535514 Spring 2024: Reinforcement Learning **(Due: 2024/05/10, 21:00)**

Homework 3: DDPG, TRPO, and PPO

Submission Guidelines: Your deliverables shall consist of 2 separate files – (i) A PDF file: Please compile all your write-ups into one .pdf file (photos/scanned copies are acceptable; please make sure that the electronic files are of good quality and reader-friendly); (ii) A zip file: Please compress all your source code into one .zip file. Please submit your deliverables via E3.

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) $\nabla_\theta L_{\pi_{\theta_1}}(\pi_\theta)|_{\theta=\theta_1} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_1}$.

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):

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

For ease of notation, we define $\lambda^* := \arg\max_{\lambda\ge 0} D(\lambda)$ 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?

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:

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

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

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

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

Could you explain the behavioral differences between the two objective functions $L_{s,a}^{\text{clip}}(\theta;\theta_k)$ and $\widetilde{L}_{s,a}^{\text{clip}}(\theta;\theta_k)$? (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	$-$	✓

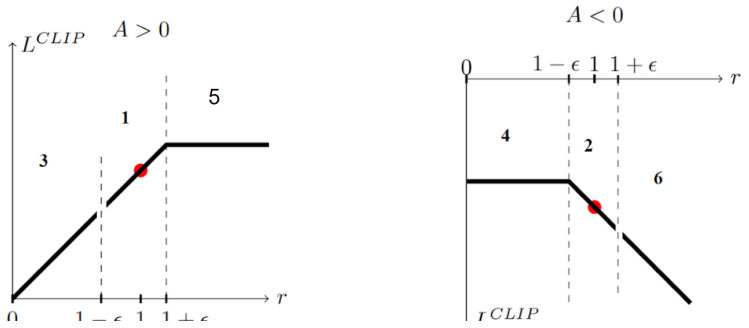


Figure 1: Behavior of the original PPO-clip objective.

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:

- PyTorch: https://pytorch.org/tutorials/
- Tensorflow: https://www.tensorflow.org/tutorials

For the deliverables, please submit the following:

- Technical report: Please summarize all your experimental results in 1 single report (and please be brief)
- All your source code
- Your well-trained models (both the actor and critic networks) saved in either .pth files or .ckpt files

(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)

(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!)

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) $\nabla_\theta L_{\pi_{\theta_1}}(\pi_\theta)|_{\theta=\theta_1} = \nabla_\theta \eta(\pi_\theta)|_{\theta=\theta_1}$.

$\theta \to$ new $\to \tilde{\pi}$

$\theta_1 \to$ old $\to \pi$

(i)

$$L_{\pi_{\theta_1}}(\pi_{\theta_1}) = \eta(\pi_{\theta_1}) + \sum_s d_M^{\pi_{\theta_1}}(s) \sum_a \pi_{\theta_1}(a|s) \cdot A^{\pi_{\theta_1}}(s,a) \quad \cdots \text{by (1)}$$

$$\sum_a \pi_{\theta_1}(a|s) \cdot A^{\pi_{\theta_1}}(s,a) = \sum_{a\in\mathcal{A}} \pi_{\theta_1}(a|s) \cdot \left(Q^{\pi_{\theta_1}}(s,a) - V^{\pi_{\theta_1}}(s) \right) = 0$$

(ii)

$$\eta(\pi_\theta) = \eta(\pi_{\theta_1}) + \sum_s d_M^{\pi_\theta}(s) \sum_a \pi_\theta(a|s) \cdot A^{\pi_{\theta_1}}(s,a)$$

$$L_{\pi_{\theta_1}}(\pi_\theta) = \eta(\pi_{\theta_1}) + \sum_s d_M^{\pi_{\theta_1}}(s) \sum_a \pi_\theta(a|s) \cdot A^{\pi_{\theta_1}}(s,a)$$

$$\because \nabla_\theta d_M^{\pi_\theta}(s) \cdot f(\theta)\big|_{\theta=\theta_1} = \nabla_\theta d_M^{\pi_{\theta_1}}(s) \cdot f(\theta)\big|_{\theta=\theta_1}$$

$$\Rightarrow \nabla_\theta L_{\pi_{\theta_1}}(\pi_\theta)\big|_{\theta=\theta_1} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_1}$$

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

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

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.i)

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

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

$$D(\lambda) = \min_\theta \mathcal{L}(\theta,\lambda) \Rightarrow \text{To find min} \to \text{let } \nabla_\theta \mathcal{L}(\theta,\lambda) = 0$$

$$\Rightarrow -\nabla_\theta L_{\theta_k}(\theta) + \lambda H_{\theta_k}(\theta-\theta_k) = 0 \Rightarrow \theta^* = \theta_k - \frac{1}{\lambda} H_{\theta_k}^{-1} \nabla_\theta L_{\theta_k}(\theta)$$

By the dual func: $D(\lambda) \le \mathcal{L}(\theta,\lambda) \Rightarrow D(\lambda^*) = \mathcal{L}(\theta^*,\lambda^*)$

$$D(\lambda^*) = -(\nabla_\theta L_{\theta_k}(\theta))^T \cdot \left(\frac{1}{\lambda} H_{\theta_k}^{-1} \nabla_\theta L_{\theta_k}(\theta)\right) + \frac{1}{2\lambda}(\nabla_\theta L_{\theta_k}(\theta))^T \underbrace{H_{\theta_k}^{-1} H_{\theta_k}}_{I} H_{\theta_k}^{-1} \nabla_\theta L_{\theta_k}(\theta) - \lambda^*\delta$$

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

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

$\because \frac{d}{d\lambda} D(\lambda)\Big|_{\lambda=\lambda^*} = 0 \Rightarrow 0 = \frac{1}{2\lambda^2}\left[\left(\nabla_\theta L_{\theta_k}(\theta)\Big|_{\theta=\theta_k}\right)^T \cdot H_{\theta_k}^{-1}\left(\nabla_\theta L_{\theta_k}(\theta)\Big|_{\theta=\theta_k}\right)\right] - \delta$

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

(b.)

put λ^* into: $-\nabla_\theta L_{\theta_k}(\theta) + \lambda H_{\theta_k}(\theta-\theta_k) = 0 \Rightarrow \theta^* = \theta_k - \frac{1}{\lambda^*} H_{\theta_k}^{-1} \nabla_\theta L_{\theta_k}(\theta)$

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

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:

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

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

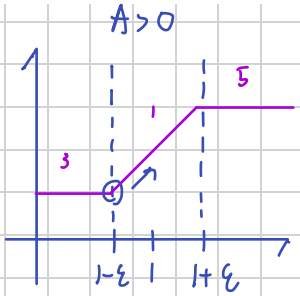
$p(t) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$

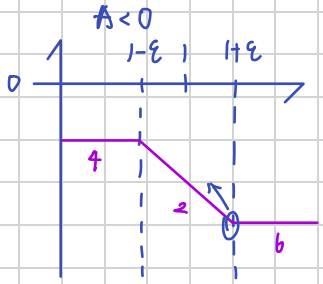
As mentioned in class, another possibility is to consider the following variant:

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

$\widetilde{L}^{\text{clip}}_{s,a}(\theta;\theta_k)$

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





constrain $L^{\text{clip}} \in [1-\varepsilon, 1+\varepsilon]$

$\widetilde{L}^{\text{clip}}_{s,a}(\theta;\theta_k)$ seems to be a more conservative strategy, which ensures the action probabilities of new π don't deviate to much from old π.

hw3

分類	
Date	@May 11, 2024 10:59 PM
Review	☐

DDPG - Pendulum-v1

Hyperparameters:

```
num_episodes = 500
gamma = 0.995
tau = 0.002
hidden_size = 128
noise_scale = 0.3
replay_size = 100000
batch_size = 256
updates_per_step = 1
```

Actor nn

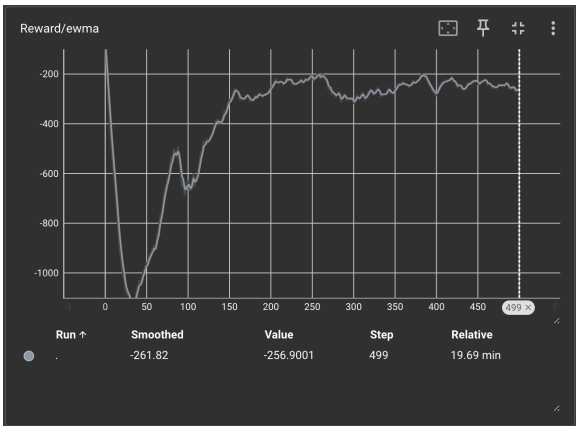
```
self.fc1 = nn.Linear(num_inputs, hidden_size)
self.fc2 = nn.Linear(hidden_size, hidden_size)
self.fc3 = nn.Linear(hidden_size, num_outputs)

ReLU for the activation function
tanh for the output layer
```

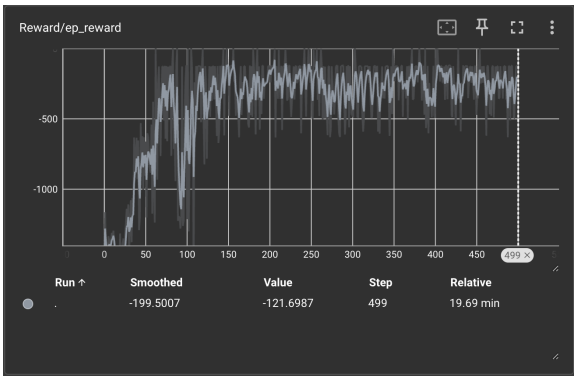
Critic nn

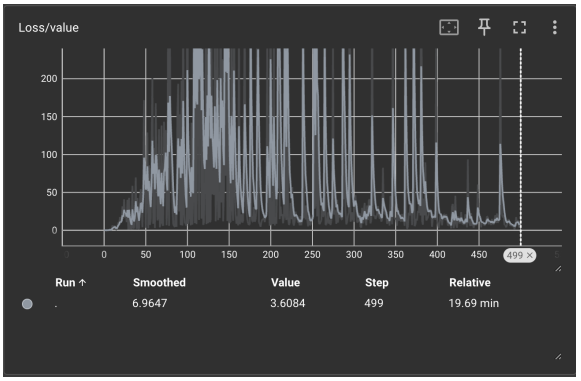
```
self.fc_state1 = nn.Linear(num_inputs, 64)
self.fc_state2 = nn.Linear(64, 64)
self.fc_action = nn.Linear(num_outputs, 64)

self.fc1 = nn.Linear(128, hidden_size)
```

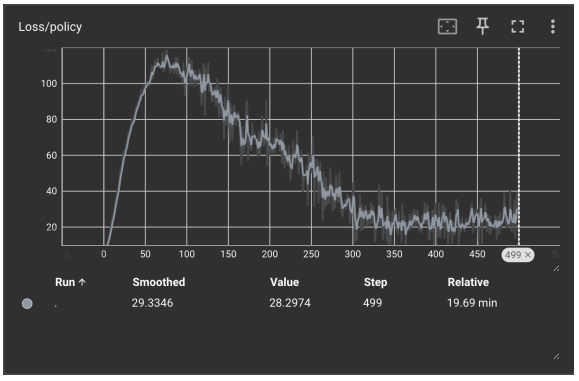
```
self.fc2 = nn.Linear(hidden_size, hidden_size)
self.fc3 = nn.Linear(hidden_size, 1)

ReLU for the activation function
Linear for the output layer
```

TensorBoard









Brief Conclusion & Notation

- Encountered Problems
  - The original implementation of my train function is:

```
while True:
    ########## YOUR CODE HERE (15~25 lines) ##########
    # 1. Interact with the env to get new (s,a,r,s') sa
    # 2. Push the sample to the replay buffer
    # 3. Update the actor and the critic
```

```
    # 1.
    action = agent.select_action(state, ounoise)
    next_state, reward, done, _ = env.step(action.numpy
    next_state = torch.Tensor([next_state])
    mask = torch.Tensor([done])

    # 2.
    memory.push( state, action, mask, next_state, torch

    # 3.
    if len(memory) > batch_size:
        for _ in range(updates_per_step):
            sampled_trans = memory.sample(batch_size)
            b_states = []
            b_action = []
            b_mask = []
            b_next_state = []
            b_reward = []
            for b in sampled_trans:
                b_states.append(b.state)
                b_action.append(b.action)
                b_mask.append(b.mask)
                b_next_state.append(b.next_state)
                b_reward.append(b.reward)

            value_loss, policy_loss = agent.update_para
                Transition(b_states,
                b_action,
                b_mask,
                b_next_state,
                b_reward)
            )
            updates += 1

    total_numsteps += 1
    state = next_state
```

- it seems to be wrong and cause the divergence of the rewards. I can’t find the reason and I had suffered a hard time for finding bugs. I insisted this implementation since the update_parameters function has:

```
state_batch = Variable(torch.cat(batch.state))
action_batch = Variable(torch.cat(batch.action))
reward_batch = Variable(torch.cat(batch.reward))
mask_batch = Variable(torch.cat(batch.mask))
next_state_batch = Variable(torch.cat(batch.next_state)
```

- I finally gave it up after the ddl sadly and turned to simply modifying the update_parameters function as:

```
state_batch = Variable(batch.state)
action_batch = Variable(batch.action)
reward_batch = Variable(batch.reward)
mask_batch = Variable(batch.mask)
next_state_batch = Variable(batch.next_state)
```

- With new train function

```
while True:
    # 1.
    action = agent.select_action(state, ounoise)
    next_state, reward, done, _ = env.step(action.numpy

    # 2.
    memory.push(state, action, done, next_state, reward

    # 3.
    if len(memory) > batch_size:
        for _ in range(updates_per_step):
            batch = memory.sample(batch_size)
            batch = Transition(state=torch.from_numpy(n
                               action=torch.from_numpy
                               mask=torch.from_numpy(n
                               next_state=torch.from_n
                               reward=torch.from_numpy
```

```
            # update the actor and the critic
            value_loss, policy_loss = agent.update_para
            updates += 1

    total_numsteps += 1
    state = next_state
    episode_reward += reward

    if done:
        break
    ########## END OF YOUR CODE ##########
```

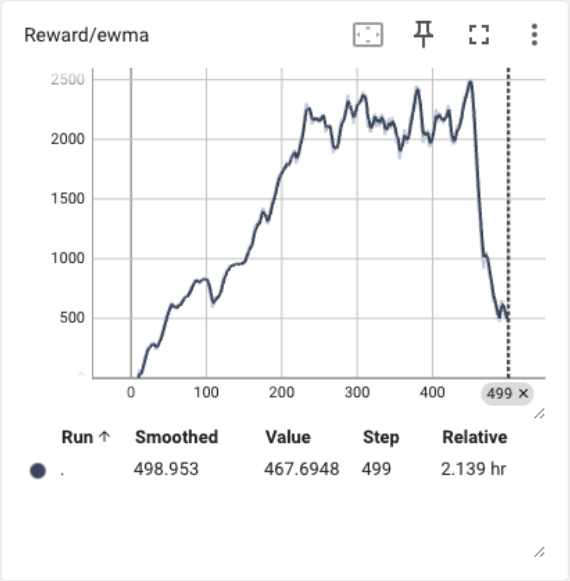
- Observations:
  - The convergence reward is around -200 and it showed up at around 200 episodes.

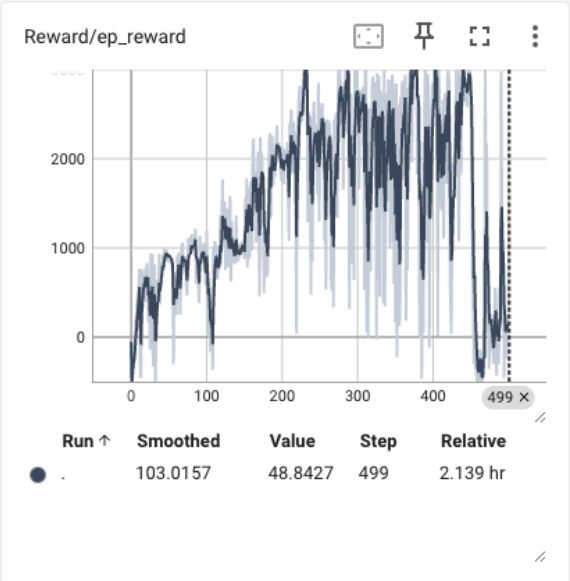
DDPG - cheetah

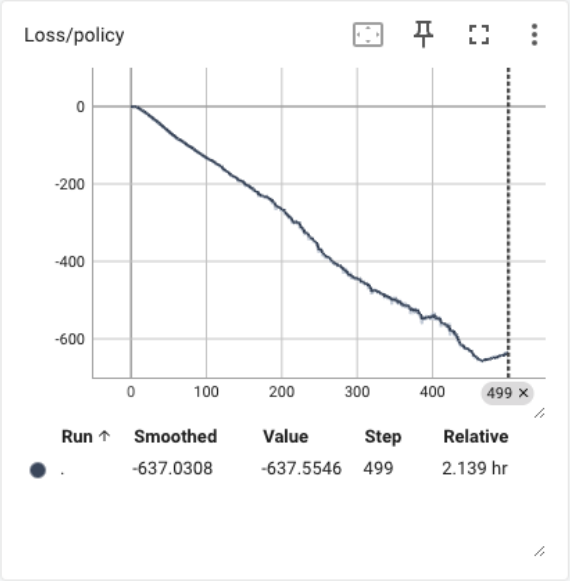
Hyperparameters

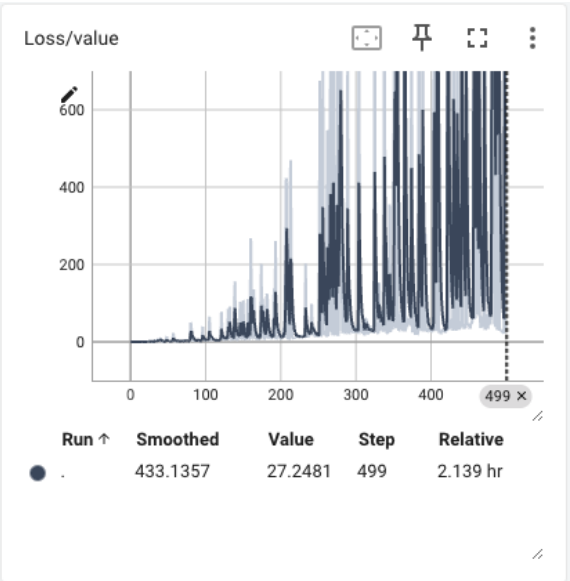
```
num_episodes = 500
gamma = 0.995
tau = 0.002
hidden_size = 128
noise_scale = 0.3
replay_size = 100000
batch_size = 256
updates_per_step = 1
print_freq = 1
ewma_reward = 0
rewards = []
ewma_reward_history = []
total_numsteps = 0
updates = 0
```

Tensor board









Google Colab Settings

- Set up environment for HalfCheetah-v3

```
# OS: Ubuntu 18.04.4 LTS x86_64
# Kernel: 4.18.0-15-generic
# CPU: Intel(R) Core(TM) i9-10920X CPU (24) @ 3.50GHz
```

```
# GPU: NVIDIA GeForce RTX 2080 Ti
!apt-get install -y \
    libgl1-mesa-dev \
    libgl1-mesa-glx \
    libglew-dev \
    libosmesa6-dev \
    software-properties-common

!apt-get install -y patchelf
!pip install git+https://github.com/Denys88/rl_games
!pip install envpool
!pip install gym

!pip install free-mujoco-py
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&
!pip install imageio==2.4.1
!pip install -U colabgymrender
```