進彎，在轉彎靠進彎道中線時盡量靠內，最後再靠外側出彎)，就是一個極佳的例子
  - 因此我會嘗試將地圖的Minimum Curvature Path給找出來，並透過修改Reward來將賽車的路線盡量與Minimum Curvature Path靠近，以此就能盡可能的　減少跑完一個lap所需的時間



Future Work - multitask learning

- 在訓練時我並沒有一個很好方法讓一個模型能夠同時跑好兩張地圖，而我目前的做法自己為使用兩個地圖(circle and austria)以1:9的比例一起作為訓練資料，但結果反而不如只訓練austria後同時測試兩個環境的結果
- 因此未來我會嘗試透過使用[5]的方法，此方法是透過將evolutionary algorithm加入RL演算法中，讓兩個環境能夠分享資訊並透過共演化actor的方式學習，最後就能夠得到一個泛化程度更高的模型



6. Reference

Selected Topics in Reinforcement Learning Final Project Report

Reference

[1] Residual Policy Learning Facilitates Efficient Model-Free Autonomous Racing:

https://arxiv.org/pdf/1312.5602.pdf

[2] Mastering Visual Continuous Control: Improved Data-Augmented Reinforcement Learning:

https://arxiv.org/pdf/2107.09645.pdf

[3] Minimum curvature trajectory planning and control for an autonomous race car:

https://www.tandfonline.com/doi/epdf/10.1080/00423114.2019.1631455?needAccess=true

[4] A Fast Approach to Minimum Curvature Raceline Planning via Probabilistic Inference:

https://arxiv.org/pdf/2203.03224.pdf

[5] Evolutionary Multitask Reinforcement Learning Using Symbiosis in Biocoenosis Optimization:

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10254037