111550177_HW3

Problem 1 (Surrogate Function in TRPO) (15 points)

In this problem, we will prove some key properties of the surrogate function used in TRPO. Recall from the slides of Lecture 16: Assume that both the state space and the action space are finite. The surrogate function $L_{\pi_{\theta_1}}(\pi_\theta)$ is defined as

$$L_{\pi_{\theta_1}}(\pi_\theta) := \eta(\pi_{\theta_1}) + \sum_{s\in\mathcal{S}} d_\mu^{\pi_{\theta_1}}(s) \sum_{a\in\mathcal{A}} \pi_\theta(a|s) A^{\pi_{\theta_1}}(s,a). \tag{1}$$

Show that $L_{\pi_{\theta_1}}(\pi_\theta)$ satisfies the two properties: (i) $L_{\pi_{\theta_1}}(\pi_{\theta_1}) = \eta(\pi_{\theta_1})$ and (ii) $\nabla_\theta L_{\pi_{\theta_1}}(\pi_\theta)|_{\theta=\theta_1} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_1}$.

(i) $L_{\pi_{\theta_1}}(\pi_{\theta_1}) = \eta(\pi_{\theta_1}) + \sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \pi_{\theta_1}(a|s) A^{\pi_{\theta_1}}(s,a)$

$= \eta(\pi_{\theta_1}) + \sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \pi_{\theta_1}(a|s) \left[Q^{\pi_{\theta_1}}(s,a) - V^{\pi_{\theta_1}}(s) \right]$

$= \eta(\pi_{\theta_1}) + \sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \pi_{\theta_1}(a|s) Q^{\pi_{\theta_1}}(s,a)$

$- \sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \pi_{\theta_1}(a|s) V^{\pi_{\theta_1}}(s)$

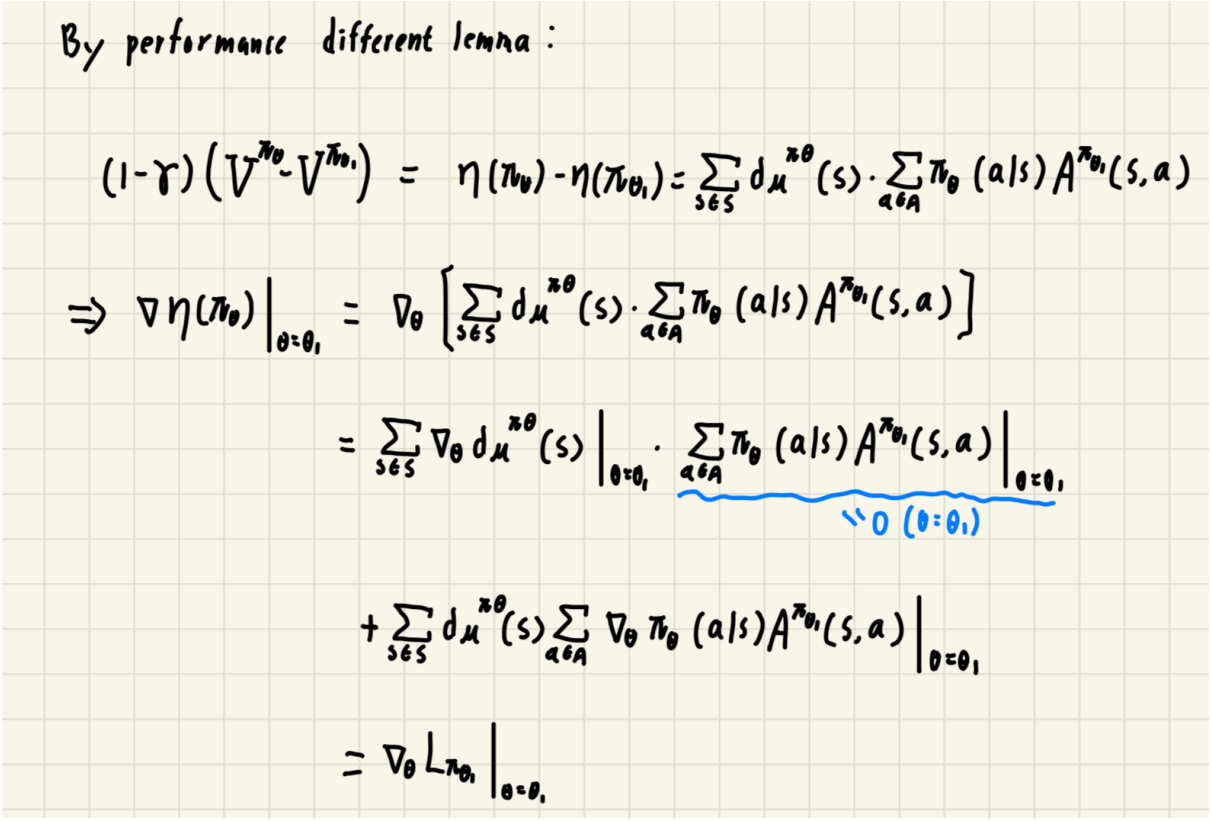
$= \eta(\pi_{\theta_1}) + V^{\pi_{\theta_1}}(s) - V^{\pi_{\theta_1}}(s)$

$= \eta(\pi_{\theta_1})$

(ii) $\nabla_\theta L_{\pi_{\theta_1}}|_{\theta=\theta_1} = \nabla_\theta \left[\eta(\pi_{\theta_1}) + \sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \pi_\theta(a|s) A^{\pi_{\theta_1}}(s,a) \right] \Big|_{\theta=\theta_1}$

$= \nabla_\theta \eta(\pi_{\theta_1})|_{\theta=\theta_1} + \nabla_\theta \left[\sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \pi_\theta(a|s) A^{\pi_{\theta_1}}(s,a) \right] \Big|_{\theta=\theta_1}$

$= \sum_{s\in S} d_\mu^{\pi_{\theta_1}}(s) \cdot \sum_{a\in A} \nabla_\theta \pi_\theta(a|s) A^{\pi_{\theta_1}}(s,a) \Big|_{\theta=\theta_1}$



Problem 2 (Solving TRPO Under Approximation Using Duality) (15+15=30 points)

Recall from the slides of Lecture 16: We would like to solve the following optimization problem (denoted by (OPT)):

$$\text{Minimize}_{\theta\in\mathbb{R}^d} \quad -(\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})^T(\theta-\theta_k) \tag{2}$$

$$\text{subject to} \quad \frac{1}{2}(\theta-\theta_k)^T H_{\theta_k}(\theta-\theta_k)-\delta\le 0. \tag{3}$$

We use θ^* to denote an optimizer of the above *primal* optimization problem (2)-(3). Note that in the above we write "Minimize" instead of "Maximize" simply to follow the conventions of the literature of optimization theory. Here we focus on the case where H is a *positive definite* matrix to avoid the technicalities (while H is only *non-negative definite* in general).

Based on the optimization theory, (OPT) is a convex optimization problem as both the objective and the constraints are convex functions. In this case, one standard way is to convert the constrained (OPT) into an unconstrained *dual* problem by defining the *Lagrangian* $\mathcal{L}(\theta,\lambda)$ and the *dual function* $D(\lambda)$ as:

$$\mathcal{L}(\theta, \lambda) := -(\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})^T(\theta - \theta_k) + \lambda\Big(\frac{1}{2}(\theta - \theta_k)^T H_{\theta_k}(\theta - \theta_k) - \delta\Big) \tag{4}$$

$$D(\lambda) := \min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta, \lambda), \tag{5}$$

where λ is called the *Lagrange multiplier*. Moreover, we call the following the *dual problem* of (OPT):

$$\max_{\lambda \geq 0} D(\lambda). \tag{6}$$

For ease of notation, we define $\lambda^* := \arg\max_{\lambda \geq 0} D(\lambda)$ as the optimizer of the dual problem. By the standard theory of convex optimization, if there exists one strictly feasible point in (OPT), then the optimal value of the dual problem is equal to the original problem (usually called "strong duality"). Moreover, if strong duality holds and a dual optimal solution λ^* exists, then any optimizer of the primal problem is also a minimizer of $\mathcal{L}(\theta, \lambda^*)$, i.e., $\theta^* = \arg\min_\theta \mathcal{L}(\theta, \lambda^*)$. For more details on duality, please refer to Chapter 5 of https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf.

(a) In this problem, please show that the dual function $D(\lambda)$ of (OPT) can be written as:

$$D(\lambda) = \frac{-1}{2\lambda}\big((\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})^T H_{\theta_k}^{-1}(\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k})\big) - \lambda\delta. \tag{7}$$

Accordingly, please find out λ^* based on (7).

(b) By the λ^* found in (a) and the property $\theta^* = \arg\min_\theta \mathcal{L}(\theta, \lambda^*)$, show that $\theta^* = \theta_k + \alpha H_{\theta_k}^{-1}\nabla_\theta L_{\theta_k}(\theta)|_{\theta=\theta_k}$. What is the step size α?

(a) Calculate $\frac{\partial \mathcal{L}(\theta,\lambda)}{\partial \theta} = 0$

$$\Rightarrow \frac{\partial \mathcal{L}(\theta,\lambda)}{\partial \theta} = -\left(\nabla_\theta L_{\theta_k}(\theta)\Big|_{\theta=\theta_k}\right) + \lambda(\theta - \theta_k)H_{\theta_k}$$

$$\Rightarrow \theta = \theta_k - \frac{\nabla_{\theta_k}(\theta)\,|_{\theta=\theta_k}}{\lambda}$$

Dfn $D(\lambda) = -\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)(\theta-\theta_k)$

$$+\lambda\left(\frac{1}{2}(\theta-\theta_k)^T H_{\theta_k}(\theta-\theta_k)-\delta\right)$$

$$= -\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)^T\cdot\left(\frac{\nabla_\theta L_{\theta_k}(\theta)}{\lambda}\Big|_{\theta=\theta_k}\cdot H_{\theta_k}^{-1}\right)+$$

$$\frac{\lambda}{2}\left(\frac{\nabla_\theta L_{\theta_k}(\theta)}{\lambda}\Big|_{\theta=\theta_k}\cdot H_{\theta_k}^{-1}\right)^T\cdot H_{\theta_k}\cdot\left(\frac{\nabla_\theta L_{\theta_k}(\theta)}{\lambda}\Big|_{\theta=\theta_k}\cdot H_{\theta_k}^{-1}\right)-\lambda\delta$$

$$= -\frac{1}{2\lambda}\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)^T\cdot H_{\theta_k}^{-1}-\lambda\delta$$

To find out λ^* $\Rightarrow$ Calculate $\frac{\partial D(\lambda)}{\partial\lambda}=0$

$$\frac{\partial D(\lambda)}{\partial\lambda} = \frac{1}{2\lambda^2}\left[\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)^T H_{\theta_k}^{-1}\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)\right]-\delta=0$$

$$\Rightarrow \lambda^* = \lambda = \left[\frac{1}{2\delta}\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)^T H_{\theta_k}^{-1}\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)\right]^{\frac{1}{2}}$$

(b) Calculate $\frac{\partial L(\theta,\lambda^*)}{\partial\theta}=0$

$$\Rightarrow \frac{\partial L(\theta,\lambda^*)}{\partial\theta} = -\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)+\lambda^*(\theta-\theta_k)H_{\theta_k}$$

$$\Rightarrow \theta^* = \theta = \theta_k - \left(\frac{1}{\lambda^*}\right)\nabla_{\theta_k}(\theta)\big|_{\theta=\theta_k}$$

← step size α

And $\alpha = \frac{1}{\lambda^*} = \left[\frac{1}{2\delta}\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)^T H_{\theta_k}^{-1}\left(\nabla_\theta L_{\theta_k}(\theta)\big|_{\theta=\theta_k}\right)\right]^{-\frac{1}{2}}$

Problem 3 (PPO With a Clipped Objective) (10 points)

Recall from Lecture 17 that the PPO-Clip updates the policy iteratively by maximizing the following objective function:

$$L^{\text{clip}}(\theta;\theta_k) := \mathbb{E}_{s\sim d_\mu^{\pi_{\theta_k}},a\sim\pi_{\theta_k}(\cdot|s)}\left[L^{\text{clip}}_{s,a}\right], \tag{8}$$

$$\text{where} \quad L^{\text{clip}}_{s,a}(\theta;\theta_k) := \min\left\{\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\theta_k}(s,a), \text{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\theta_k}(s,a)\right\}. \tag{9}$$

As mentioned in class, another possibility is to consider the following variant:

$$\widetilde{L}^{\text{clip}}_{s,a}(\theta;\theta_k) := \text{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}, 1-\varepsilon, 1+\varepsilon\right)\cdot A^{\theta_k}(s,a). \tag{10}$$

Could you explain the behavioral differences between the two objective functions $L^{\text{clip}}_{s,a}(\theta;\theta_k)$ and $\widetilde{L}^{\text{clip}}_{s,a}(\theta;\theta_k)$? (Hint: For a clear comparison, it could be better to make a table and illustrative plots similar to Figure 1 below)

	$p_t(\theta) > 0$	A_t	**Return Value of *min***	**Objective is Clipped**	**Sign of Objective**	**Gradient**
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
3	$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
4	$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
5	$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
6	$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓

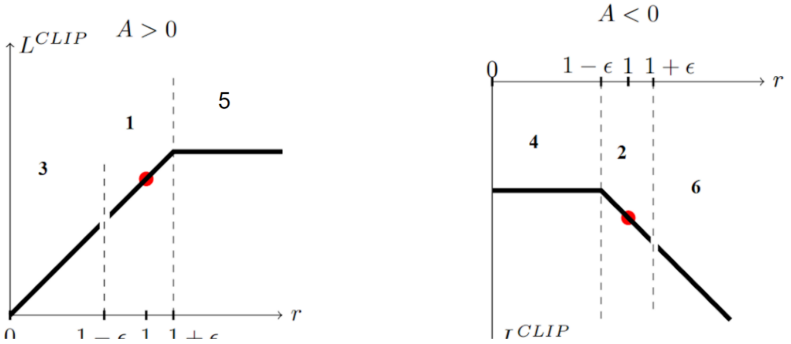
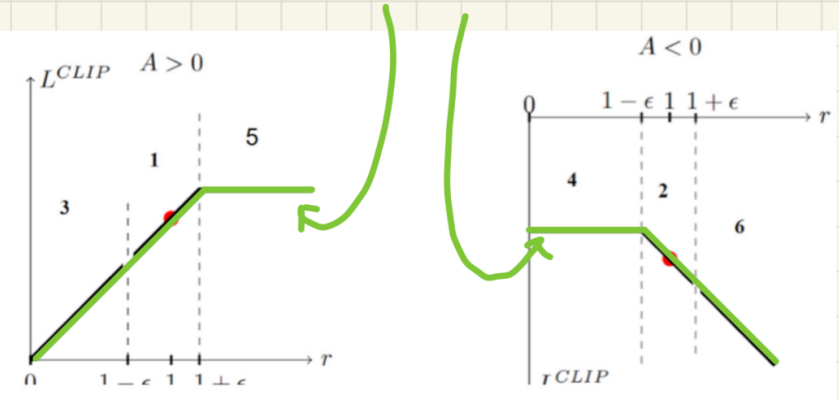


Figure 1: Behavior of the original PPO-clip objective.

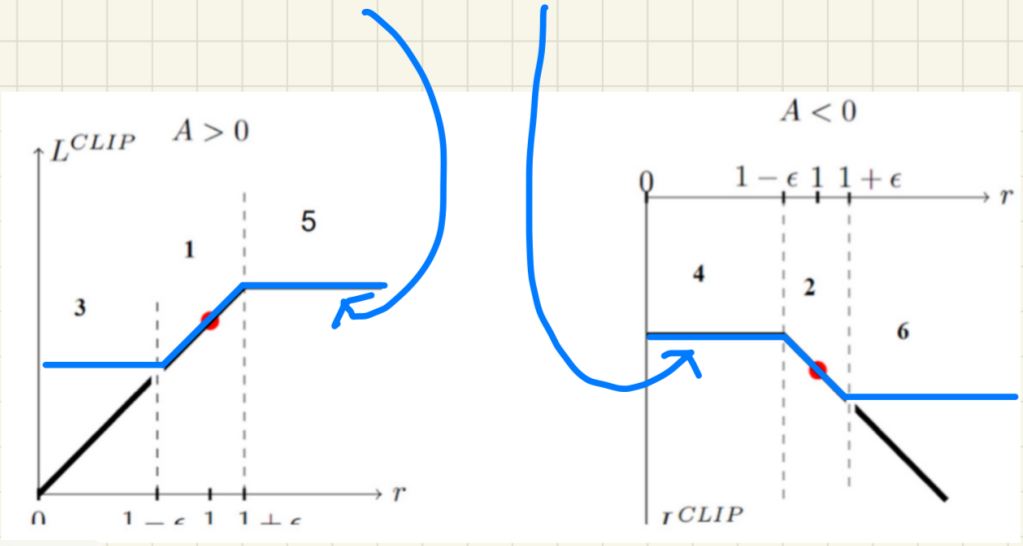
For $L^{clip}_{s,a}(\theta, \theta_k)$, The objective function graph is just as green line

$$L^{\text{clip}}_{s,a}(\theta;\theta_k) := \min\left\{\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}A^{\theta_k}(s,a), \text{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}, 1-\varepsilon, 1+\varepsilon\right)A^{\theta_k}(s,a)\right\}$$



But for $\tilde{L}^{clip}_{s,a}(\theta, \theta_k)$, the objective function is like the Blue line in the graph below.

$$\tilde{L}^{\text{clip}}_{s,a}(\theta;\theta_k) := \text{clip}\left(\frac{\pi_\theta(a\mid s)}{\pi_{\theta_k}(a\mid s)}, 1-\varepsilon, 1+\varepsilon\right)\cdot A^{\theta_k}(s,a).$$



We can see that $\tilde{L}^{clip}_{s,a}(\theta, \theta_k)$ will clip the objective function at both $1-\varepsilon$ and $1+\varepsilon$, but $L^{clip}_{s,a}(\theta, \theta_k)$ only clip one side (if $A>0$, clip $1+\varepsilon$; $A<0$, clip $1-\varepsilon$)

black pen (original table) red pen (I modified)

The table for $L^{clip}_{s,a}(\theta, \theta_k)$ and $\tilde{L}^{clip}_{s,a}(\theta, \theta_k)$

	$p_t(\theta) > 0$	A_t	Return Value of *min*	Objective is Clipped	Sign of Objective	Gradient
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	+	$p_t(\theta)A_t$ $P_t(\theta)A_t$	no no	+	✓ ✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	−	$p_t(\theta)A_t$ $P_t(\theta)A_t$	no no	−	✓ ✓
3	$p_t(\theta) < 1-\epsilon$	+	$p_t(\theta)A_t$ $(1-\varepsilon)A_t$	no yes	+	✓ 0
4	$p_t(\theta) < 1-\epsilon$	−	$(1-\epsilon)A_t$ $(1-\varepsilon)A_t$	yes yes	−	0 0
5	$p_t(\theta) > 1+\epsilon$	+	$(1+\epsilon)A_t$ $(1+\varepsilon)A_t$	yes yes	+	0 0
6	$p_t(\theta) > 1+\epsilon$	−	$p_t(\theta)A_t$ $(1+\varepsilon)A_t$	no yes	−	✓ 0

$\tilde{L}^{clip}_{s,a}(\theta, \theta_k)$

Problem 4 (Deep Deterministic Policy Gradient for Continuous Control) (30+20=50 points)

In this problem, we will implement the deep determinisitc policy gradient (DDPG) algorithm with the help of neural function approximators: You may write your code in either PyTorch or TensorFlow (though the sample code presumes PyTorch framework). Moreover, you are recommended to use Tensorboard or Weight & Bias to keep track of the loss terms and other related quantities of your implementation. If you are a beginner in learning the deep learning framework, please refer to the following tutorials:

(a) We start by solving the simple "Pendulum-v0" problem (https://gym.openai.com/envs/Pendulum-v0/) using the DDPG algorithm. Read through **ddpg.py** and then implement the member functions of the classes **Actor, Critic, and DDPG** as well as the function **train**. Please briefly summarize your results (including the snapshots of the Tensorboard or Weight & Bias record) in the report and document all the hyperparameters (e.g. learning rates, NN architecture, and batch size) of your experiments. (Note: Pendulum is a rather basic environment mostly for the purpose of sanity check. As a result, typically it would take no more than 300 episodes to reach a well-performing policy)

Below is my hyper parameter :

```
num_episodes = 300
gamma = 0.995
tau = 0.002
hidden_size = 100
noise_scale = 0.3
replay_size = 100000
batch_size = 256
updates_per_step = 1
print_freq = 1
ewma_reward = 0
rewards = []
ewma_reward_history = []
total_numsteps = 0
updates = 0
```

I use three linear layer for actor. Two activation function Relu, and the output layer use tanh. I normalize the ouput with a low and high value to calculate the ratio to normalization.

```
class Actor(nn.Module):
    def __init__(self, hidden_size, num_inputs, action_space):
        super(Actor, self).__init__()
        self.action_space = action_space
        num_outputs = action_space.shape[0]

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Construct your own actor network
        self.layer1 = nn.Linear(num_inputs, hidden_size)
        self.layer2 = nn.Linear(hidden_size, hidden_size)
        self.output = nn.Linear(hidden_size, num_outputs)

        ########## END OF YOUR CODE ##########

    def forward(self, inputs):
```

```
    def forward(self, inputs):

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Define the forward pass your actor network
        x = self.layer1(inputs)
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        action = torch.tanh(self.output(x))
        low = -2
        high = 6
        return low + (high - low) / (1 - (-1)) * action
        ########## END OF YOUR CODE ##########
```

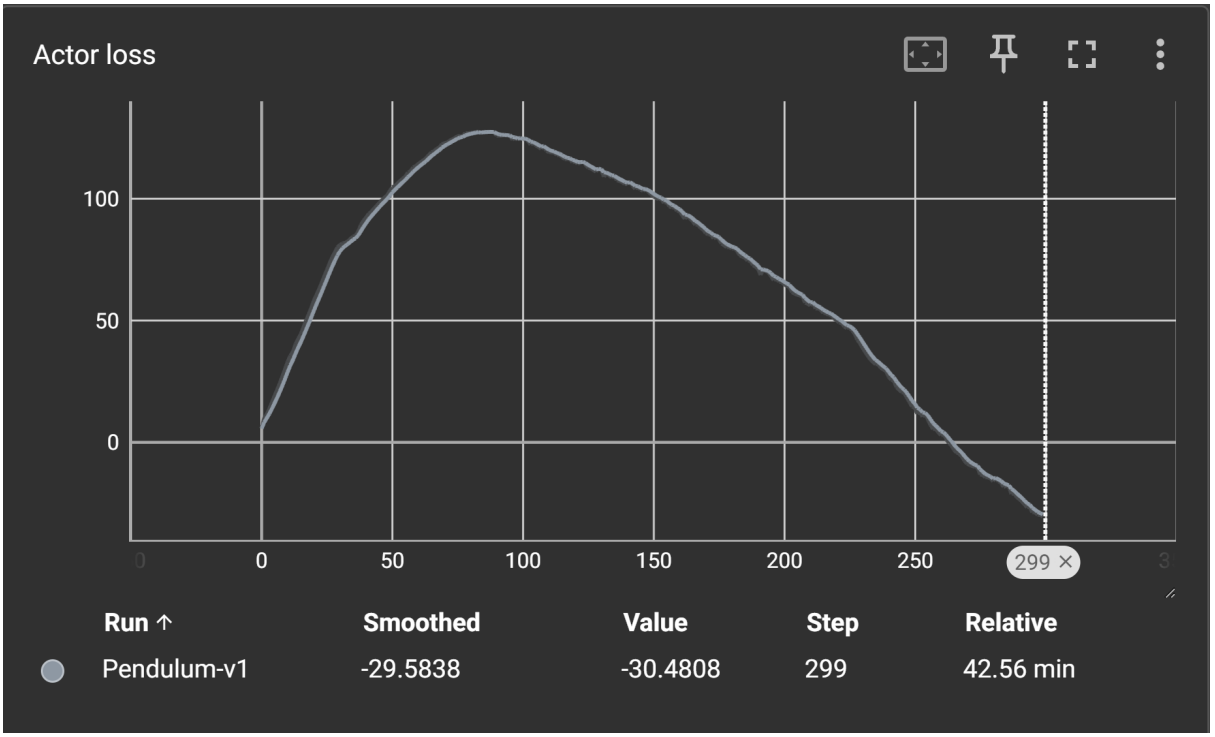
I use three linear layer for critic. Two activation function Relu, and the output layer directly output one element which is the action.

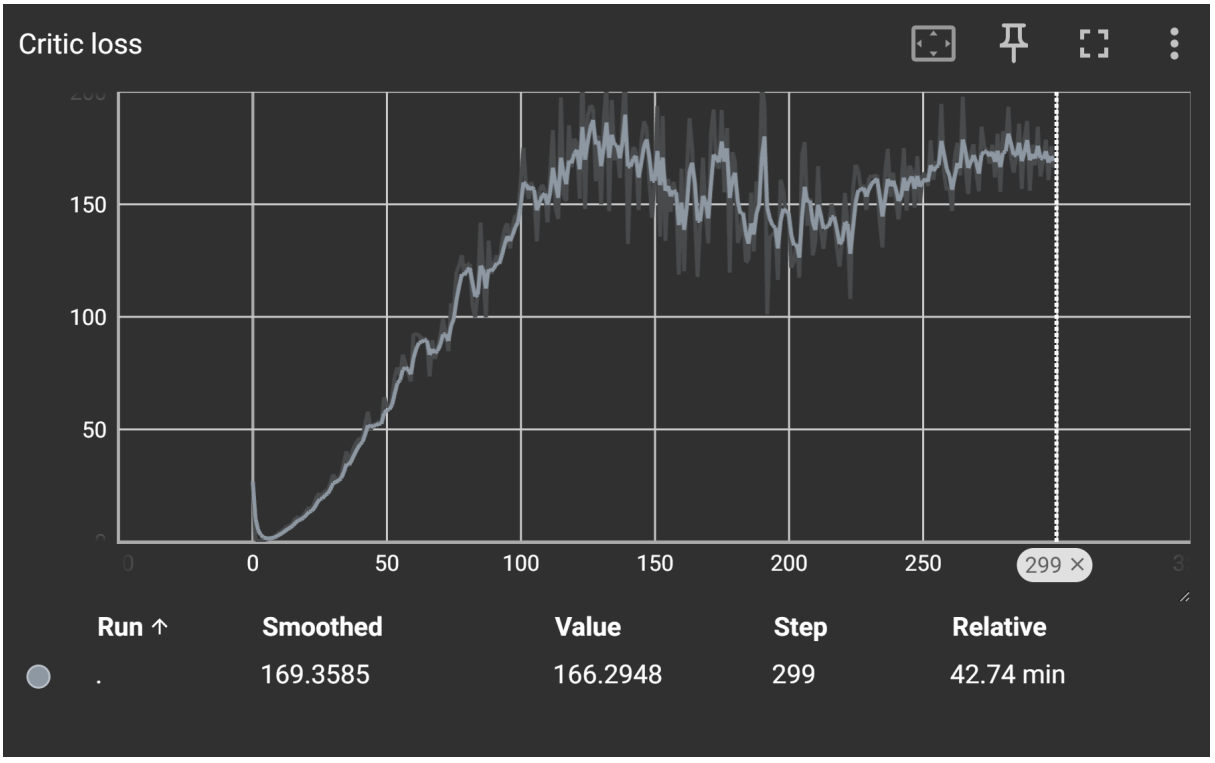
```
    def __init__(self, hidden_size, num_inputs, action_space):
        super(Critic, self).__init__()
        self.action_space = action_space
        num_outputs = action_space.shape[0]

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Construct your own critic network
        self.layer1 = nn.Linear(num_inputs+num_outputs, hidden_size)
        self.layer2 = nn.Linear(hidden_size, hidden_size)
        self.output = nn.Linear(hidden_size, 1)
        ########## END OF YOUR CODE ##########
```

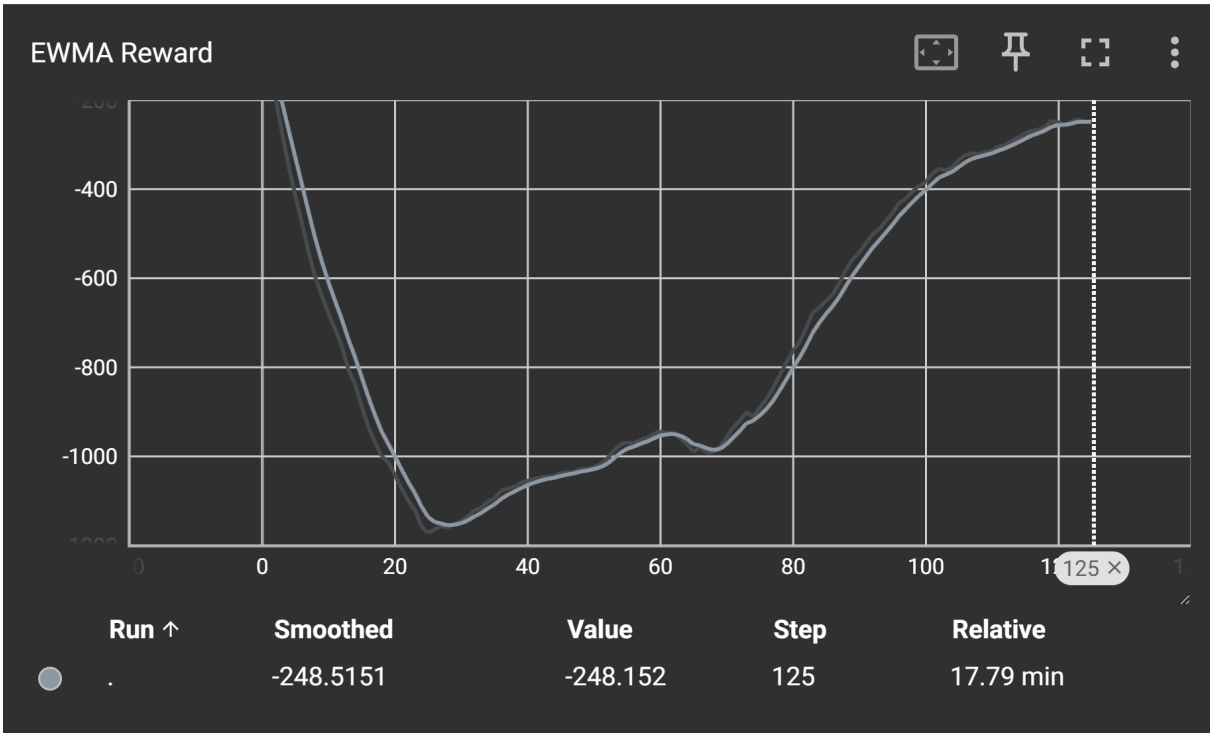
```
    def forward(self, inputs, actions):

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Define the forward pass your critic network
        x = self.layer1(torch.cat([inputs, actions], dim=-1))
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        action_value = self.output(x)
        return action_value
        ########## END OF YOUR CODE ##########
```

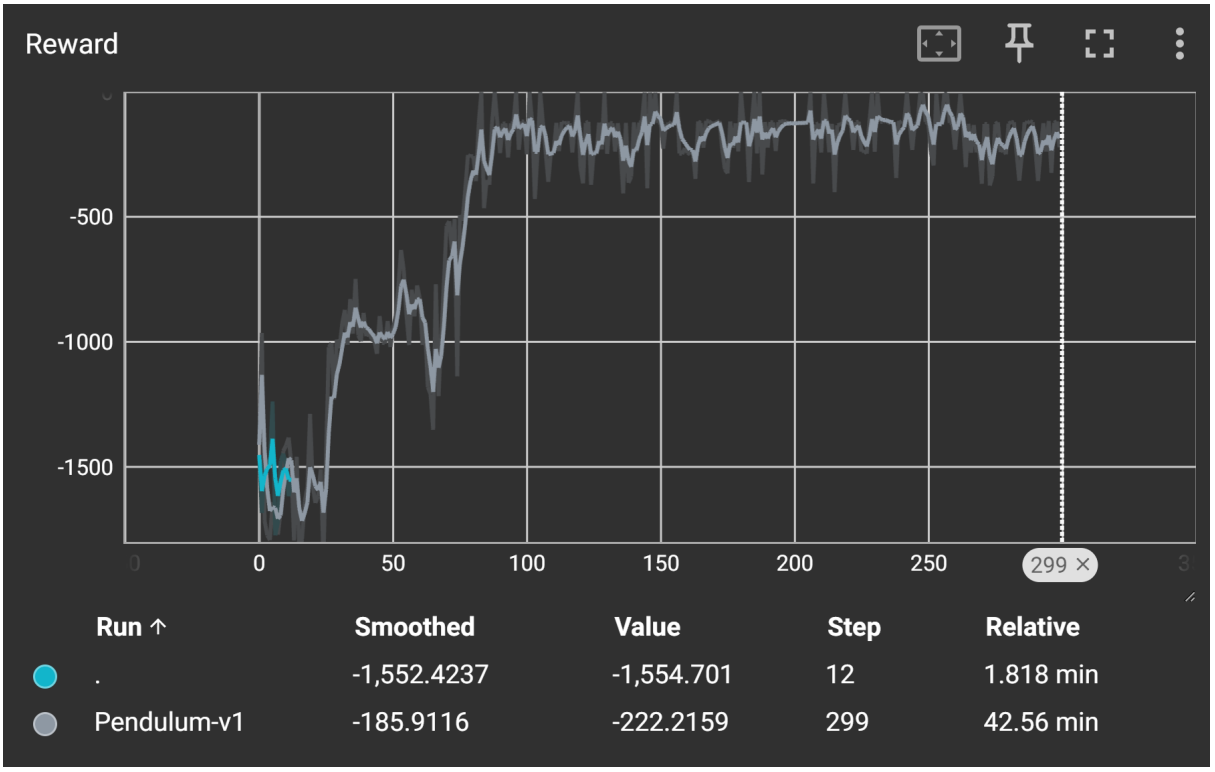
I run 300 episode. The last 20 episode has average Reward no more than 300, so i thought it had reached a well-performing policy.

```
Episode: 282, length: 200, reward: -121.60, ewma reward: -348.29
Episode: 283, length: 200, reward: -343.49, ewma reward: -348.05
Episode: 284, length: 200, reward: -362.20, ewma reward: -348.76
Episode: 285, length: 200, reward: -349.16, ewma reward: -348.78
Episode: 286, length: 200, reward: -132.54, ewma reward: -337.96
Episode: 287, length: 200, reward: -135.55, ewma reward: -327.84
Episode: 288, length: 200, reward: -126.04, ewma reward: -317.75
Episode: 289, length: 200, reward: -129.69, ewma reward: -308.35
Episode: 290, length: 200, reward: -231.25, ewma reward: -304.50
Episode: 291, length: 200, reward: -370.42, ewma reward: -307.79
Episode: 292, length: 200, reward: -247.68, ewma reward: -304.79
Episode: 293, length: 200, reward: -142.36, ewma reward: -296.66
Episode: 294, length: 200, reward: -268.49, ewma reward: -295.26
Episode: 295, length: 200, reward: -130.77, ewma reward: -287.03
Episode: 296, length: 200, reward: -327.37, ewma reward: -289.05
Episode: 297, length: 200, reward: -137.44, ewma reward: -281.47
Episode: 298, length: 200, reward: -129.62, ewma reward: -273.88
Episode: 299, length: 200, reward: -221.06, ewma reward: -271.23
```




Critic loss
150
100
50
0
0
50
100
150
200
250
299
Run
Smoothed
Value
Step
Relative
.
169.3585
166.2948
299
42.74 min


EWMA Reward
-400
-600
-800
-1000
0
0
20
40
60
80
100
125
Run
Smoothed
Value
Step
Relative
.
-248.5151
-248.152
125
17.79 min



(b) Based on the code for (a), adapt your DDPG algorithm to solve the "HalfCheetah" locomotion task in MuJoCo (https://gymnasium.farama.org/main/environments/mujoco/half_cheetah/). Save your code in another file named **ddpg_cheetah.py**. Please add comments to your code whenever needed for better readability. Recall from HW1 that you have already installed the MuJoCo package and shall be able to directly play with the Halfcheetah locomotion task. Again, please briefly summarize your results in the report and document all the hyperparameters of your experiments. (Note: As HalfCheetah is a more challenging environment than Pendulum, it would take a bit more training time to reach a well-performing policy, say reaching a score of 4000 within 300k training steps or so. Also, it might require some efforts to tune the hyperparameters, e.g., learning rates and batch size. As the main purpose of this homework is to help you get familiar with RL implementation for continuous control, there is NO hard requirement on the obtained returns of this Halfcheetah locomotion task. Just try your best and enjoy!)

Below is my hyper parameter :

```
def train(env_name):
        num_episodes = 300
        gamma = 0.995
        tau = 0.0005
        lr_a = 0.002
        lr_c = 0.006
        hidden_size = 128
        noise_scale = 0.3
        replay_size = 100000
        batch_size = 512
        updates_per_step = 1
        print_freq = 1
        ewma_reward = 0
        rewards = []
        ewma_reward_history = []
        total_numsteps = 0
        updates = 0
```

I use five linear layer for actor. four activation function Relu, and the output layer use tanh. I normalize the ouput with a low and high value to calculate the ratio to normalization.

```
class Actor(nn.Module):
        def __init__(self, hidden_size, num_inputs, action_space):
                super(Actor, self).__init__()
                self.action_space = action_space
                num_outputs = action_space.shape[0]

                ########## YOUR CODE HERE (5~10 lines) ##########
                # Construct your own actor network
                self.layer1 = nn.Linear(num_inputs, hidden_size)
                self.layer2 = nn.Linear(hidden_size, hidden_size)
                self.layer3 = nn.Linear(hidden_size, hidden_size)
                self.layer4 = nn.Linear(hidden_size, hidden_size)
                self.output = nn.Linear(hidden_size, num_outputs)

                ########## END OF YOUR CODE ##########
```

```
    def forward(self, inputs):

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Define the forward pass your actor network
        x = self.layer1(inputs)
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        x = self.layer3(x)
        x = F.relu(x)
        x = self.layer4(x)
        x = F.relu(x)
        action = torch.tanh(self.output(x))
        low = -2
        high = 6
        return low + (high - low) / (1 - (-1)) * action
        ########## END OF YOUR CODE ##########
```

I use five linear layer for critic. four activation function Relu, and the output layer directly output one element which is the action.

```
    def __init__(self, hidden_size, num_inputs, action_space):
        super(Critic, self).__init__()
        self.action_space = action_space
        num_outputs = action_space.shape[0]

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Construct your own critic network
        self.layer1 = nn.Linear(num_inputs+num_outputs, hidden_size)
        self.layer2 = nn.Linear(hidden_size, hidden_size)
        self.layer3 = nn.Linear(hidden_size, hidden_size)
        self.layer4 = nn.Linear(hidden_size, hidden_size)
        self.output = nn.Linear(hidden_size, 1)
        ########## END OF YOUR CODE ##########
```

```
    def forward(self, inputs, actions):

        ########## YOUR CODE HERE (5~10 lines) ##########
        # Define the forward pass your critic network
        x = self.layer1(torch.cat([inputs, actions], dim=-1))
        x = F.relu(x)
        x = self.layer2(x)
        x = F.relu(x)
        x = self.layer3(x)
        x = F.relu(x)
        x = self.layer4(x)
        x = F.relu(x)
        action_value = self.output(x)
        return action_value
        ########## END OF YOUR CODE ##########
```

I run 200 episode. I had tried so many hyperparameter combination, hope it can achieve well performing policy. However, I lost. So the above hyperparameter is one of the hyperparameter i had tried as an example in this report.

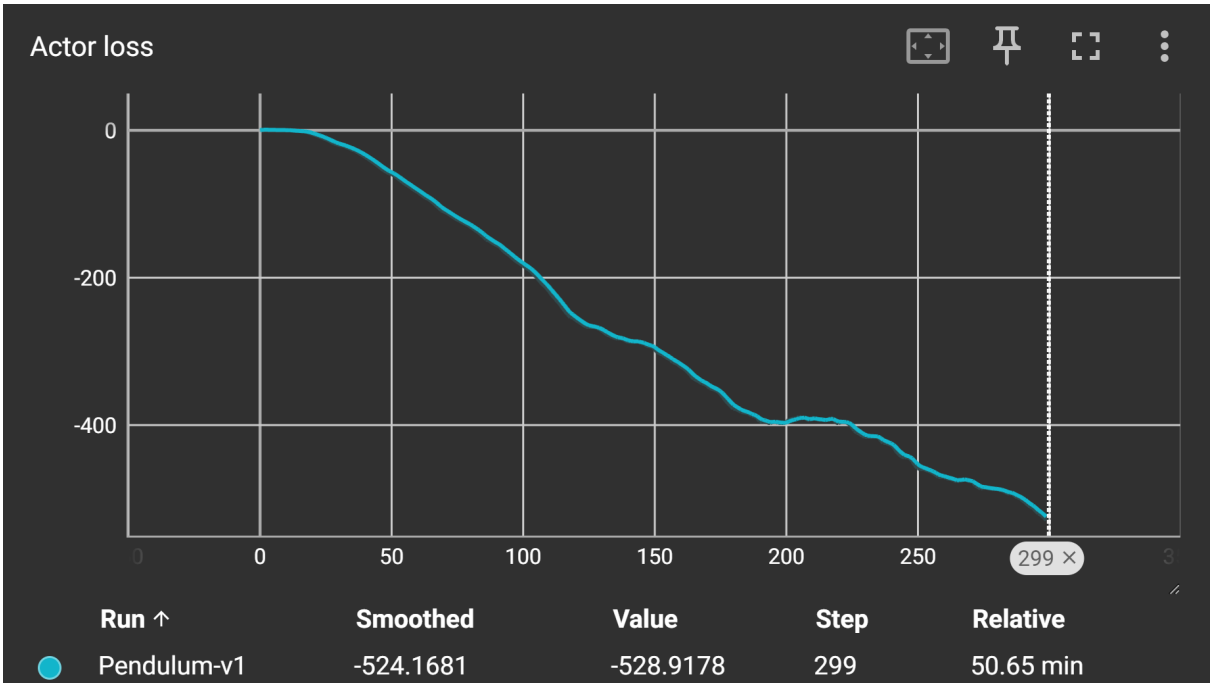
In this example, it was great with Reward more than 1000 around 100 episode.

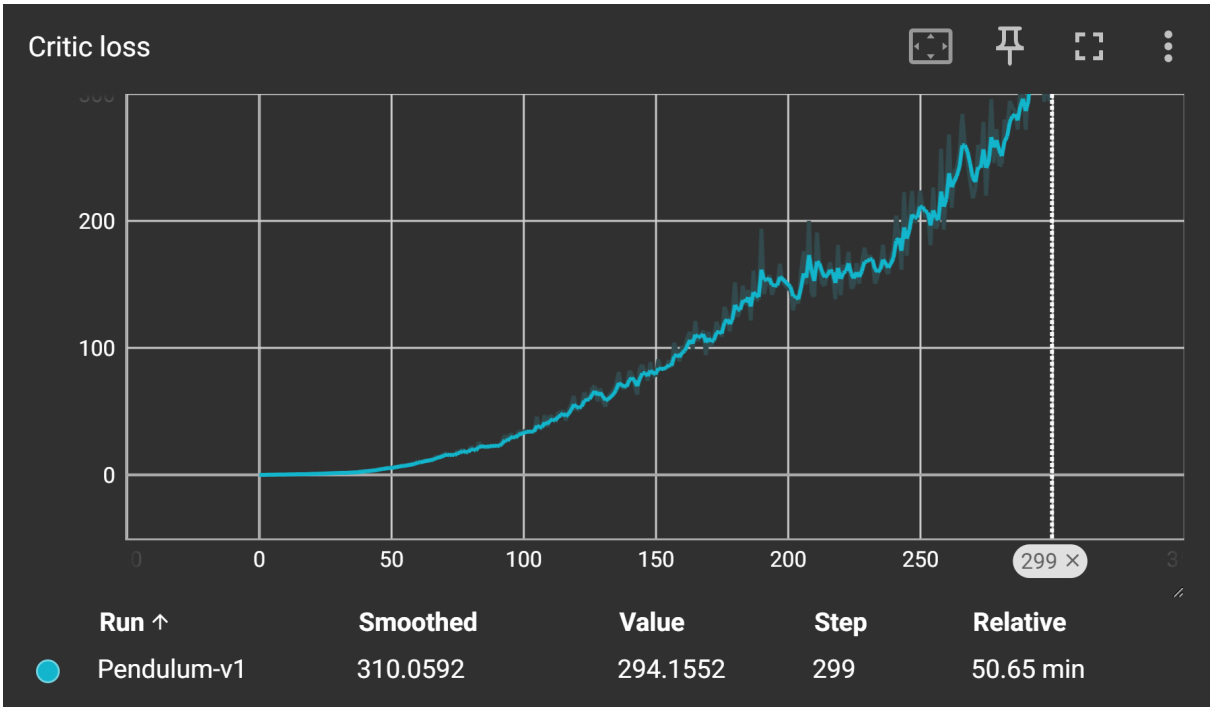
```
Episode: 96, length: 1000, reward: 1193.26, ewma reward: 234.45
Episode: 97, length: 1000, reward: 1183.44, ewma reward: 281.90
Episode: 98, length: 1000, reward: 1076.03, ewma reward: 321.61
Episode: 99, length: 1000, reward: 989.65, ewma reward: 355.01
Episode: 100, length: 1000, reward: 672.03, ewma reward: 370.86
Episode: 101, length: 1000, reward: 924.58, ewma reward: 398.55
Episode: 102, length: 1000, reward: 792.00, ewma reward: 418.22
Episode: 103, length: 1000, reward: 905.88, ewma reward: 442.60
Episode: 104, length: 1000, reward: 1189.10, ewma reward: 479.93
Episode: 105, length: 1000, reward: 1220.34, ewma reward: 516.95
Episode: 106, length: 1000, reward: 1135.05, ewma reward: 547.85
```

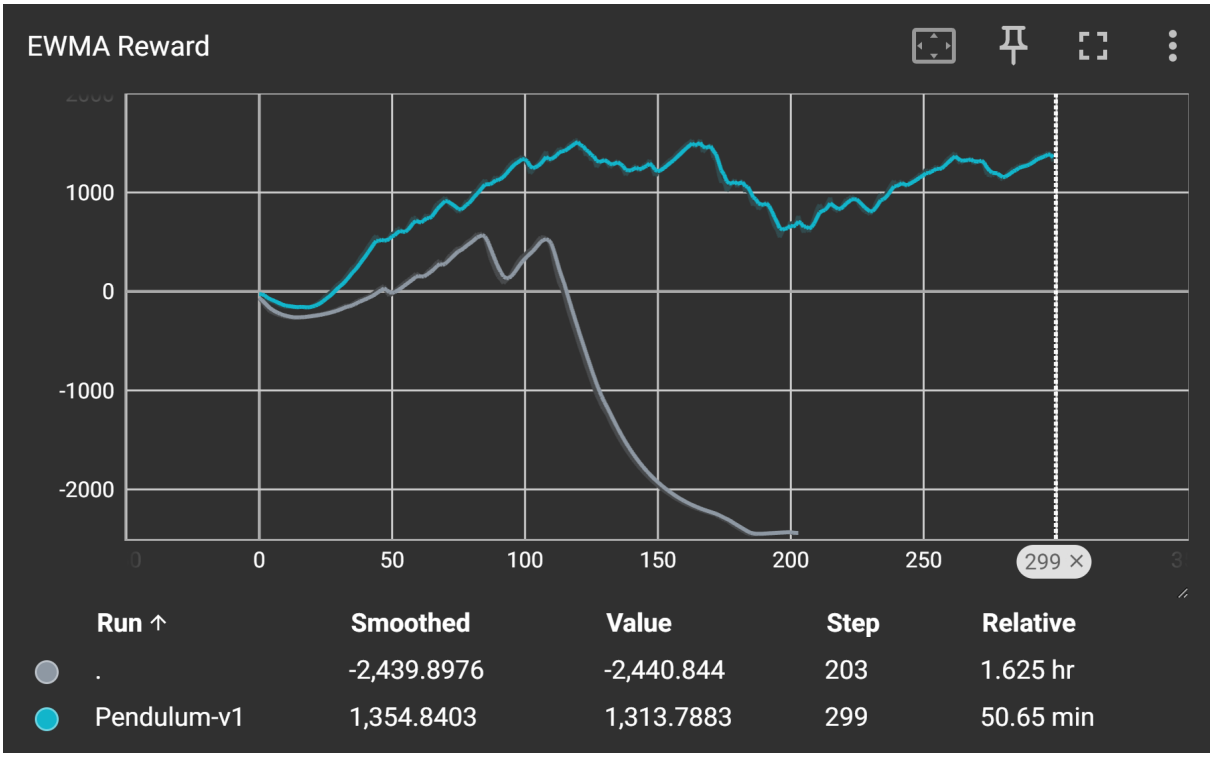
But it falied and Reward rapidly drop down to -2400.

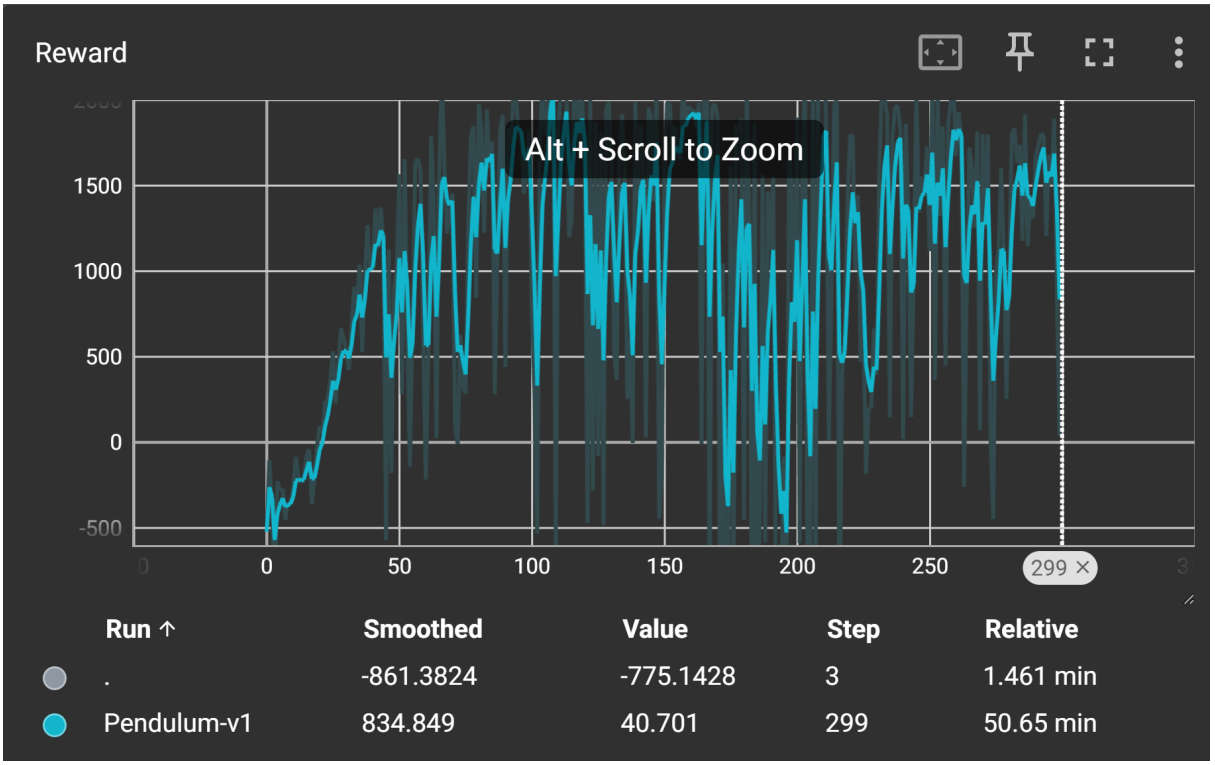
```
Episode: 190, length: 1000, reward: -2403.27, ewma reward: -2445.56
Episode: 191, length: 1000, reward: -2405.20, ewma reward: -2443.54
Episode: 192, length: 1000, reward: -2406.12, ewma reward: -2441.67
Episode: 193, length: 1000, reward: -2408.77, ewma reward: -2440.02
Episode: 194, length: 1000, reward: -2405.39, ewma reward: -2438.29
Episode: 195, length: 1000, reward: -2406.67, ewma reward: -2436.71
Episode: 196, length: 1000, reward: -2403.69, ewma reward: -2435.06
Episode: 197, length: 1000, reward: -2404.31, ewma reward: -2433.52
Episode: 198, length: 1000, reward: -2406.93, ewma reward: -2432.19
Episode: 199, length: 1000, reward: -2403.87, ewma reward: -2430.78
Episode: 200, length: 1000, reward: -2405.65, ewma reward: -2429.52
```

By the way, the following TernsorBoard graph has wrong curve name. I forgot to change ‘Pendulum-v1’ into ‘HalfCheetah-v2’. Please forgive me.




Critic loss
200
100
0
0
50
100
150
200
250
299
Run ↑
Smoothed
Value
Step
Relative
Pendulum-v1
310.0592
294.1552
299
50.65 min


EWMA Reward
1000
0
-1000
-2000
0
50
100
150
200
250
299
Run ↑
Smoothed
Value
Step
Relative
.
-2,439.8976
-2,440.844
203
1.625 hr
Pendulum-v1
1,354.8403
1,313.7883
299
50.65 min



I will try to tuning the hyperparamter to reach Reward more than 4000 in the future. But now, I need to submit my assignment before the deadline.