Meeting Discussion 23rd May 2021

Lunar Lander Pseudo Code

1. Initialize Policy parameters θ and State Value function parameters W
2. For each episode:
    1. Initialize State S_0 (first state of the episode)
    2. While S is not terminal:
        1. Sample action a_t, based on actor's policy μ_θ

           $$a \sim \pi(S, ., \theta)$$

        2. Receive reward R_{t+1} and update to next state (S_t to S_{t+1}).
        3. Save the action and the Value of the state into a list.

           $$S_a.insert(ln\mu_\theta(s_t, a_t), V_w(s_t))$$

           where,

           $ln\mu_\theta(s_t, a_t)$ = Log probability of selecting an action at time t

           $V_w(s_t)$ = Value of the state at time t

           S_a = List of saved actions and Value of the state

        4. Add the rewards to the episode's total rewards

           $$E_r = E_r + R_{t+1}$$

           where,

           E_r = Episode Reward,

    3. Update Average Reward:

       $$avg_r = (E_r + avg_r)/2$$

       where,

       avg_r = Average Reward,

4. Update the Loss functions and the parameters of critic and actor:

    1. Compute all n-step returns of the episode

$$G_t = \sum_{k=0}^{k=n-t} \gamma^k R_{k+t+1}$$

    Where,

    G_t = t step total return

    R = Reward

    2. Compute the total Policy Loss, i.e.

$$\delta_{policy\ loss} = \sum_{t=0}^{t=n} (-\ln\mu_\theta(s_t, a_t)(G_t - V_w(s_t)))$$

    3. Compute the total Value loss, i.e.

$$\delta_{value\ loss} = \sum_{t=0}^{t=n} smooth_{L_1}(V_w(s_t), G_t))$$

    4. Compute total loss, i.e.

$$total_{loss} = \delta_{policy\ loss} + \delta_{value\ loss}$$

    5. Update the parameters based on $total_{loss}$

    6. Hence updating the weights θ and w of the networks.

```
# saved_actions = list of (log_prob of actions selectected, and Value of state)
# returns = list of t-step return

for (log_prob, value), R in zip(saved_actions, returns):
        # Advantage function calculated
        advantage = R - value.item()
        # Actor's loss appended to sum later
        ActorCritic_losses.append((-log_prob) * advantage)
        # Critic's loss appended to sum later
        value_losses.append(F.smooth_l1_loss(value, torch.tensor([R])))

    # Clears the previous episodes gradients
    optimizer.zero_grad()
    # Total loss calculated which is distributed to the respective parameters
    # by pytorch itself.
    loss = torch.stack(ActorCritic_losses).sum() + torch.stack(value_losses).sum()
    # Perform backward propagation
    loss.backward()
    # Perform gradient descent (both parameter's update)
    optimizer.step()
```