RNE 2024 HW1 Report

許媄香 109000168

TODO 1

For policy network architecture, I made a rough guess by selecting 32 neurons in hidden layers. Then for the distribution, I saw the hint on the slide and use the implemented **DiagGaussian** class directly.

```
#TODO 1: policy network architecture
self.main = nn.Sequential(
    init_(nn.Linear(s_dim, 32)),
    nn.ReLU(),
    init_(nn.Linear(32, 32)),
    nn.ReLU(),
    init_(nn.Linear(32, 1))
)
self.dist = DiagGaussian(1, a_dim, std)
```

TODO 2

For value network, it is similar to policy network but without the probability distribution.

```
#TODO 2: value network architecture
self.main = nn.Sequential(
    init_(nn.Linear(s_dim, 32)),
    nn.ReLU(),
    init_(nn.Linear(32, 32)),
    nn.ReLU(),
    init_(nn.Linear(32, 1))
)
```

TODO 3

TODO 3 is just simply need to store the information in batch memory. For actions, log probabilities and values, we just need to retrieve them from the networks designed in **TODO 1** and **2**, the policy and value networks.

```
#TODO 3: Run a step to collect data
actions, a_logps = policy_net(torch.from_numpy(self.states).float().to(self.device))
actions, a_logps = actions.cpu().numpy(), a_logps.cpu().numpy()
self.mb_states[step, :]  = self.states
self.mb_dones[step, :]   = self.dones
self.mb_actions[step, :] = actions
self.mb_a_logps[step, :] = a_logps
self.mb_values[step, :]  = value_net(torch.from_numpy(self.states).float().to(self.device)).cpu().numpy()
self.states, rewards, self.dones, _ = self.env.step(actions)
self.mb_rewards[step, :] = rewards
```

TODO 4

The loss function follows the formula in the slide but with additional entropy to introduce randomness and enhance exploration within the model. To prevent division by zero, computation of ratio is taking the exponential of log probability.

$$\nabla_{\theta'}^{PPO2} J(\theta) \approx \sum_{s_t,a_t} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} G^{\theta'}(\tau), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) G^{\theta'}(\tau)\right)$$

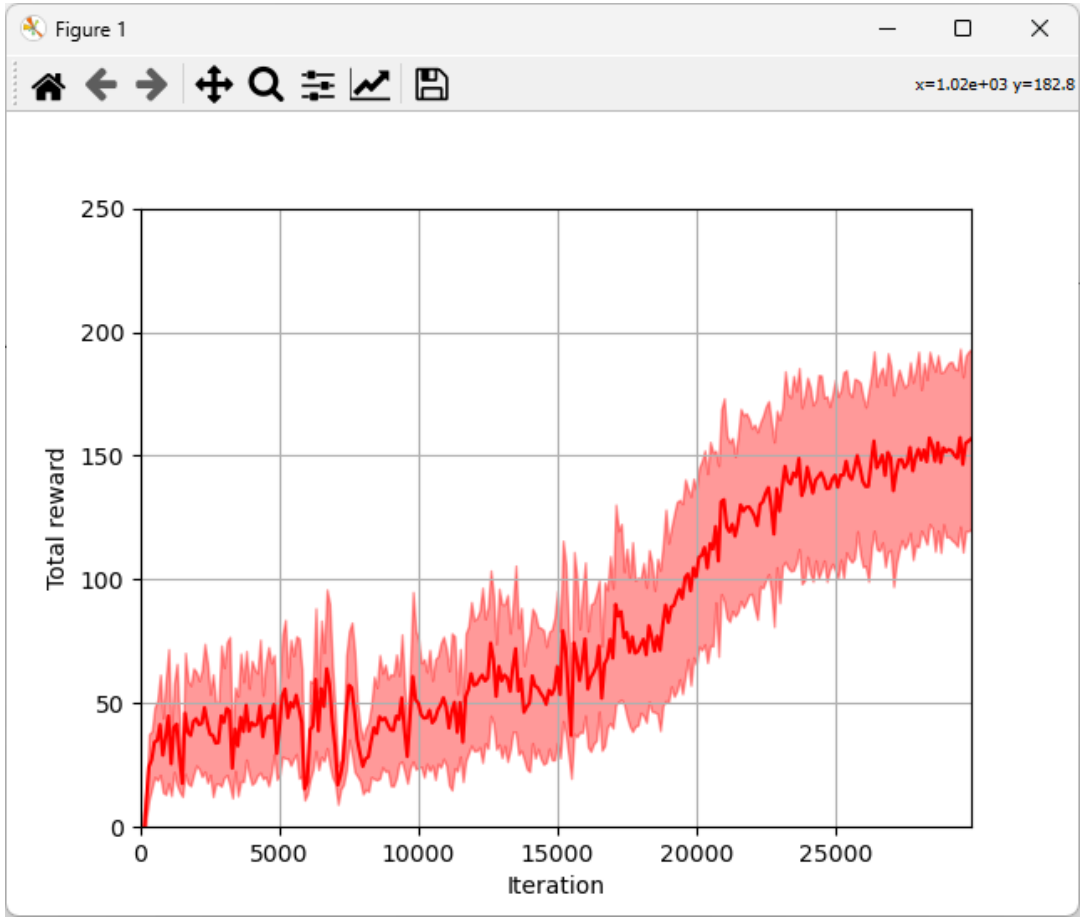
```
#TODO 4: Policy gradient loss for PPO
ratio = torch.exp(sample_a_logps - sample_old_a_logps)
clip_adv = torch.clamp(ratio, 1.0 - self.clip_val, 1.0 + self.clip_val) * sample_advs
pg_loss = -torch.min(ratio * sample_advs, clip_adv).mean() - 0.01 * sample_ents.mean()
```

TODO 5

I didn’t change any of the parameters because I obtained pretty okay result in the first trial using the given parameters.

```
#TODO 5: Adjust these parameters if needed
#Parameters that can be modified
#----------------------------
n_env          = 8
n_step         = 128
sample_mb_size = 64
sample_n_epoch = 4
a_std          = 0.5
lamb           = 0.95
gamma          = 0.99
clip_val       = 0.2
lr             = 1e-4
n_iter         = 30000
# device         = 'cpu'
device         = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

plot.py / eval.py




```
Total reward = 124.753029, length = 309
Total reward = 99.536675, length = 296
Total reward = 175.995713, length = 321
Total reward = 109.961607, length = 256
Total reward = 171.274486, length = 292
Total reward = 182.626463, length = 316
Total reward = 188.595095, length = 289
Total reward = 195.568619, length = 319
Evaluation Score: 170.1296
```

play.py