Robotic Navigation and Exploration

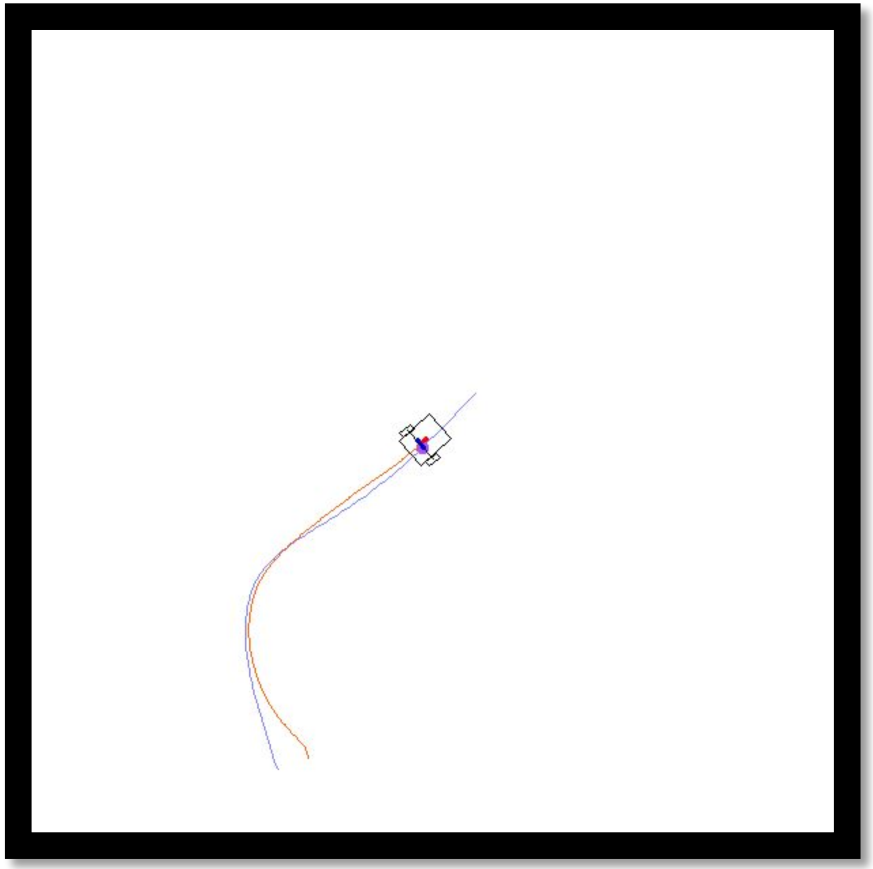
HW3: Deep Reinforcement Learning on Path Tracking

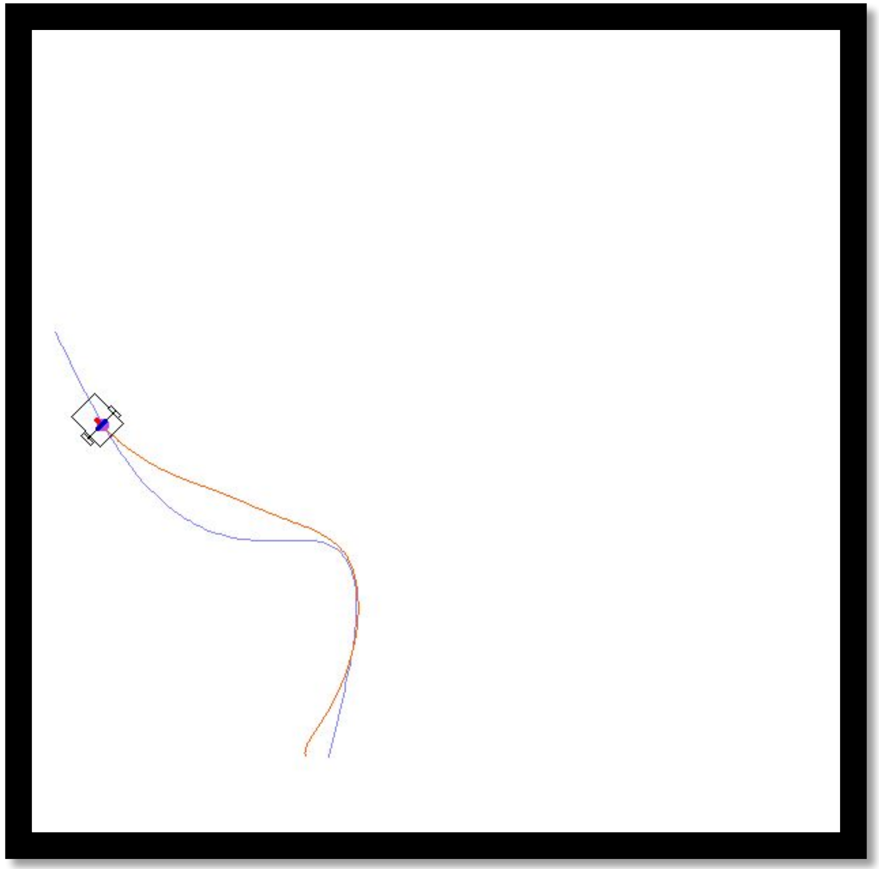
Min-Chun Hu anitahu@cs.nthu.edu.tw
CS, NTHU

Navigation Environment

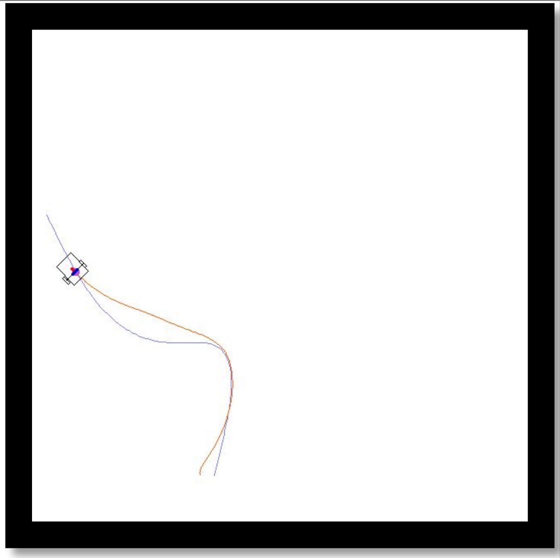
Navigation Environment

- Path following





Navigation Environment



- **State:** future positions + past positions + past yaws ($s \in \mathbb{R}^{14}$)
- **Action:** current angular velocity ($a \in \mathbb{R}$)
- **Reward:** $r = w_d r_d + w_y r_y + r_p$

Distance reward: $r_d = \exp(-0.1\|p - p'\|^2)$

Yaw reward: $r_y = \exp(-0.1\|\theta - \theta'\|^2)$

Progress reward: $r_p = \begin{cases} 0.1 & \text{, if progress is positive} \\ 0 & \text{, if progress is } 0 \\ -1 & \text{, if progress is negative} \end{cases}$

- **Termination:**

100% progress or reach 400 steps

Note:
The value range of action is normalized to $[-1, 1]$. If the input action value is out of range, the value will be clipped.

PPO Algorithm

PPO Algorithm

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

Run an episode to collect data

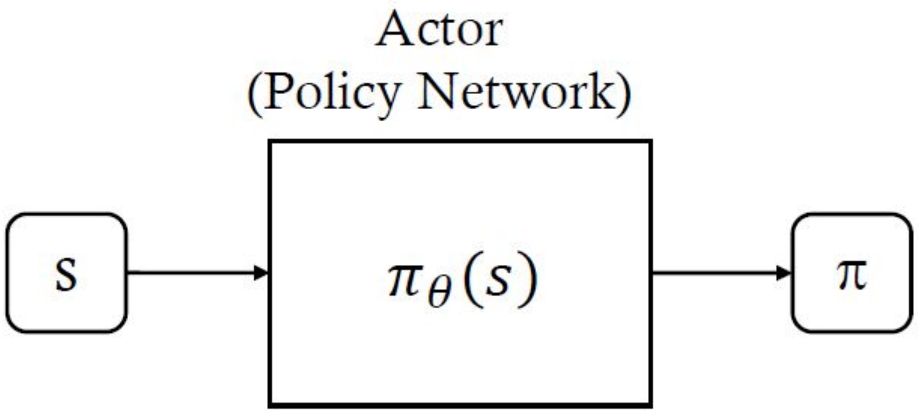
Train the model using the collected data

Reference: John Schulman, et al. “Proximal Policy Optimization Algorithms." , CoRR abs/1707.06347.arXiv:1707.06347

PPO Algorithm

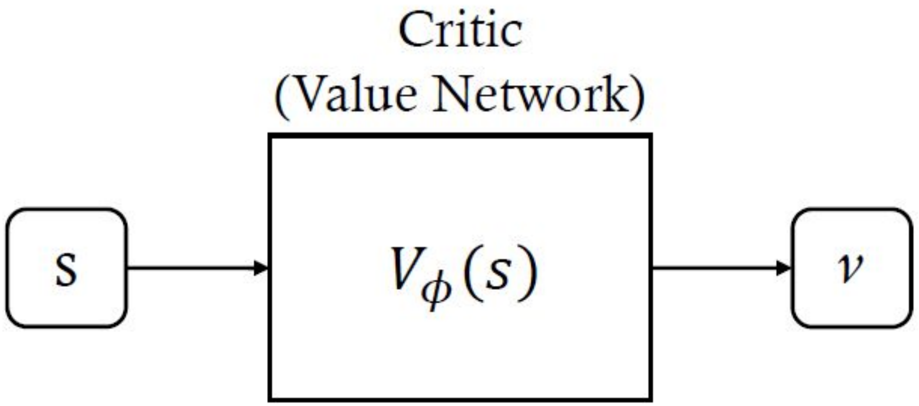
- Actor Network:

$$\pi_\theta(a|s)$$



- Critic Network:

$$V_\omega(s)$$



PPO Algorithm

- Value Loss:

$$L(\omega) = \frac{1}{2}\sum_n [G_t - V_\omega(s_t^{(n)})]^2$$

- Policy Gradient Loss:

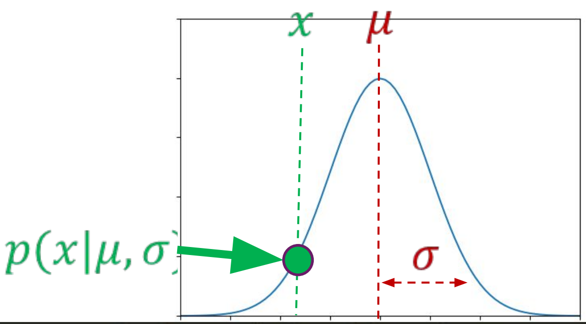
$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

$$L^{CLIP}(\theta) = E_{(s_t,a_t)\sim\pi_{\theta_{old}}}\left[min\{\underline{r_t(\theta)A^{\theta_{old}}(s_t,a_t)}, \quad \underline{clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A^{\theta_{old}}(s_t,a_t)}\}\right]$$

Policy Network & Value Network Construction

Diagonal Gaussian Distribution Module (model.py)

1. FixedNormal



$$N(x|\mu,\Sigma) = \frac{1}{(2\pi)^{(D/2)}} \frac{1}{|\Sigma|^{1/2}} exp\{\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\}$$

$$\ln p(X|\mu,\Sigma) = -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\Sigma| - \frac{1}{2}\sum_{n=1}^{N}(x_n-\mu)^T\Sigma^{-1}(x_n-\mu)$$

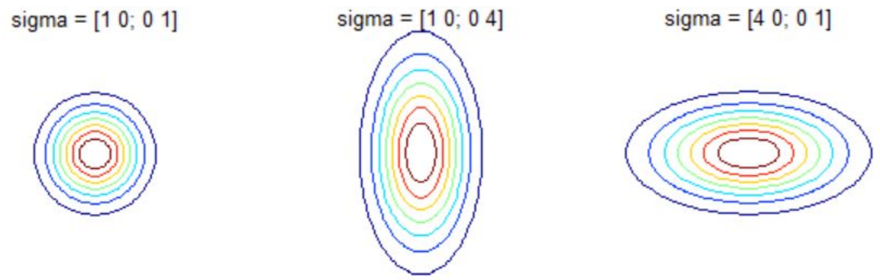
```python
#Normal distribution module with fixed mean and std.
class FixedNormal(torch.distributions.Normal):
    # Log-probability
    def log_probs(self, actions):
        return super().log_prob(actions).sum(-1)

    # Entropy
    def entropy(self):
        return super().entropy().sum(-1)

    # Mode
    def mode(self):
        return self.mean

```

Diagonal Gaussian Distribution Module (model.py)

2. DiagGaussian

$$mean = \begin{bmatrix} m_1 \\ m_2 \\ \vdots \end{bmatrix}, \qquad std = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \end{bmatrix} \Rightarrow \Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k^2 \end{bmatrix}$$

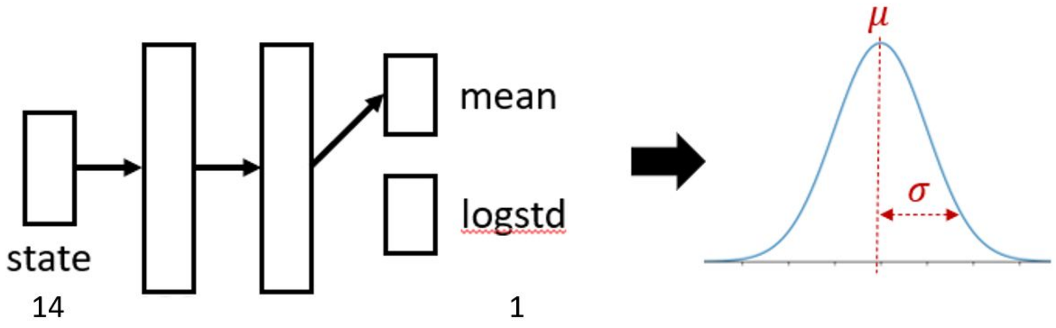
```python
#Diagonal Gaussian distribution
class DiagGaussian(nn.Module):
    # Constructor
    def __init__(self, inp_dim, out_dim, std=0.5):
        super(DiagGaussian, self).__init__()

        init_ = lambda m: init(
            m,
            nn.init.orthogonal_,
            lambda x: nn.init.constant_(x, 0)
        )
        self.fc_mean = init_(nn.Linear(inp_dim, out_dim))
        self.std = torch.full((out_dim,), std)

    # Forward
    def forward(self, x):
        mean = self.fc_mean(x)
        return FixedNormal(mean, self.std.to(x.device))

```



Policy Network Module

- **PolicyNet** class has the following functions:

  - ```
    #Constructor
    __init__(self, s_dim, a_dim, std)
    ```
  - ```
    #Forward pass of nn.Module
    forward(self, state, deterministic)
    ```
  - ```
    #Forward pass for outputting action only
    action_step(self, state, deterministic)
    ```
  - ```
    #Evaluate log-prob. & entropy
    evaluate(self, state, action)
    ```

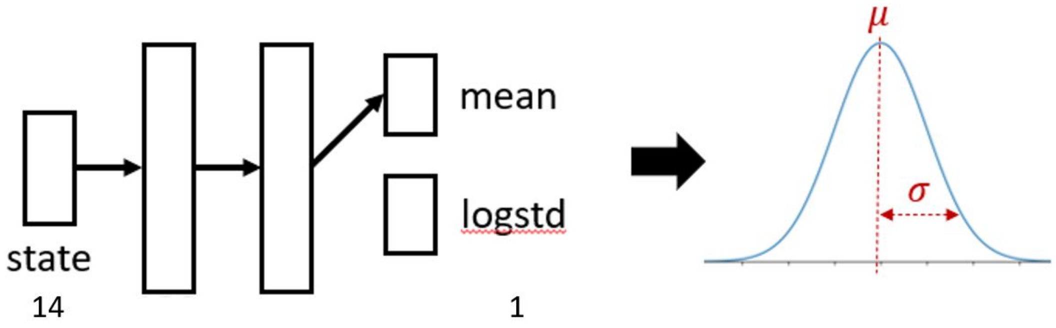


Policy Network Module (model.py)

- constructor (TODO 1)

(Hint: Use **nn.Sequntial** & **DiagGaussian**)

```python
#Policy network
class PolicyNet(nn.Module):
    # Constructor
    def __init__(self, s_dim, a_dim, std=0.5):
        super(PolicyNet, self).__init__()

        init_ = lambda m: init(
            m,
            nn.init.orthogonal_,
            lambda x: nn.init.constant_(x, 0),
            nn.init.calculate_gain('relu')
        )
        #TODO 1: policy network architecture
        '''
        self.main = ...
        self.dist = ...
        '''
```

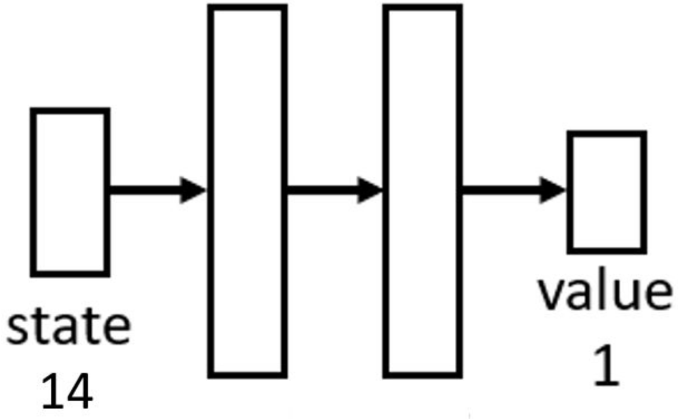


Value Network Module

- **ValueNet** class has the following functions:

    #Constructor
  - `__init__(self, s_dim)`

    #Forward pass of nn.Module
  - `forward(self, state)`

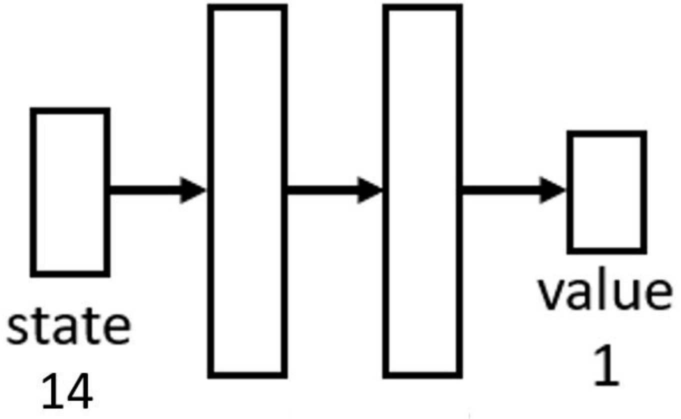


Value Network Module (model.py)

- constructor (TODO 2)

(Hint: Use **nn.Sequntial**)

```python
#Value network
class ValueNet(nn.Module):
    # Constructor
    def __init__(self, s_dim):
        super(ValueNet, self).__init__()

        init_ = lambda m: init(
            m,
            nn.init.orthogonal_,
            lambda x: nn.init.constant_(x, 0),
            nn.init.calculate_gain('relu')
        )
        #TODO 2: value network architecture
        '''
        self.main = ...
        '''
```



Environment Runner Construction

EnvRunner Class

- **EnvRunner** class has the following functions:

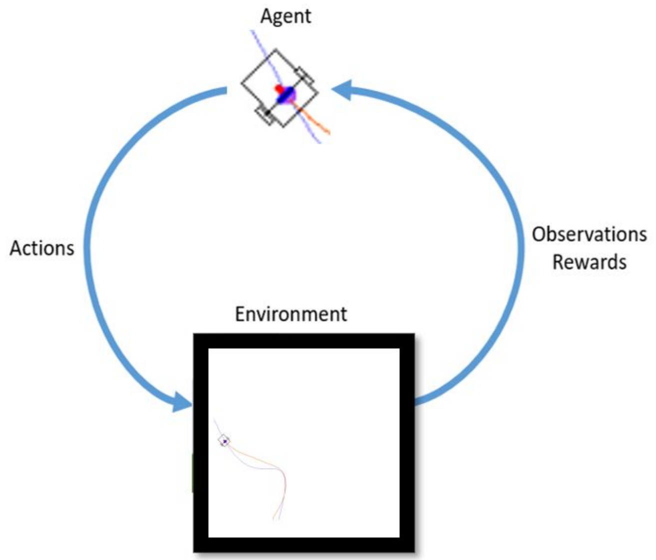
  - ```
    #Constructor
    __init__(self, env, s_dim, a_dim, n_step, gamma, lamb, device)
    ```
  - ```
    #Run n steps to get a batch
    run(self, policy_net, value_net)
    ```
  - ```
    #Record return & length
    record(self)
    ```
  - ```
    #Get current performance
    get_performance(self)
    ```

EnvRunner Class (env_runner.py)

- run (TODO 3)

(Hint: Use **policy_net** & **value_net**)

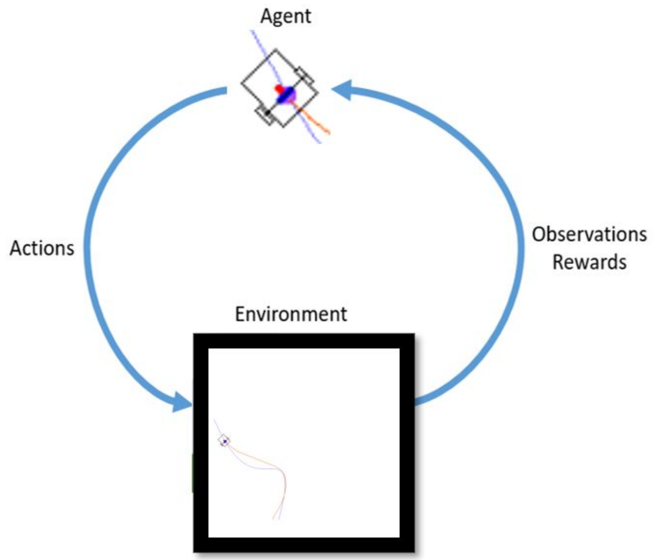
```
    # Run n steps to get a batch
    def run(self, policy_net, value_net):
        #1. Run n steps
        #-------------------------------------
        for step in range(self.n_step):
            #self.states : (n_env, s_dim)
            #actions     : (n_env, a_dim)
            #self.dones  : (n_env)
            #a_logps     : (n_env)
            #values      : (n_env)
            #rewards     : (n_env)
            #TODO 3: Run a step to collect data
            '''
            self.mb_states[step, :]  = ...
            self.mb_dones[step, :]   = ...
            self.mb_actions[step, :] = ...
            self.mb_a_logps[step, :] = ...
            self.mb_values[step, :]  = ...
            self.states, rewards, self.dones, info = self.env.step(actions)
            self.mb_rewards[step, :] = ...
            '''

        last_values = value_net(torch.from_numpy(self.states).float().to(self.device)).cpu().numpy()
        self.record()
```



- **mb_states:** (n_step, n_env, s_dim)
- **mb_actions:** (n_step, n_env, a_dim)
- **mb_dones:** (n_step, n_env)
- **mb_a_logps:** (n_step, n_env)
- **mb_values:** (n_step, n_env)
- **mb_rewards:** (n_step, n_env)

EnvRunner Class (env_runner.py)

Output: $\{s_t, a_t, \log\pi(a_t|s_t), V(s_t), G_t, r(s_t, a_t)\}$




```python
        #2. Compute returns
        #------------------------------------
        mb_returns = compute_gae(self.mb_rewards, self.mb_values, self.mb_dones, last_values, self.don

        #mb_states : (n_step*n_env, s_dim)
        #mb_actions: (n_step*n_env) or (n_env*n_step, a_dim)
        #mb_a_logps: (n_step*n_env)
        #mb_values : (n_step*n_env)
        #mb_returns: (n_step*n_env)
        return self.mb_states.reshape(self.n_step*self.n_env, self.s_dim), \
                self.mb_actions.reshape(self.n_step*self.n_env, self.a_dim), \
                self.mb_a_logps.flatten(), \
                self.mb_values.flatten(), \
                mb_returns.flatten()

```

PPO Agent

PPO Class

- **PPO** class has the following functions:

  - #Constructor
    `__init__(self, policy_net, value_net, lr, max_grad_norm, clip_val, sample_n_epoch, sample_mb_size, mb_size, device)`
  - #Train PPO
    `train(self, mb_states, mb_actions, mb_old_values, mb_advs, mb_returns, mb_old_a_logps)`
  - #Learning rate decay
    `lr_decay(self, it, n_it)`

PPO Class (agent.py)

- train (TODO 4)

```python
    #Train PPO
    def train(self, mb_states, mb_actions, mb_old_values, mb_advs, mb_returns, mb_old_a_logps):
        mb_states       = torch.from_numpy(mb_states).to(self.device)
        mb_actions      = torch.from_numpy(mb_actions).to(self.device)
        mb_old_values   = torch.from_numpy(mb_old_values).to(self.device)
        mb_advs         = torch.from_numpy(mb_advs).to(self.device)
        mb_returns      = torch.from_numpy(mb_returns).to(self.device)
        mb_old_a_logps = torch.from_numpy(mb_old_a_logps).to(self.device)

        for i in range(self.sample_n_epoch):
            np.random.shuffle(self.rand_idx)

            for j in range(self.sample_n_mb):
                sample_idx          = self.rand_idx[j*self.sample_mb_size : (j+1)*self.sample_mb_size]
                sample_states       = mb_states[sample_idx]
                sample_actions      = mb_actions[sample_idx]
                sample_old_values   = mb_old_values[sample_idx]
                sample_advs         = mb_advs[sample_idx]
                sample_returns      = mb_returns[sample_idx]
                sample_old_a_logps = mb_old_a_logps[sample_idx]

                sample_a_logps, sample_ents = self.policy_net.evaluate(sample_states, sample_actions)
                sample_values = self.value_net(sample_states)

```

PPO Class (agent.py)

(Hint: Use **sample_a_logps** & **sample_old_a_logps** to compute the probability ratio)

```
            #PPO loss
            v_pred_clip = sample_old_values + torch.clamp(sample_values - sample_old_values, -self.clip_va
            v_loss1     = (sample_returns - sample_values).pow(2)
            v_loss2     = (sample_returns - v_pred_clip).pow(2)
            v_loss      = torch.max(v_loss1, v_loss2).mean()

            #TODO 4: Policy gradient loss for PPO
            '''
            pg_loss  = ...
            '''

            #Train actor
            self.opt_actor.zero_grad()
            pg_loss.backward()
            nn.utils.clip_grad_norm_(self.policy_net.parameters(), self.max_grad_norm)
            self.opt_actor.step()

            #Train critic
            self.opt_critic.zero_grad()
            v_loss.backward()
            nn.utils.clip_grad_norm_(self.value_net.parameters(), self.max_grad_norm)
            self.opt_critic.step()

        return pg_loss.item(), v_loss.item()
```

Parameters

Parameters (train.py)

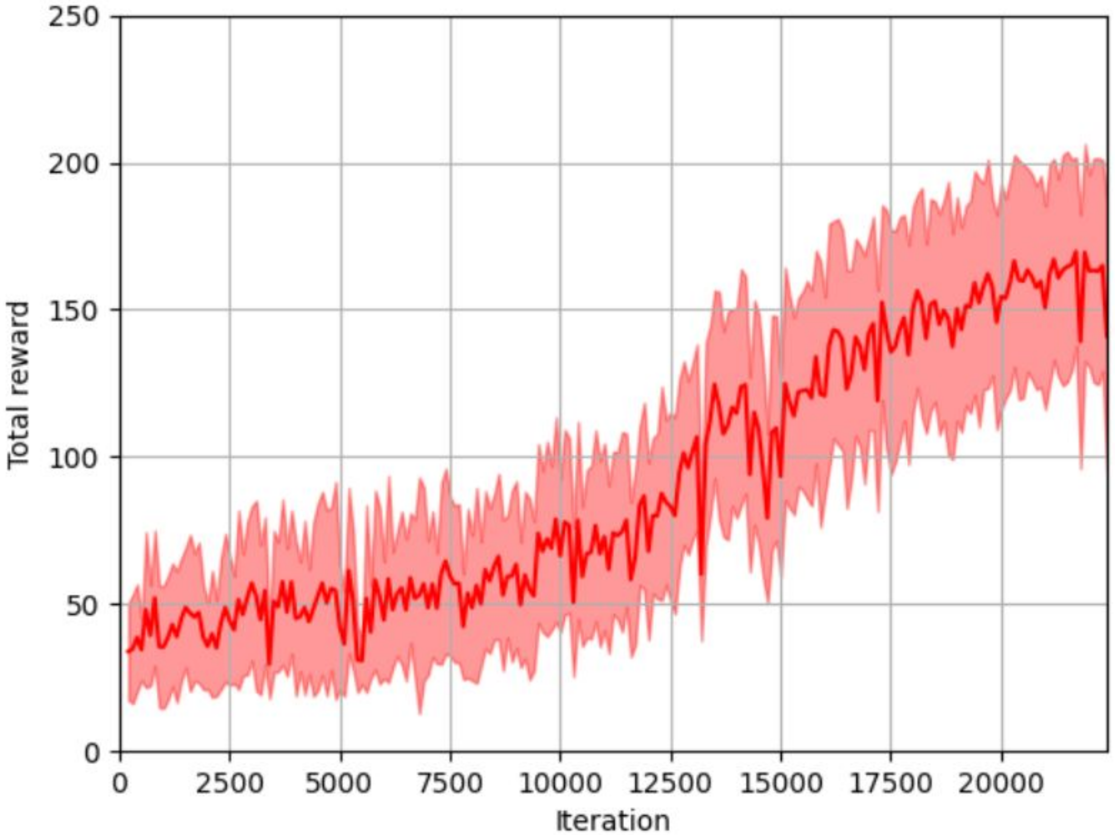
```
def main():
    #TODO 5: Adjust these parameters if needed
    #Parameters that can be modified
    #----------------------------
    n_env          = 8
    n_step         = 128
    sample_mb_size = 64
    sample_n_epoch = 4
    a_std          = 0.5
    lamb           = 0.95
    gamma          = 0.99
    clip_val       = 0.2
    lr             = 1e-4
    n_iter         = 30000

```

Note:
Too many environments may cause "OSError: The paging file is too small for this operation to complete". If so, please set **n_env** smaller.

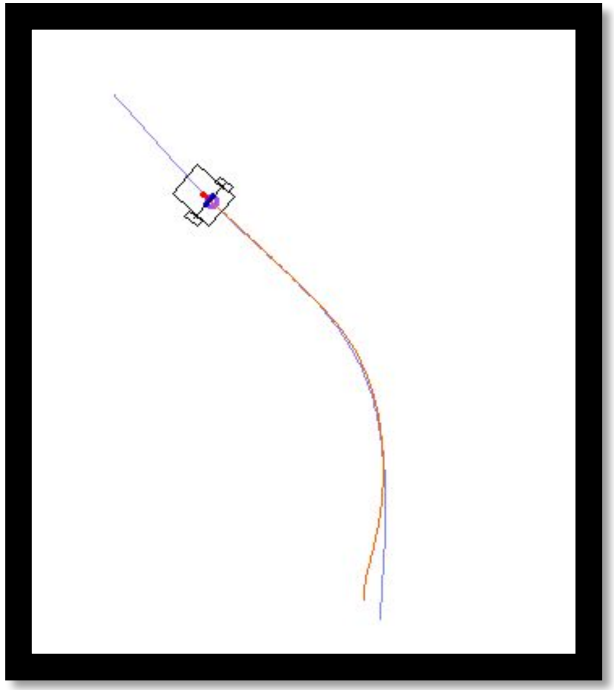
- `n_env`: number of environments (actors)
- `n_step`: number of step for runner
- `sample_mb_size`: sample mini-batch size
- `sample_n_epoch`: number of epoch in PPO
- `a_std`: std. dev. of action distribution
- `lamb`: gae factor λ
- `gamma`: discount factor γ
- `clip_val`: clip value ϵ
- `lr`: learning rate
- `n_iter`: number of iteration

Experimental Results

plot.py



play.py



eval.py

```
Total reward = 258.102730, length = 320
Total reward = 206.884708, length = 325
Total reward = 151.192916, length = 307
Total reward = 169.869626, length = 282
Total reward = 150.867508, length = 306
Total reward = 174.895304, length = 298
Total reward = 170.725990, length = 295
Evaluation Score: 159.6902
```

Requirements

- Python3.6+
- Numpy=1.20+ (not supported 2.0+)
- Matplotlib
- Opencv
- PyTorch
- Cloudpickle

Execution

- Training: `python train.py`
- Playing: `python play.py`
- Plotting: `python plot.py`
- Evaluating: `python eval.py`

Score & Requirement

- You should complete the code in "model.py", "env_runner.py", and "agent.py" and train the model. (TODO 1 ~ 4)
  - Score:
  - TODO 1(15%)
  - TODO 2(15%)
  - TODO 3(15%)
  - TODO 4(15%)
- After training, evaluate your model by executing "plot.py" and "eval.py".
  - Score: plot(10%), evaluation(30%)
  - Evaluation: 30 * ES / 120, where ES is your evaluation score (you will get 30 if ES > 120)
- Submit the zip of the project folder. It should include:
  - Code (*.py)
  - Training results ("save/")
  - Result folder should include:
    - return record ("return.txt")
    - Weightings ("model.pt")
- **Deadline: 2025/04/06 (11:59 pm)**