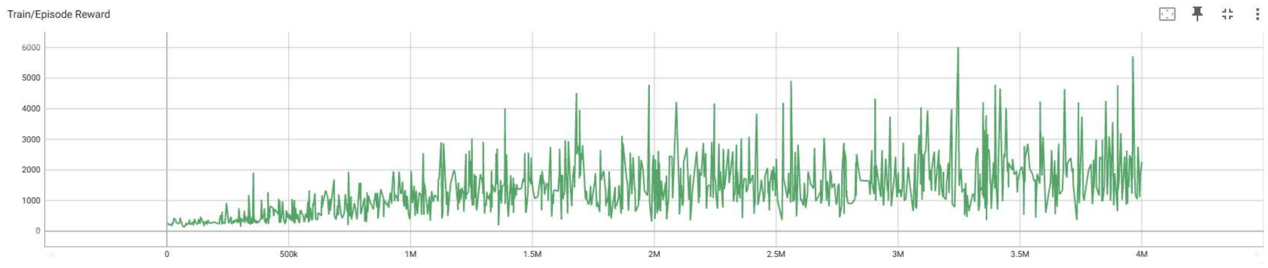
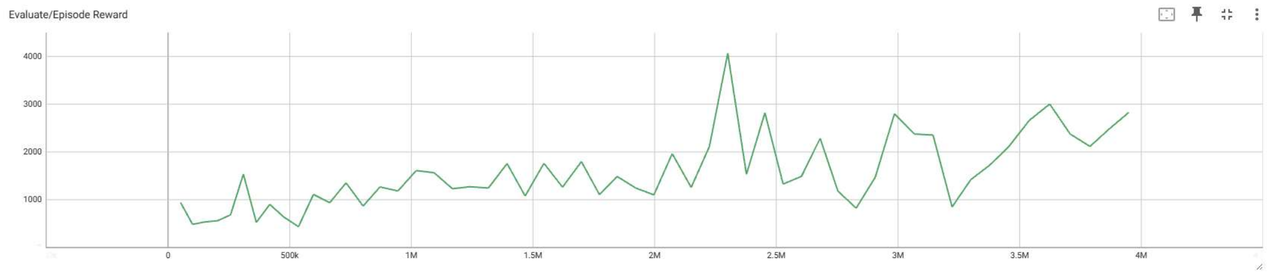
LAB2

Screenshot of Tensorboard training curve and testing results on DQN.

DQN training curve



DQN testing curve

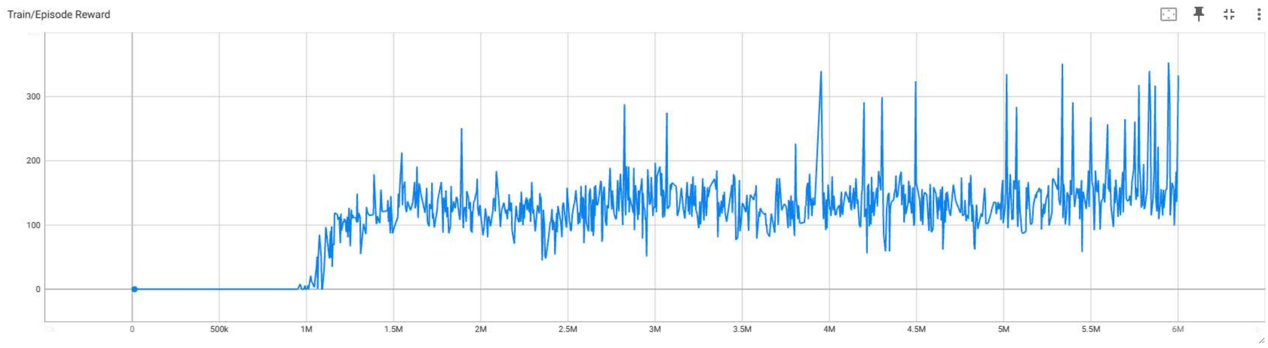


DQN testing result

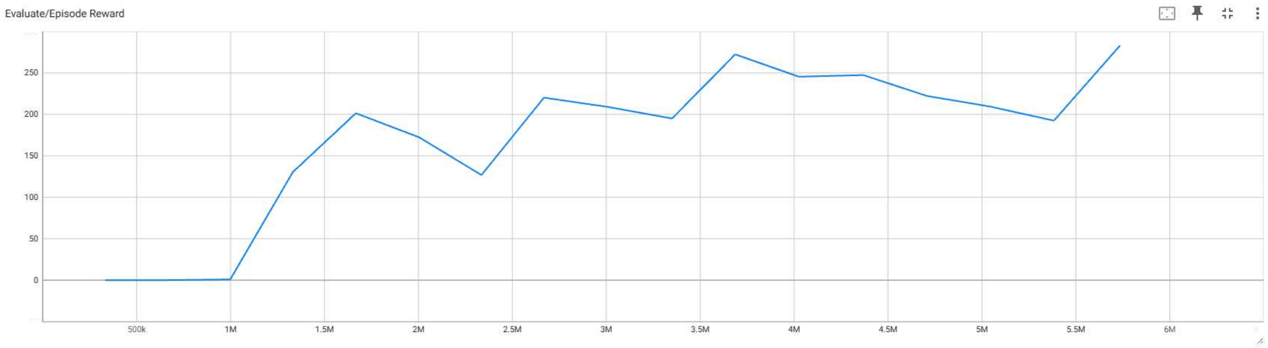
```
episode 1 reward: 3980.0
episode 2 reward: 3980.0
episode 3 reward: 4690.0
episode 4 reward: 4140.0
episode 5 reward: 4100.0
average score: 4178.0
```

Screenshot of Tensorboard training curve and testing results on Enduro-v5 using DQN

DQN training curve



DQN testing curve



DQN testing result

```
episode 1 reward: 429.0
episode 2 reward: 354.0
episode 3 reward: 322.0
episode 4 reward: 332.0
episode 5 reward: 437.0
average score: 374.8
```

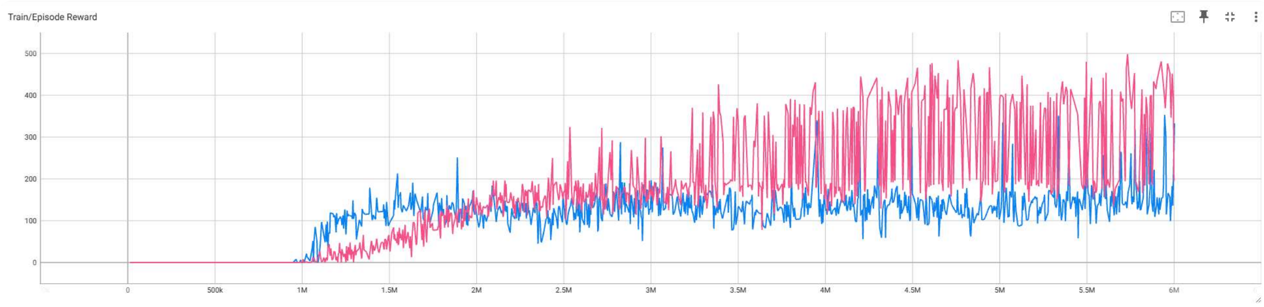
DuelingDQN testing result

- 因為 DQN 好像有點爛，所以多測試了 Dueling DQN

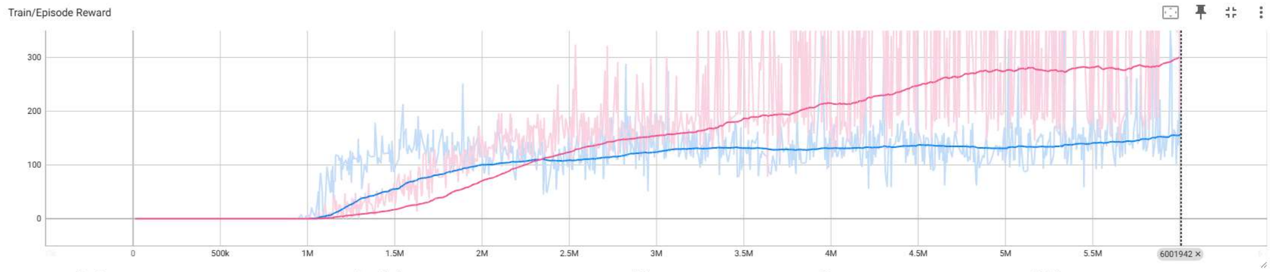
  ```
  episode 1 reward: 436.0
  episode 2 reward: 763.0
  episode 3 reward: 793.0
  episode 4 reward: 480.0
  episode 5 reward: 691.0
  average score: 632.6
  ```

DQN V.S Dueling DQN training curve

- DQN
- Dueling_DQN

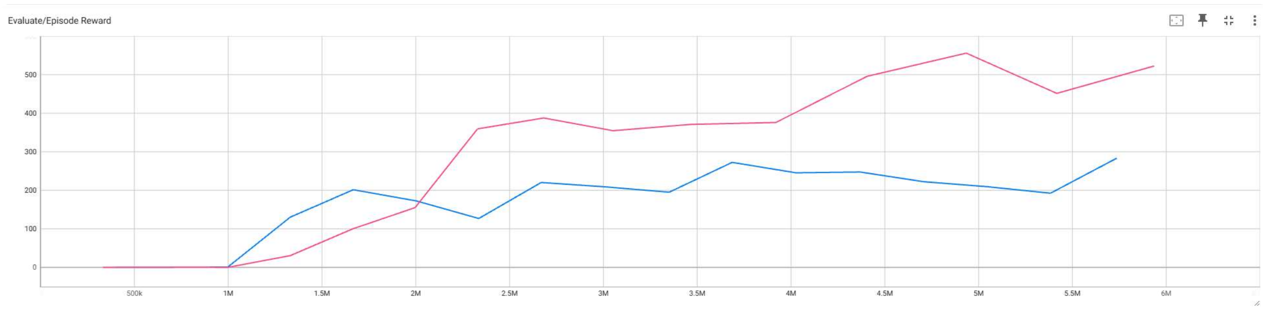


DQN V.S Dueling DQN training curve (smoothed)



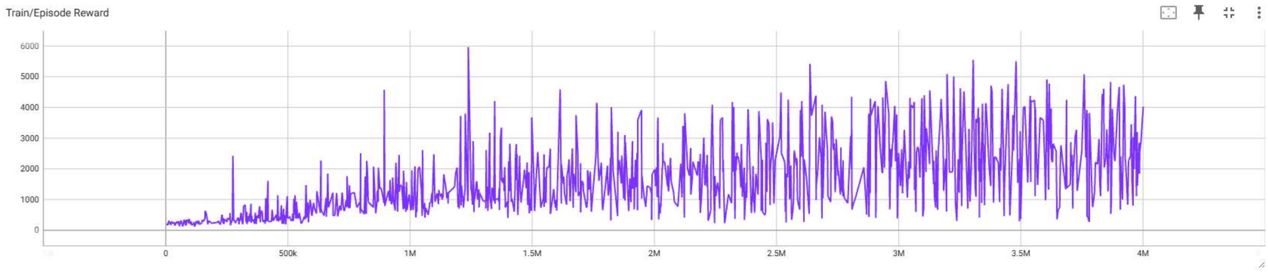
	Run	Smoothed	Value	Step	Relative
●	DQN	156.643	333	6,001,942	8.894 hr
●	Dueling_DQN	298.9722	195	5,998,874	9.216 hr

DQN V.S Dueling DQN testing curve

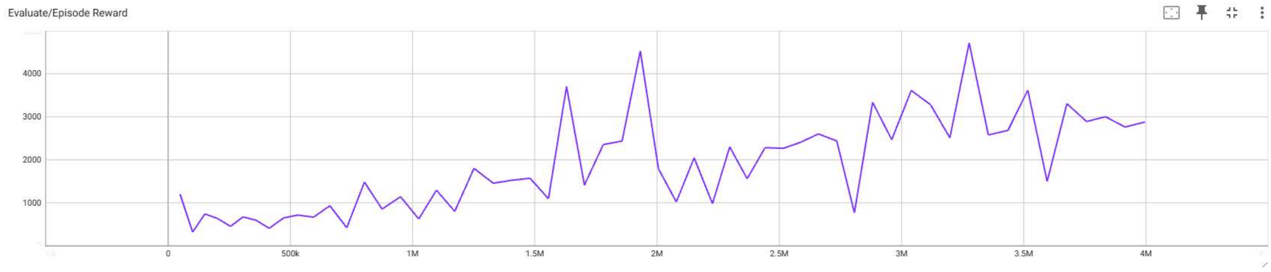


Screenshot of Tensorboard training curve and testing results on DDQN, and discuss the difference between DQN and DDQN

DDQN training curve



DDQN testing curve



DDQN testing result

```
episode 1 reward: 5360.0
episode 2 reward: 5360.0
episode 3 reward: 5360.0
episode 4 reward: 4040.0
episode 5 reward: 5360.0
average score: 5096.0
```

Difference Between DQN & DDQN

- DQN

  DQN 會使用 target net 從 next state 得到所有 action 中最好的分數

  $Y_t^Q = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a'|\theta)$

```
q_value = torch.gather(self.behavior_net(state), 1, action.long())
with torch.no_grad():
    q_next = self.target_net(next_state).max(1)[0].unsqueeze(1)

    # if episode terminates at next_state, then q_target = reward
    q_target = torch.where(done.bool(), reward, reward + self.gamma * q_next)
```

- DDQN

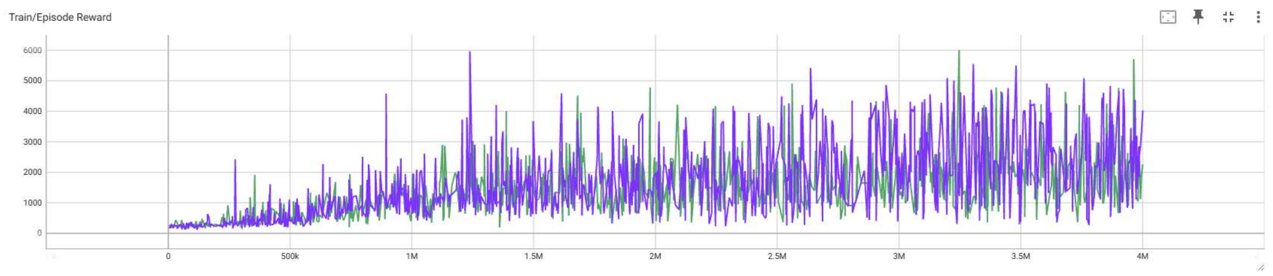
  而 DDQN 會先使用 behavior net 得到 best action 後，再使用 target net 對於 next state 的各個 action 的分數中，取出該 best action 的分數

```python
q_value = torch.gather(self.behavior_net(state), 1, action.long())
with torch.no_grad():
    next_action = self.behavior_net(next_state).argmax(dim=1)
    q_next = torch.gather(self.target_net(next_state), 1, next_action.long().unsqueeze(1))

    # if episode terminates at next_state, then q_target = reward
    q_target = torch.where(done.bool(), reward, reward + self.gamma * q_next)
```

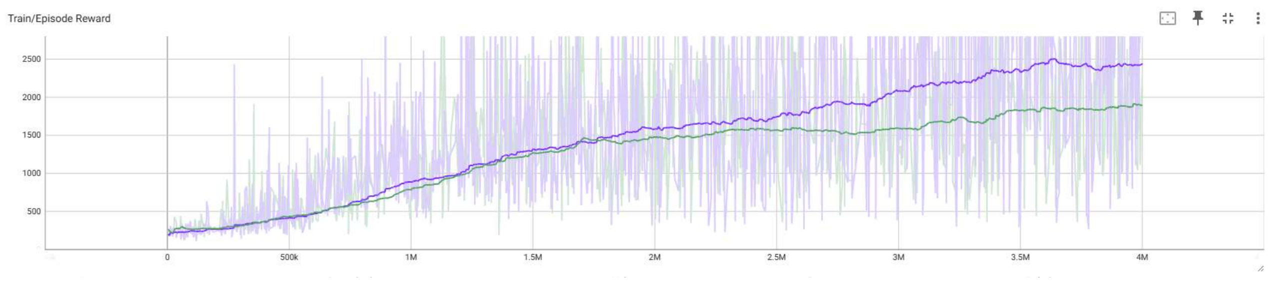
- 差異

  因為 target net 是比較久之前的 behavior net，target net 中做出最好的 action 會和現在較新的 behavior net 做出的 action 有所差距，使用 behavior net 中的 best action 會更接近當前 agent 做出的選擇，並且不會每次都選擇 target net 中 best action 的分數，可以解決 DQN 高估 Q 值的問題，使得 Q 值更接近真實值。

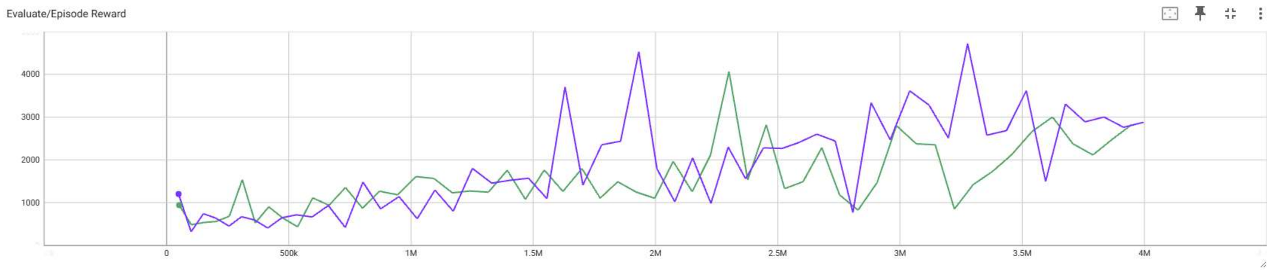
DQN V.S DDQN training curve



DQN V.S DDQN testing curve (smoothed)

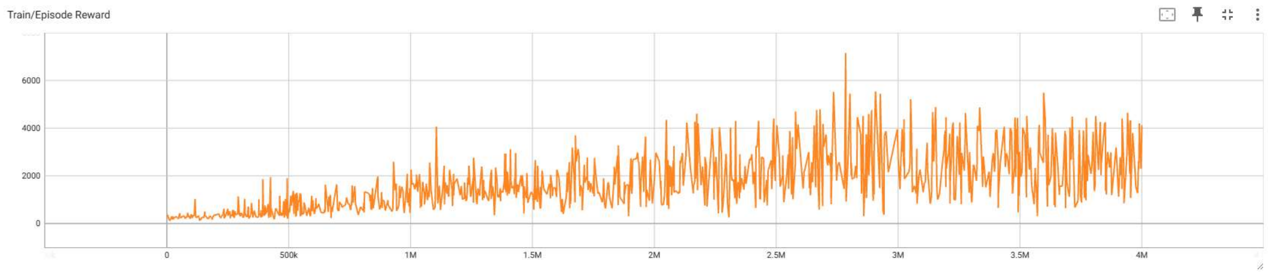


DQN V.S DDQN testing curve

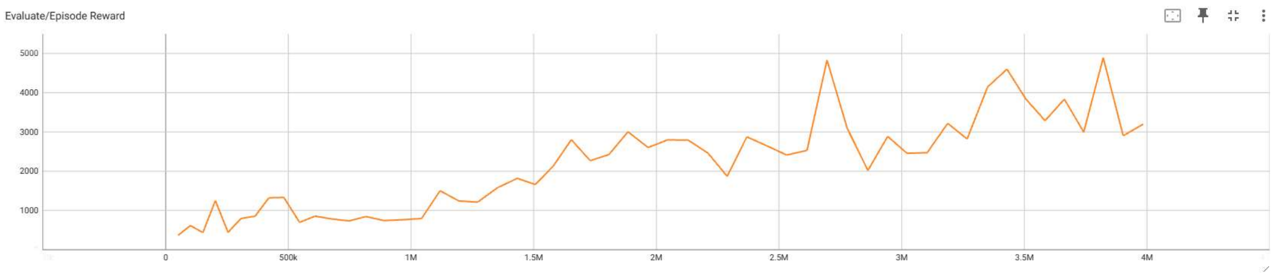


Screenshot of Tensorboard training curve and testing results on DDQN, and discuss the difference between DQN and Dueling DQN

Dueling DQN training curve



Dueling DQN testing curve



Dueling DQN testing result

```
episode 1 reward: 4240.0
episode 2 reward: 5330.0
episode 3 reward: 3980.0
episode 4 reward: 5560.0
episode 5 reward: 5430.0
average score: 4908.0
```

Difference Between DQN & Dueling DQN

- 相對 DQN 直接使用一個 network 來得出 Q 值

```python
class AtariNetDQN(nn.Module):
    def __init__(self, num_classes=4, init_weights=True):
        super(AtariNetDQN, self).__init__()
        self.cnn = nn.Sequential(nn.Conv2d(4, 32, kernel_size=8, stride=4),
                                        nn.ReLU(True),
                                        nn.Conv2d(32, 64, kernel_size=4, stride=2),
                                        nn.ReLU(True),
                                        nn.Conv2d(64, 64, kernel_size=3, stride=1),
                                        nn.ReLU(True)
                                        )
        self.classifier = nn.Sequential(nn.Linear(7*7*64, 512),
                                        nn.ReLU(True),
                                        nn.Linear(512, num_classes)
                                        )

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = x.float() / 255.
        x = self.cnn(x)
        x = torch.flatten(x, start_dim=1)
        x = self.classifier(x)
        return x
```

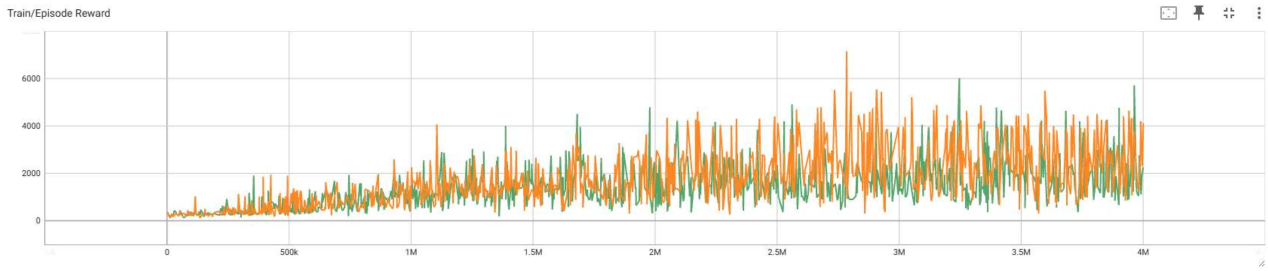
- Dueling DQN 把 Q 值分成 V 和 A 值，分別代標該 state 的 state_value 和該 state 下每個 action 的 advantage value，再將兩個值相加

```python
class AtariNetDuelingDQN(nn.Module):
    def __init__(self, num_classes=4, init_weights=True):
        super(AtariNetDuelingDQN, self).__init__()
        self.cnn = nn.Sequential(nn.Conv2d(4, 32, kernel_size=8, stride=4),
                                        nn.ReLU(True),
                                        nn.Conv2d(32, 64, kernel_size=4, stride=2),
                                        nn.ReLU(True),
                                        nn.Conv2d(64, 64, kernel_size=3, stride=1),
                                        nn.ReLU(True)
                                        )
        self.extractor = nn.Sequential(nn.Linear(7*7*64, 512),
                                        nn.ReLU(True)
                                        )
        self.value_network = nn.Sequential(nn.Linear(512, 1))
        self.advantage_network = nn.Sequential(nn.Linear(512, num_classes))

        if init_weights:
            self._initialize_weights()

    def forward(self, x):
        x = x.float() / 255.
        x = self.cnn(x)
        x = torch.flatten(x, start_dim=1)
        x = self.extractor(x)
        value = self.value_network(x)
        advantage = self.advantage_network(x)
        q_value = value + (advantage - advantage.mean(dim=1, keepdim=True))
        return q_value
```

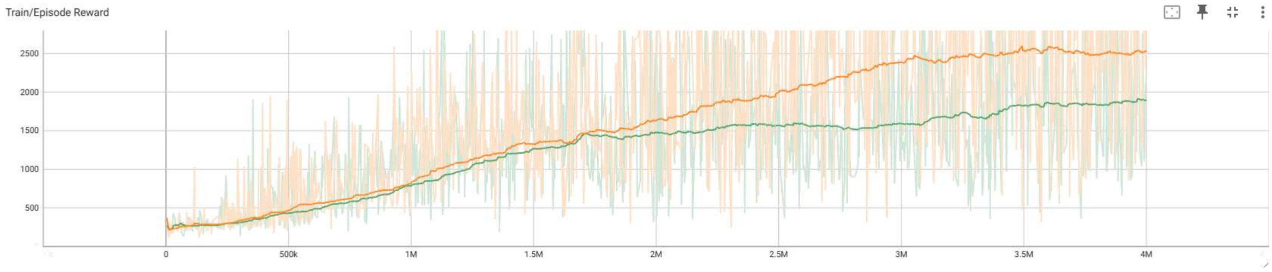
- 差異

  Dueling DQN 確保 Q 值基於該狀態下的價值，加上每個動作的相對優勢值，這樣定義可以讓模型對於環境的理解更靈活，以達到更有效的學習

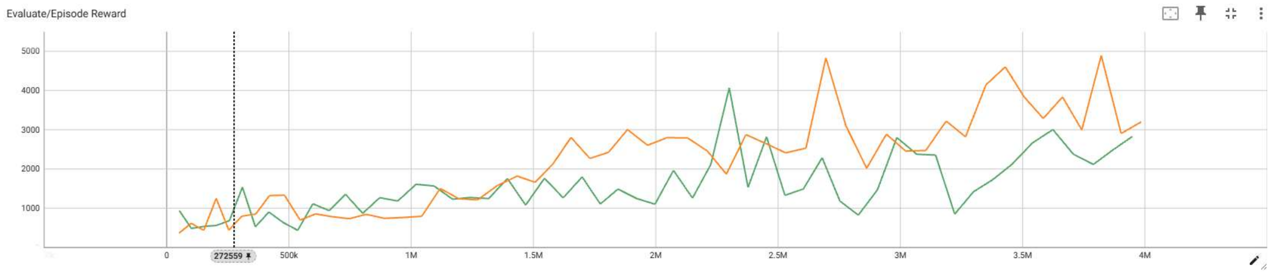
DQN V.S Dueling DQN training curve



DQN V.S Dueling DQN training curve (smoothed)

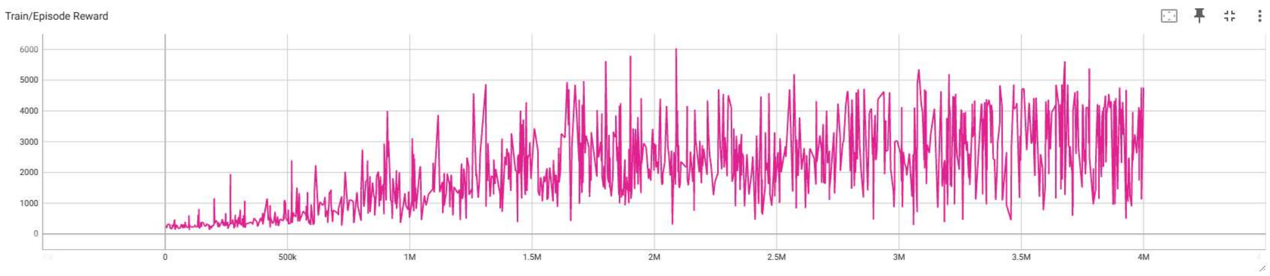


DQN V.S Dueling DQN testing curve

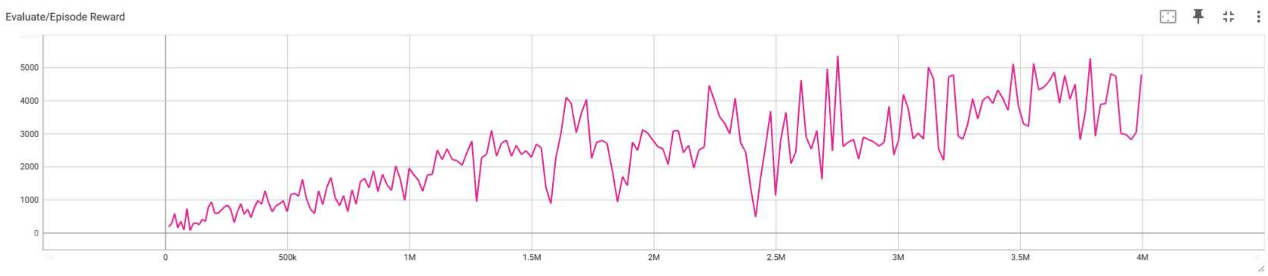


Screenshot of Tensorboard training curve and testing results on DQN with parallelized rollout, and discuss the difference between DQN and DQN with parallelized rollout

DQN with parallelized rollout training curve



DQN with parallelized rollout testing curve



DQN with parallelized rollout testing result

```
episode 1 reward: 4970.0
episode 2 reward: 5010.0
episode 3 reward: 4970.0
episode 4 reward: 5800.0
episode 5 reward: 4830.0
average score: 5116.0
```

Difference between DQN && DQN with parallelized rollout

- Parrallized 和 gym 互動的 API 沒差太多，差別在於它會回傳 np.array of enviroment，我們需要用 np.array of action 跟他互動

```
envs = gym.make_vec(config["env_id"], wrappers=[myWrapper], num_envs=4)
next_observation, reward, terminate, truncate, info = envs.step(action)
```

- decide actions 這邊我新增判斷 observations 維度若為 4，代表有多個 observations 被傳進來，update 那邊本來就可以一個一個 batch 處理，沒做更改

```python
def decide_agent_actions(self, observation, epsilon=0.0, action_space=None):
    ### TODO ###
    # get action from behavior net, with epsilon-greedy selection

    if random.random() < epsilon:
        action = action_space.sample()
    elif len(observation.shape) == 4:
        observation = np.array(observation)
        observation = torch.tensor(observation).float().to(self.device)
        output = self.behavior_net(observation)
        _, action = torch.max(output, 1)
        action = action.cpu().numpy()
    else:
        observation = np.array(observation)
        observation = torch.tensor(observation).float().to(self.device).unsqueeze(0)
        action = self.behavior_net(observation).argmax().item()

    return action
```

- 做比較多改動的地方是對於 episode 的計算，需要把原本的迴圈架構做一些更改，前半部就是把 reward 和 len 改成 array，分別記錄各個np.array of environment 的狀態，然後分別把這些 record 放到 replay buffer 裡面

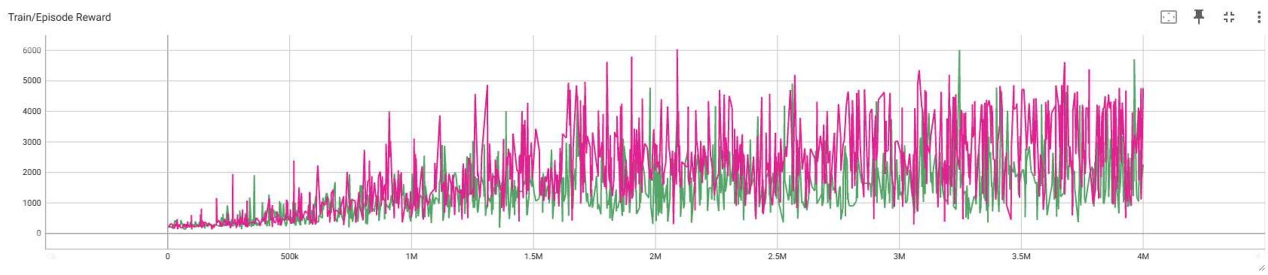
```python
def train(self):
    episode_idx = 0
    observation, info = self.envs.reset()
    episode_reward = np.zeros(self.envs.num_envs)
    episode_len = np.zeros(self.envs.num_envs)
    episode_idx += 1
    while self.total_time_step <= self.training_steps:
        if self.total_time_step < self.warmup_steps:
            action = self.decide_agent_actions(observation, 1.0, self.envs.action_space)
        else:
            action = self.decide_agent_actions(observation, self.epsilon, self.envs.action_space)
            self.epsilon_decay()

        next_observation, reward, terminate, truncate, info = self.envs.step(action)
        for obs, act, rew, next_obs, term in zip(observation, action, reward, next_observation, terminate):
            self.replay_buffer.append(obs, [act], [rew], next_obs, [term])

        if self.total_time_step >= self.warmup_steps:
            self.update()

        episode_reward += reward
        episode_len += 1
```

- 第二部分就是分別判斷每個 environment terminate 的時候要對 reward 和 episode_length 做更新，其他沒甚麼差別

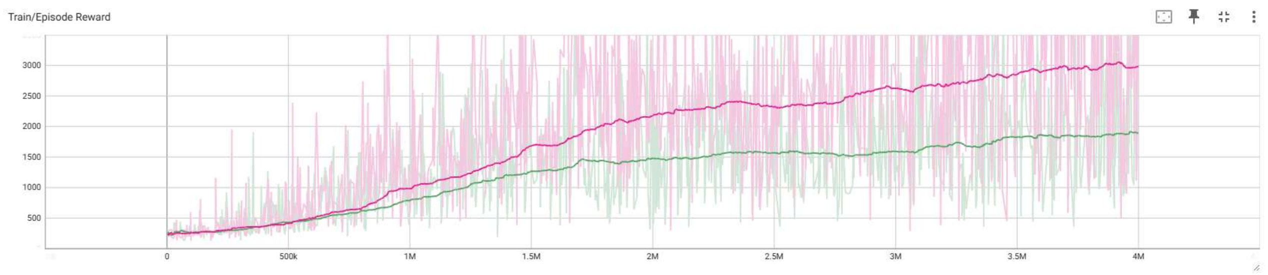
```python
for t in range(len(terminate)):
    if terminate[t] or truncate[t]:
        self.writer.add_scalar('Train/Episode Reward', episode_reward[t], self.total_time_step)
        self.writer.add_scalar('Train/Episode Len', episode_len[t], self.total_time_step)
        episode_reward[t] = 0
        episode_len[t] = 0
        episode_idx += 1

        if episode_idx % self.eval_interval == 0:
            # save model checkpoint
            avg_score, _ = self.evaluate()
            self.save(os.path.join(self.writer.log_dir, f"model_{self.total_time_step}_{int(avg_score)}.pth"))
            self.writer.add_scalar('Evaluate/Episode Reward', avg_score, self.total_time_step)

observation = next_observation
self.total_time_step += 1
```

- 差異
  平行化的更新次數雖然一樣是每個 timestamp 更新 batch 個資料，但因為同時有四個環境在跑，所以 replay buffer 中的資料會更多樣一點，並且因為資料變成四倍，所以塞滿 buffer 後，可以取到最舊的資料也會相對新很多。

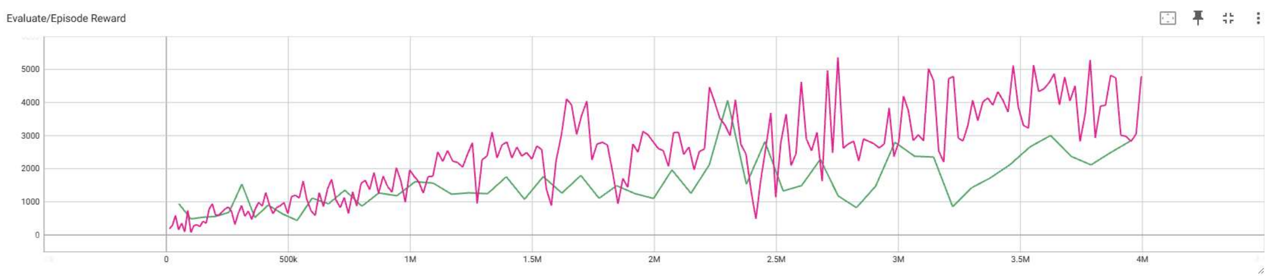
DQN V.S DQN with parallelized rollout training curve



DQN V.S DQN with parallelized rollout training curve (smoothed)



DQN V.S DQN with parallelized rollout testing curve



Compare all