一、 Introduction

本實驗探討在強化學習中使用 A2C 和 PPO 演算法的實現與效果，並將其應用在 Pendulum 與 Walker2d 環境進行實驗。在 Task1 當中為基本的 A2C 實作，而 Task2 和 Task3 詳細說明 PPO 在 Pendulum 和 Walker2d 環境中的實現，同時包含 GAE 估計等關鍵技術。並且在最後顯示所有的實驗模型表現。

二、 Implementation

1. Stochastic policy gradient and the TD error

```python
def select_action(self, state: np.ndarray) -> np.ndarray:
    """Select an action from the input state."""
    state = torch.FloatTensor(state).to(self.device)
    action, dist = self.actor(state)
    selected_action = dist.mean if self.is_test else action

    if not self.is_test:
        log_prob = dist.log_prob(selected_action).sum(dim=-1)
        self.transition = [state, log_prob, dist]

    return selected_action.clamp(-2.0, 2.0).cpu().detach().numpy()
```

```python
policy_loss = -(log_prob * advantage) - self.entropy_weight * dist.entropy()
```

Stochastic policy gradient 通過 Actor 輸出平均值和標準差，並採樣動作以計算 log_prob 作為策略梯度的一部分。

```
next_state = torch.tensor(next_state, dtype=torch.float32, device=self.device)
reward = torch.tensor(reward, dtype=torch.float32, device=self.device)
mask = torch.tensor(1 - done, dtype=torch.float32, device=self.device)
value = self.critic(state)
next_value = self.critic(next_state)
td_target = reward + self.gamma * next_value * mask

############TODO#############
# value_loss = ?
value_loss = F.smooth_l1_loss(value, td_target.detach())
############################

# update value
self.critic_optimizer.zero_grad()
value_loss.backward()
torch.nn.utils.clip_grad_norm_(self.critic.parameters(), max_norm=1.0)
self.critic_optimizer.step()

# advantage = Q_t - V(s_t)
############TODO#############
# policy_loss = ?
advantage = (td_target - value).detach()
policy_loss = -(log_prob * advantage) - self.entropy_weight * dist.entropy()
```

TD error 透過 Critic 預測的 $V(S_t)$和 reward + gamma * $V(S_{t+1})$之間的誤差計算得來。

2. Clipped objective in PPO

```
_, dist = self.actor(state)
log_prob = dist.log_prob(action)
ratio = (log_prob - old_log_prob).exp()

# actor_loss
############TODO#############
# actor_loss = ?
surr1 = adv * ratio
surr2 = adv * torch.clamp(ratio, 1 - self.epsilon, 1 + self.epsilon)
policy_loss = -torch.min(surr1, surr2).mean()
entropy = dist.entropy().mean()
actor_loss = policy_loss - self.entropy_weight * entropy
```

使用 torch.clamp 將新舊策略比率限制在[1-ε, 1+ε]範圍內，然後乘上 advantages 並與未做 clamp 的的版本取 min 再取負號，作為 policy_loss 用來更新 Actor。

3. Estimator of GAE

```python
def compute_gae(
    next_value: list, rewards: list, masks: list, values: list, gamma: float, tau: float) -> List:
    """Compute gae."""

    ############TODO#############
    gae_returns = list()
    values.append(next_value)
    sum = 0
    for i in range(len(rewards)-1, -1, -1):
        td_error = rewards[i] + gamma * values[i+1] * masks[i] - values[i]
        sum = sum * (gamma * tau) * masks[i] + td_error
        gae_returns.append(sum + values[i])
    gae_returns.reverse()
    ############################
    return gae_returns
```

透過 compute_gae 來取得其估計值，公式為 $GAE = \delta_t + gamma * \tau * mask * GAE_{t+1}$

其中 δ 為 TD error。

4. collect samples from the environment

```python
def select_action(self, state: np.ndarray) -> np.ndarray:
    """Select an action from the input state."""
    state = torch.FloatTensor(state).to(self.device).unsqueeze(0)
    action, dist = self.actor(state)
    selected_action = dist.mean if self.is_test else action

    if not self.is_test:
        value = self.critic(state)
        self.states.append(state)
        self.actions.append(selected_action)
        self.values.append(value)
        self.log_probs.append(dist.log_prob(selected_action))

    return selected_action.cpu().detach().numpy()

def step(self, action: np.ndarray) -> Tuple[np.ndarray, np.float64, bool]:
    """Take an action and return the response of the env."""
    action = action.flatten()
    next_state, reward, terminated, truncated, _ = self.env.step(action)
    done = terminated or truncated
    next_state = np.reshape(next_state, (1, -1)).astype(np.float64)
    reward = np.reshape(reward, (1, -1)).astype(np.float64)
    done = np.reshape(done, (1, -1))

    if not self.is_test:
        self.rewards.append(torch.FloatTensor(reward).to(self.device))
        self.masks.append(torch.FloatTensor(1 - done).to(self.device))

    return next_state, reward, done
```

使用 step 取得環境的所有資訊，並透過 Action 和 Critic 模型取得當前的 action 和 value 並全部記錄起來以方便做 PPO 的模型更新。

5. enforce exploration

```
entropy = dist.entropy().mean()
actor_loss = policy_loss - self.entropy_weight * entropy
```

在 Action 模型的 loss 計算中，多增加了 entropy 的 loss，-entropy_weight * dist.entropy()，以此來鼓勵 Action 輸出的 std 不要太小，藉此保持搜索的力道。

6. Weight & Bias

```
while not done:
    # self.env.render()  # Render the environment
    action = self.select_action(state)
    next_state, reward, done = self.step(action)
    actor_loss, critic_loss = self.update_model()
    actor_losses.append(actor_loss)
    critic_losses.append(critic_loss)

    state = next_state
    score += reward
    step_count += 1
    # W&B logging
    wandb.log({
        "step": step_count,
        "actor loss": actor_loss,
        "critic loss": critic_loss,
        })
```

```python
if done:
    scores.append(score)
    # W&B logging
    wandb.log({
        "episode": ep,
        "return": score,
        })
```

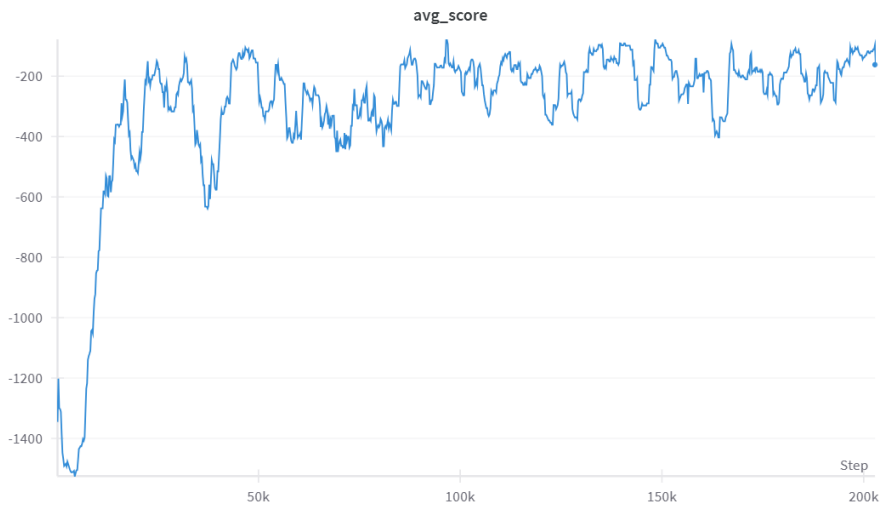
```python
if avg_score > best_score:
    best_score = avg_score
    self.save_model(best_model_path)
wandb.log({"avg_score": avg_score})
wandb.log({"best_avg_score": best_score})
```

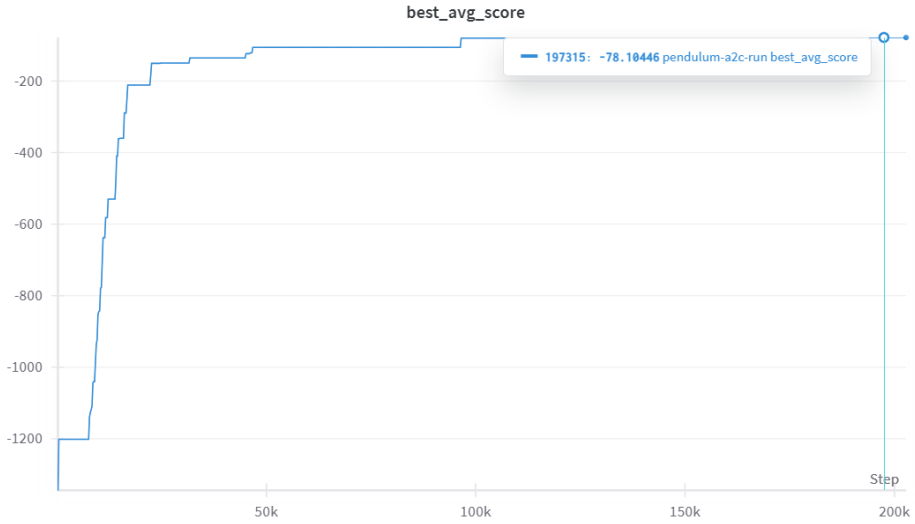
使用 wandb.log 記錄每步的 actor loss、critic loss、reward、episode 數等指標。除此之外，在每一次episode之後執行test並且也將 test 的分數和過往 20 場的平均分數記錄在 wandb 當中以便觀察。

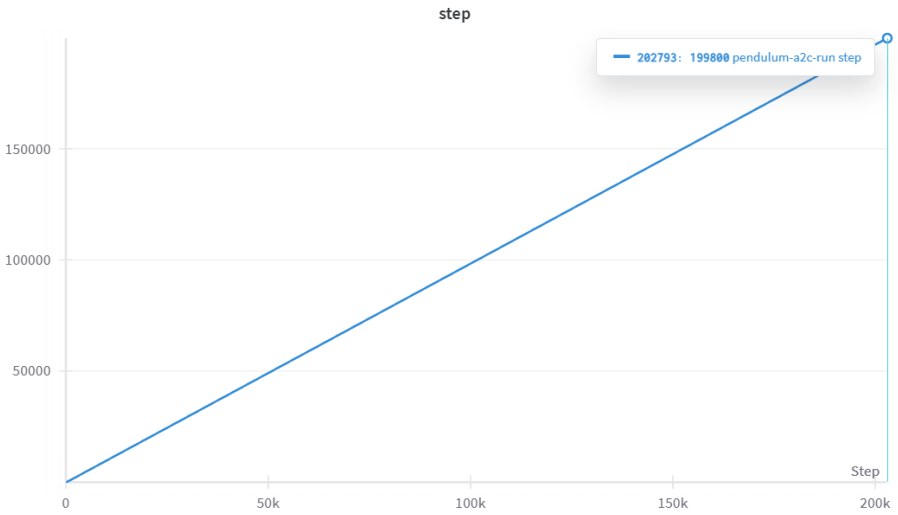
三、 Analysis and discussions

1. training curves

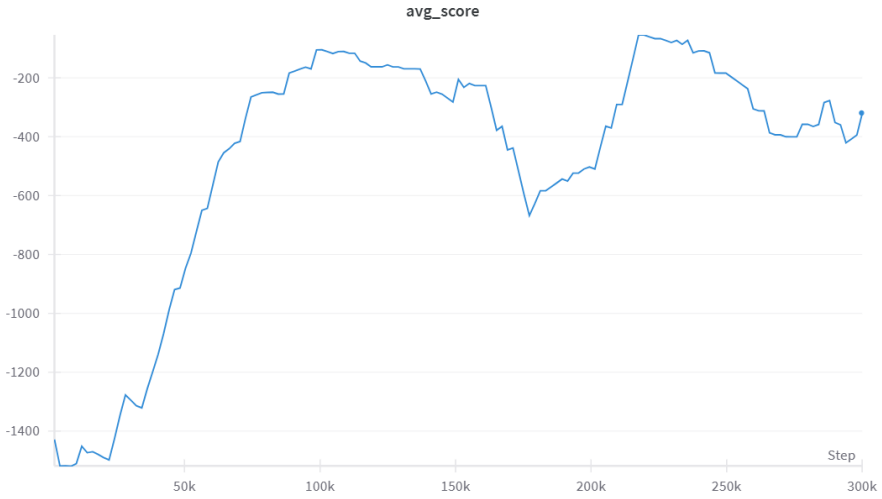
A. Task 1

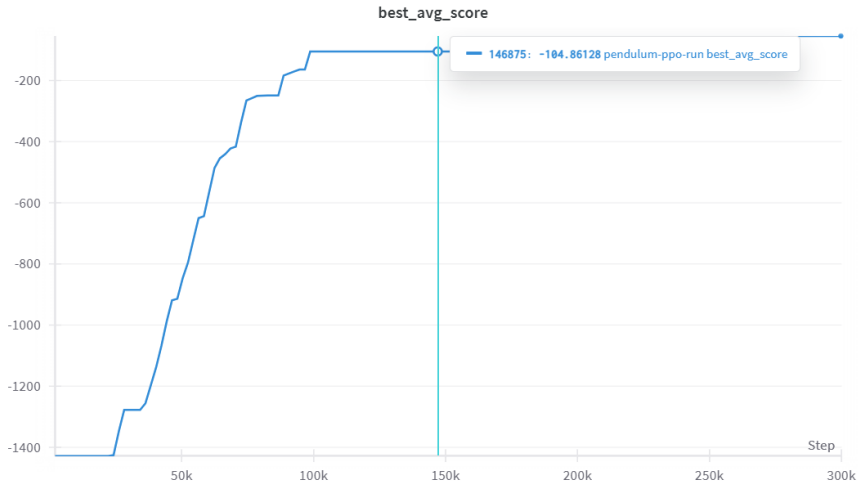


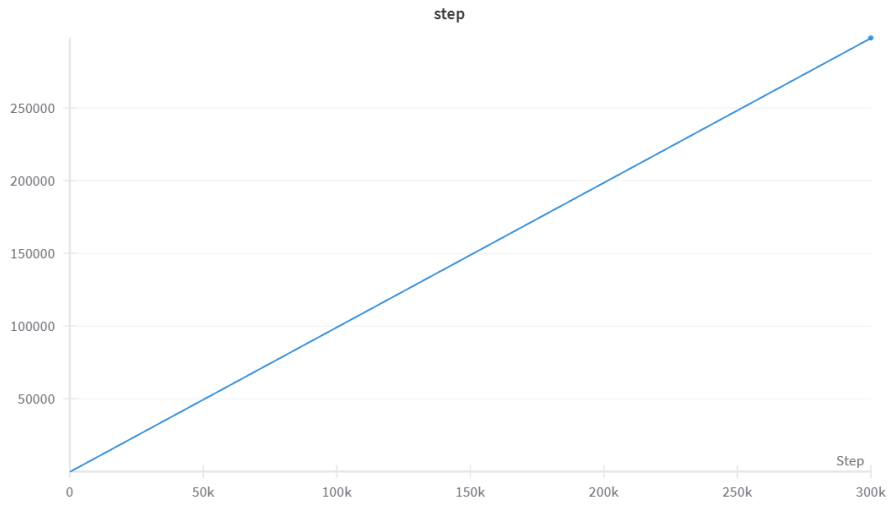

best_avg_score
197315: -78.10446 pendulum-a2c-run best_avg_score
-200
-400
-600
-800
-1000
-1200
Step
50k
100k
150k
200k


step
202793: 199800 pendulum-a2c-run step
150000
100000
50000
Step
0
50k
100k
150k
200k

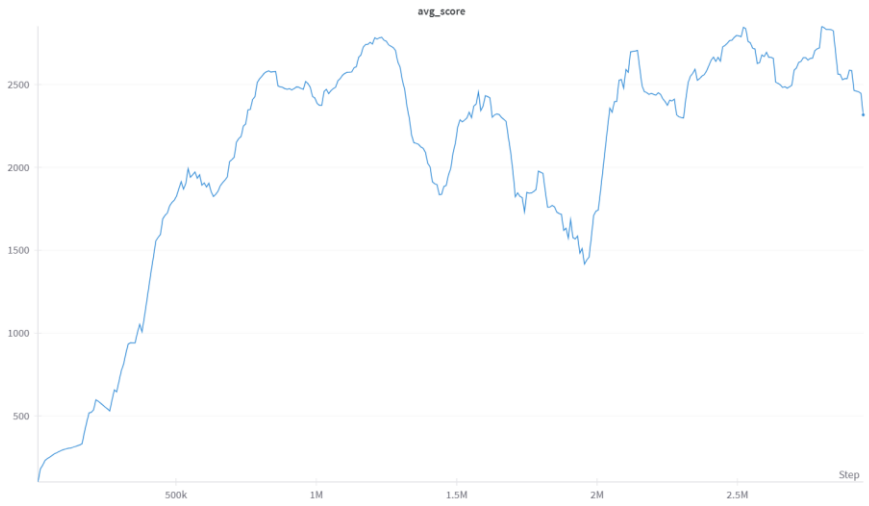
B. Task 2

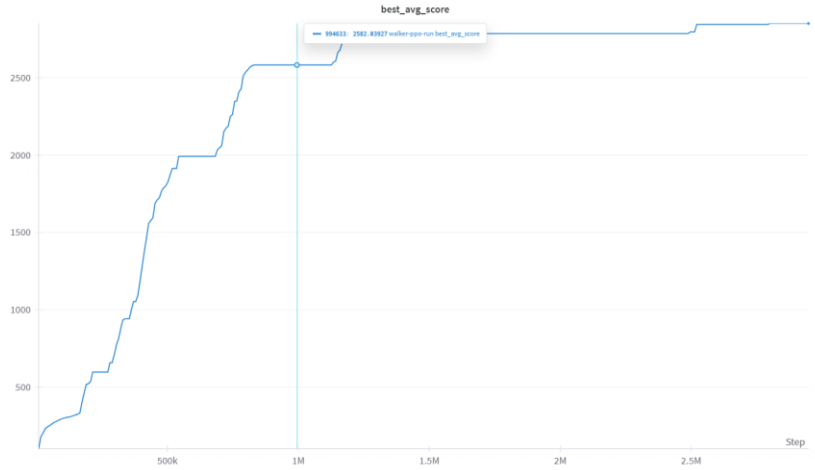

avg_score
-200
-400
-600
-800
-1000
-1200
-1400
Step
50k
100k
150k
200k
250k
300k

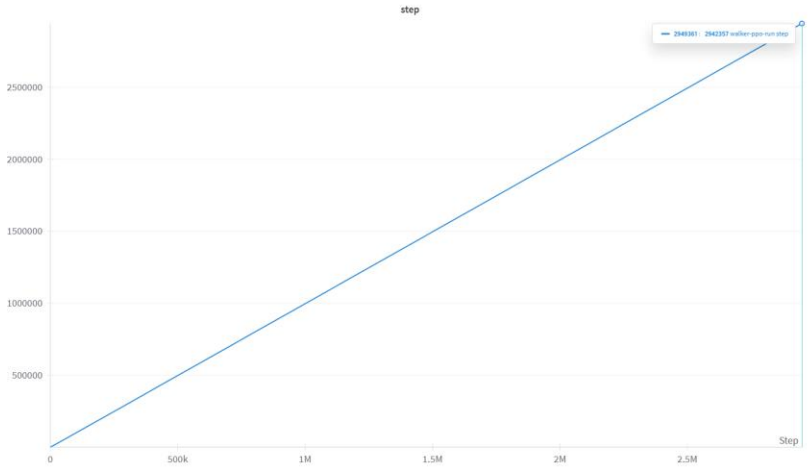

best_avg_score
146875: -104.86128 pendulum-ppo-run best_avg_score
-200
-400
-600
-800
-1000
-1200
-1400
Step
50k
100k
150k
200k
250k
300k


step
250000
200000
150000
100000
50000
Step
0
50k
100k
150k
200k
250k
300k

C. Task 3


avg_score
2500
2000
1500
1000
500
Step
500k
1M
1.5M
2M
2.5M





D. Conclusion

Task 1、Task 2 可以在 200K 的 environment step 之內達到平均-150 分以上。

Task 3 可以在 1M 之內達到平均 2500 分以上。

所有 Task 所使用的 seed:

[4140, 5339, 3232, 3940, 3164, 1885,

4789, 7802, 9140, 3896, 2383, 9107, 8202, 39, 4586, 464, 8145, 2829, 3133, 8311]

2. Compare the sample efficiency and training stability of A2C and PPO
   A. Sample efficiency

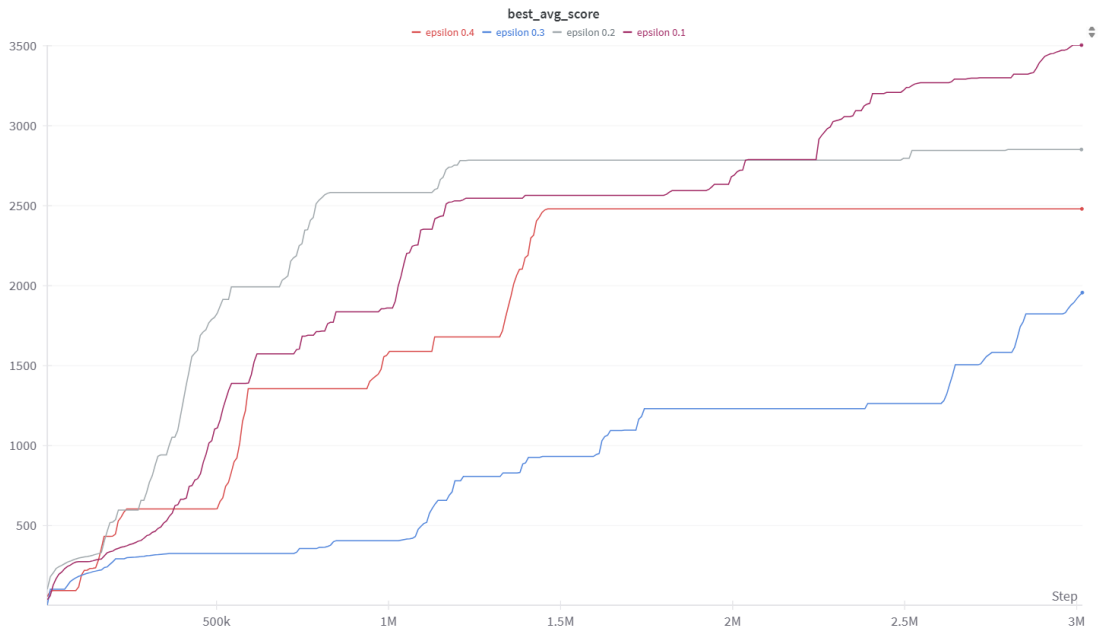
      PPO 透過多次更新 update_epoch=10 以重複利用 sample，尤其在 Walker2d 較難的環境中表現更好。A2C 即時更新因為樣本利用率低，每次只會使用一次用來更新模型，適合較簡單環境，但效率較差。

   B. Training stability

      PPO 的訓練穩定性優於 A2C，通過限制新舊策略比率在[1-ε, 1+ε]之間和 GAE 估計，控制模型的更新幅度，因此表現會較於平滑。而 A2C 因為每次會直接使用剛剛取得的 sample 做更新，在模型的更新上波動較大。

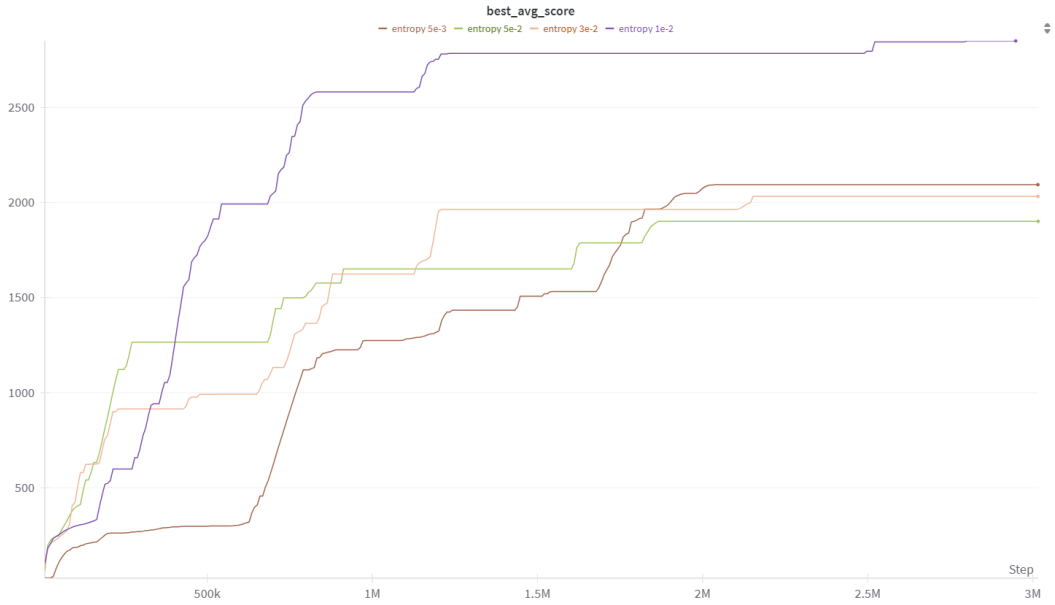
3. Empirical study

A. Clipping parameter



在所有的 hyper parameter 都不變的情況下，只更改 epsilon (Clipping)的參數，可以發現對於 Lab 的要求，使用 epsilon 為 0.2 的參數在 1M 的 environment step 之前表現最好，因此選用此超參數作為最終模型的訓練，但是 epsilon 為 0.1 的實驗中，在後期能夠達到 3500 的高分，顯示若只論模型表現的情況下，將 epsilon 設為 0.1 對此環境較為適合。

B. Entropy coefficient



根據先前 Clipping parameter 的實驗，選用 epsilon 為 0.2 為基礎，其他參數皆不便的情況下，只更改 entropy weight 可以發現，將 entropy weight 設為 1e-2 的表現效果遠高於其他版本。

四、 Execute code

1. Training

   python a2c_pendulum.py

   python ppo_pendulum.py

   python ppo_walker.py

2. Testing (and generate video)

python a2c_pendulum.py -p “model path”

python ppo_pendulum.py -p “model path”

python ppo_walker.py -p “model path”