DATA403 Final Presentation:

with HumanoidStandup-v4
and HalfCheetah-v4

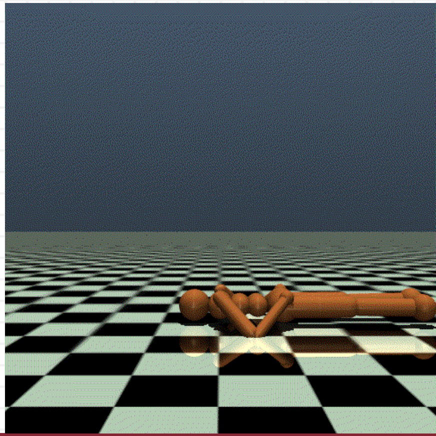
2021320303
Jiwon Jeong

Agenda

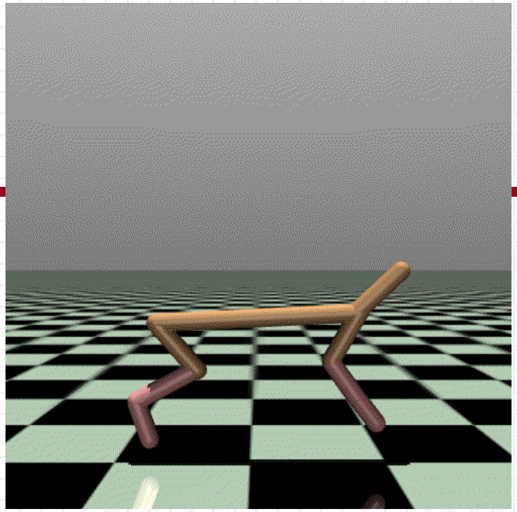
- Introduction
- Method
- Experiments & Analysis (HumanoidStandup-v4)
- Experiments & Analysis (HalfCheetah-v4)
- Conclusion

Introduction

- HumanoidStandup-v4
  - Goal: Make the humanoid standup and then keep it standing
  - Action Space: 17 continuous actions in [-0.4, 0.4]
    - Action represents the numerical torques applied at the hinge joints
  - Observation Space: 376 continuous space in (-inf, +inf)
    - State consists of positional values of different body parts of Humanoid
  - Why? Very Challenging and Interesting (Unsolved)



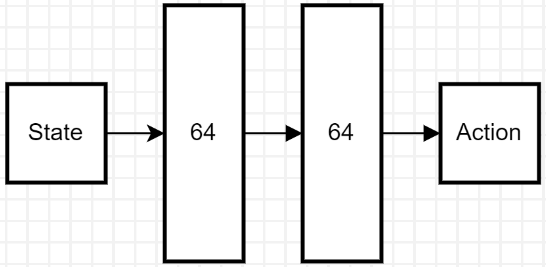
Introduction



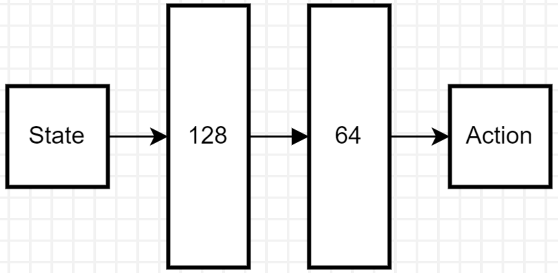
- HalfCheetah-v4
  - Goal: Make the cheetah run forward as fast as possible
  - Action Space: 6 continuous actions in [-1, 1]
    - Action represents the torques applied between links
  - Observation Space: 17 continuous space in (-inf, +inf)
    - State consists of positional values of different body parts of the cheetah
  - Why? Looks Fun !

Method

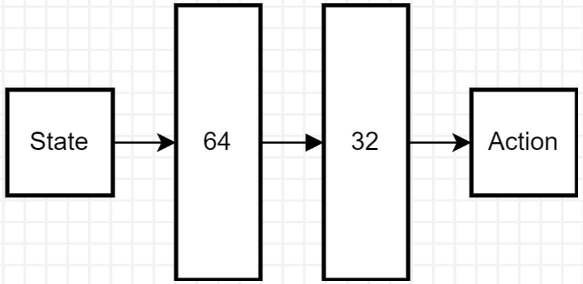
- Based on PPO (Proximal Policy Gradient)
- MLP with tanh



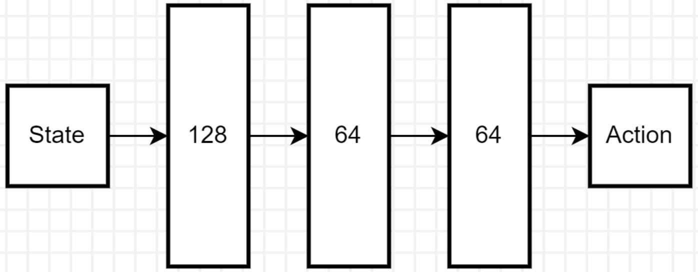
State → 64 → 64 → Action



State → 128 → 64 → Action



State → 64 → 32 → Action



State → 128 → 64 → 64 → Action

etc...

Method

- Gym Wrappers
  - FlattenObservation – flatten the observation
  - ClipAction – clip the continuous action to the valid bound
  - NormalizeObservation – normalize observation
  - TransformObservation – clip(obs, -10, 10)
  - NormalizeReward – normalize reward
  - TransformReward – clip(reward, -10, 10)

Method

- Robust Policy Optimization (ICLR 2023)

ROBUST POLICY OPTIMIZATION IN DEEP REINFORCEMENT LEARNING

Md Masudur Rahman & Yexiang Xue
Department of Computer Science
Purdue University
West Lafayette, IN 47907, USA
`{rahman64,yexiang}@purdue.edu`

  - Modified from PPO
  - RPO leverages a method of perturbing the distribution representing actions
  - Improved Performance compared to PPO

Method

- Robust Policy Optimization (ICLR 2023)

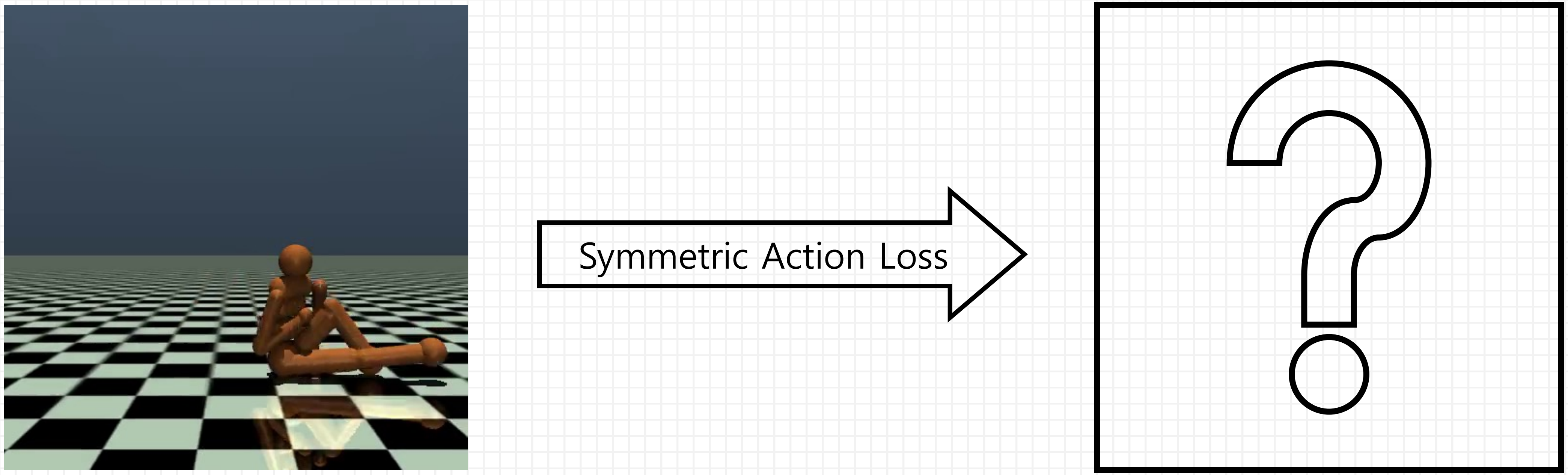
Algorithm 1 Robust Policy Optimization (RPO)

Initialize parameter vectors θ for policy network.
for each iteration **do**
  $\mathcal{D} \leftarrow \{\}$
  for each environment step **do**
    $\mu, \sigma \leftarrow \pi_\theta(.|s_t)$
    $a_t \sim \mathcal{N}(\mu, \sigma)$
    $s_{t+1} \sim P(s_{t+1}|s_t, a_t)$
    $r_t \sim R(s_t, a_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_t, s_{t+1})\}$
  end for
  for each observation s_t in $\mathcal{D}$ **do**
    $\mu, \sigma \leftarrow \pi_\theta(.|s_t)$
    $z \sim \mathcal{U}(-\alpha, \alpha)$
    $\mu' \leftarrow \mu + z$
    $prob \leftarrow \mathcal{N}(\mu', \sigma)$
    $logp \leftarrow prob(a_t)$
    Compute RL loss L_π using $logp$, a_t, and value function.
  end for
end for

```python
def get_action_and_value(self, x, action=None):
    ###################### Implement here : policy distribution #######################
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    action_mean = self.actor_mean(x)
    action_logstd = self.actor_logstd.expand_as(action_mean)
    action_std = torch.exp(action_logstd)
    probs = Normal(action_mean, action_std)
    if action is None:
        action = probs.sample()
    else: # RPO
        z = torch.FloatTensor(action_mean.shape).uniform_(-self.rpo_alpha, self.rpo_alpha).to(device)
        action_mean = action_mean + z
        probs = Normal(action_mean, action_std)
```

Method

- "Symmetric Action Loss" in HumanoidStandup setting
  - Why do Humanoid use only one arm or leg?



Method

- "Symmetric Action Loss" in HumanoidStandup setting
  - Gym Documentation: Action descriptions

6	Torque applied on the rotor between the right hip/thigh and the right shin	-0.4	0.4	right_knee	hinge	torque (N m)
10	Torque applied on the rotor between the left hip/thigh and the left shin	-0.4	0.4	left_knee	hinge	torque (N m)

  - Implementation (Use MSELoss() in torch.nn)

```
###################### Symmetric action loss ########################
sym_act_loss_1 = self.action_loss(probs.log_prob(action)[:, 3:7], probs.log_prob(action)[:, 7:11])
sym_act_loss_2 = self.action_loss(probs.log_prob(action)[:, 11:14], probs.log_prob(action)[:, 14:17])
sym_act_loss = 0.5 * (sym_act_loss_1 + sym_act_loss_2)
```

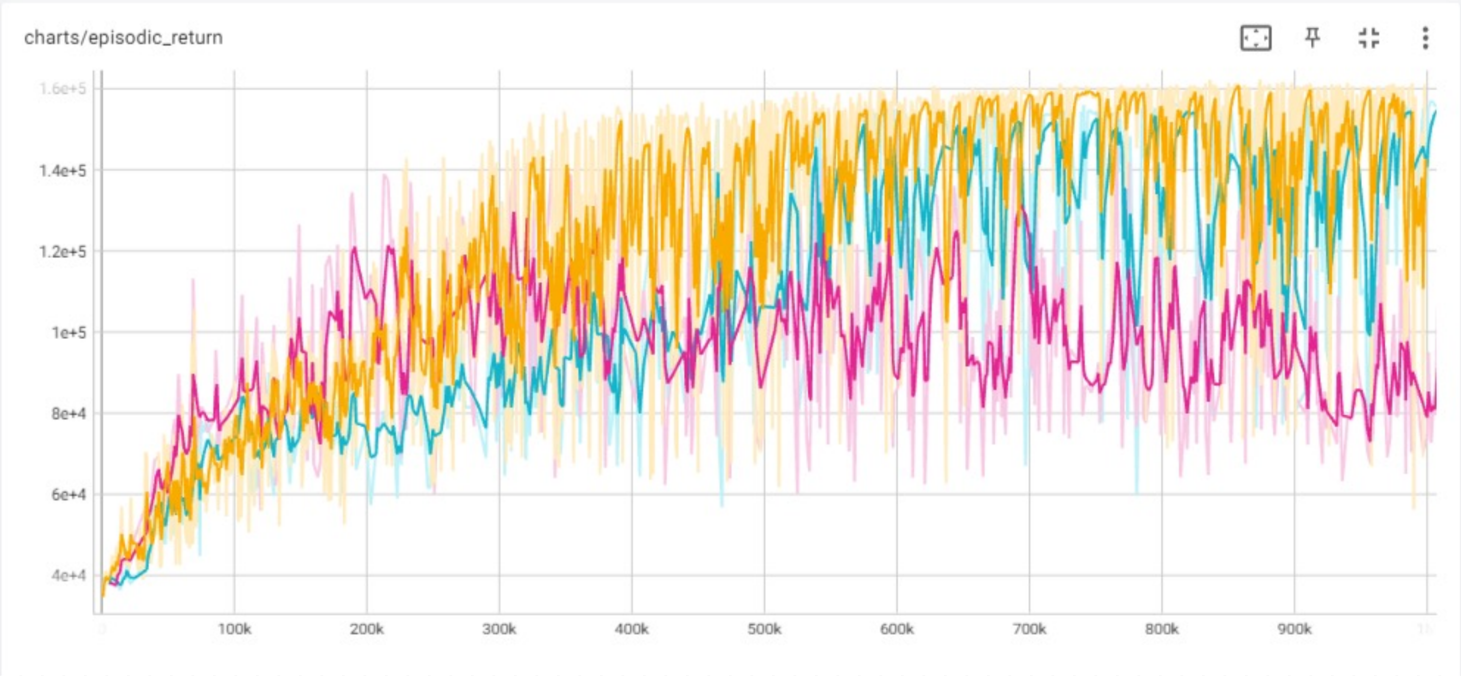
Experiments

- Device
  - CPU: AMD Ryzen 5 3600
  - GPU: GTX 1060 3GB

- Changed Hyperparameters
  - Seed
  - Entropy coefficient, Symmetric action coefficient
  - Learning rate
  - RPO alpha

Experiments (HumanoidStandup-v4)

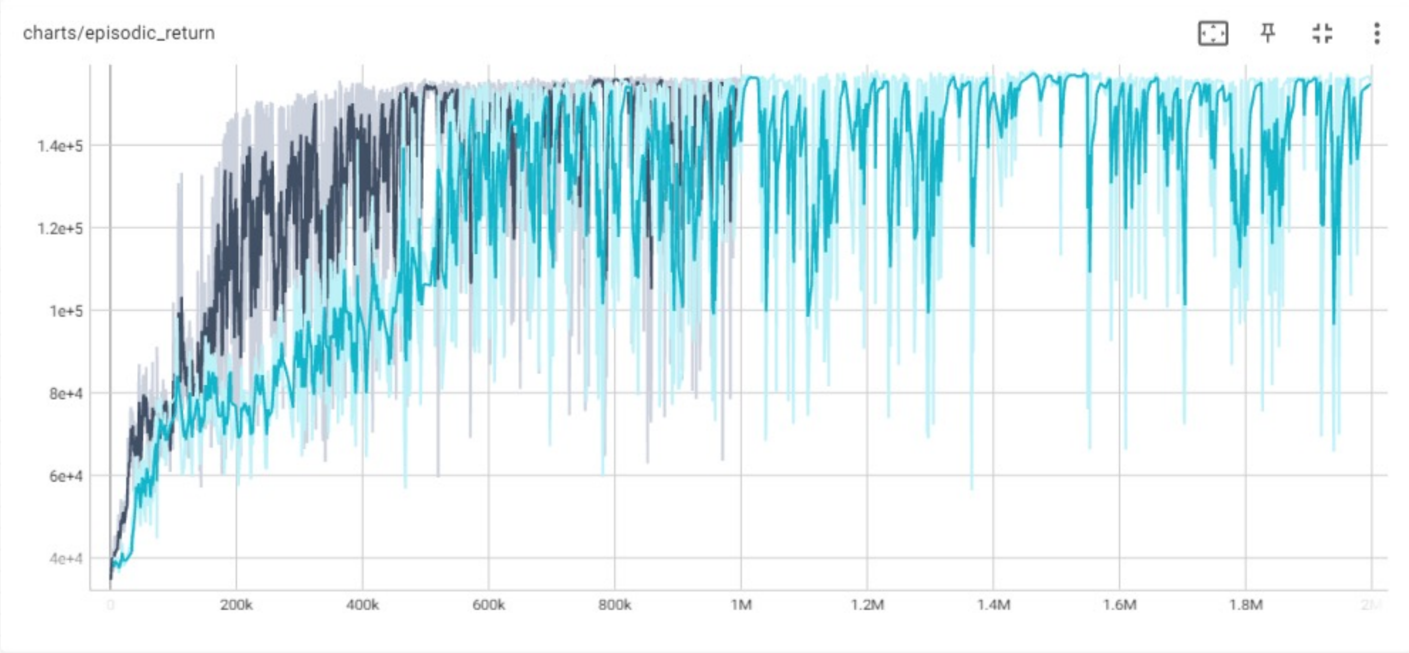
- 1. Entropy Coefficient
  - 0.01 (pink) vs. **0.0001 (yellow)** vs. 0.0 (blue)



Tensorboard

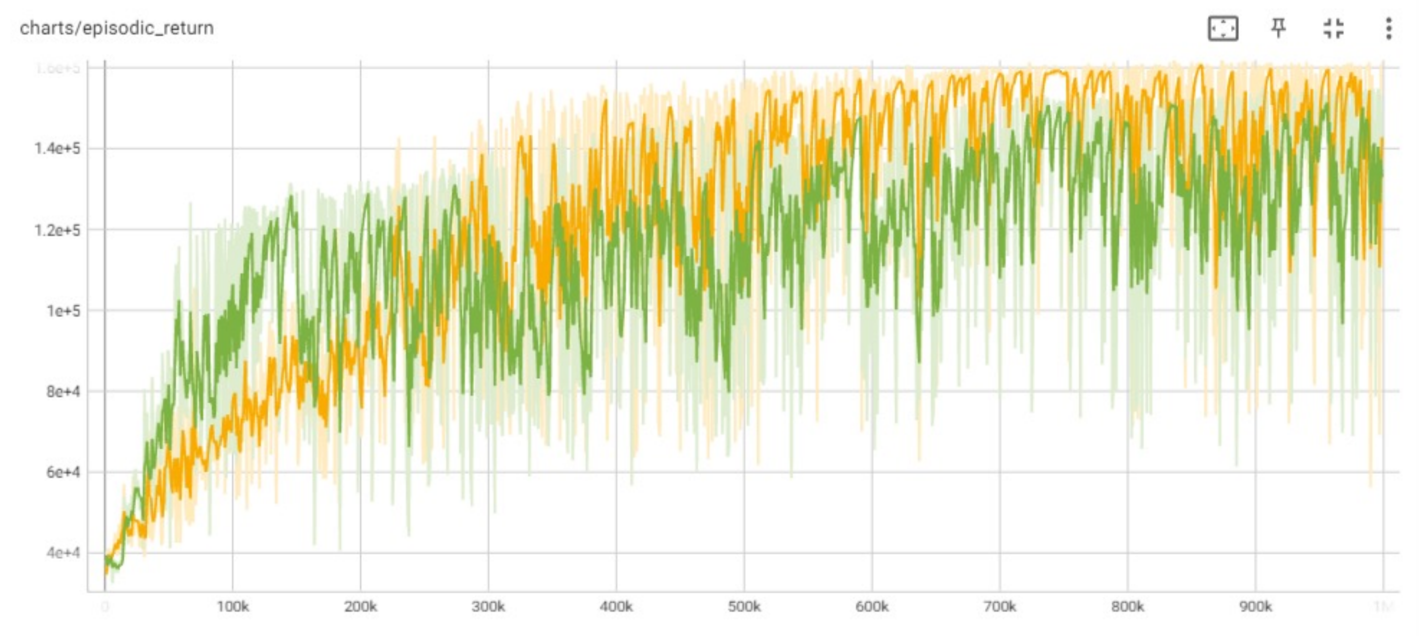
Experiments (HumanoidStandup-v4)

- 2. Learning rate
  - 3e-4 (gray) with 1M timesteps vs. 1e-4 (cyan) with 2M timesteps



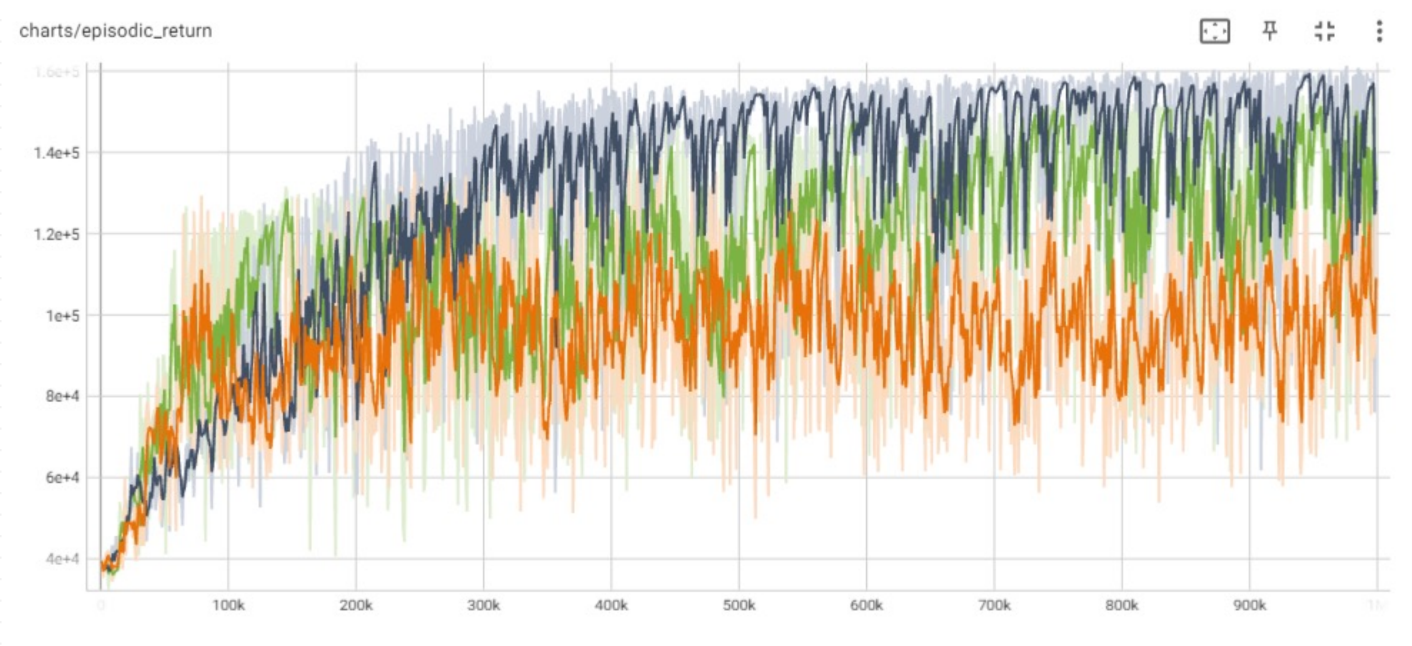
Experiments (HumanoidStandup-v4)

- 3. PPO vs. RPO
  - PPO (green) vs. **RPO with coef=0.5 (yellow)**



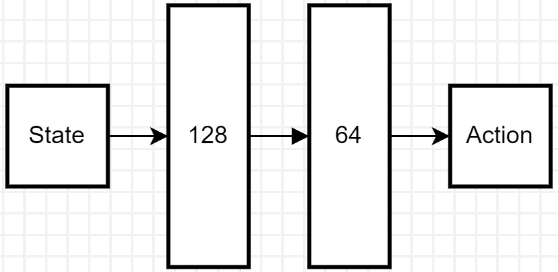
Experiments (HumanoidStandup-v4)

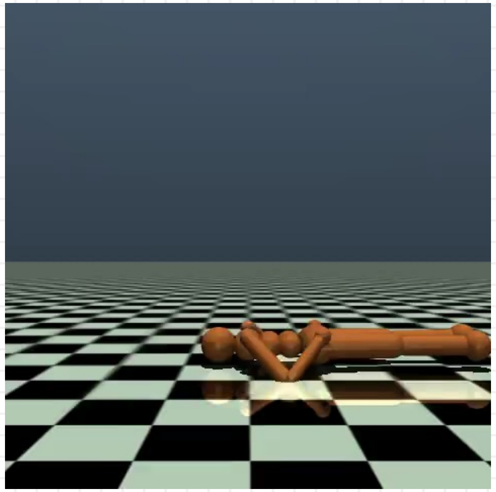
- 4. Symmetric action coefficient
  - 0.0 (green) vs. **0.01 (black)** vs. 0.05 (orange). Positive effect!

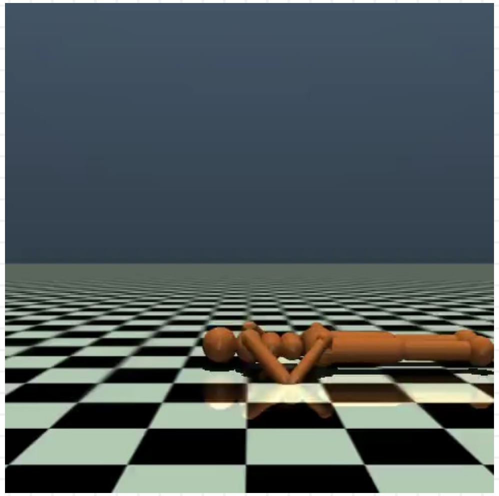


Experiments (HumanoidStandup-v4)

- Best Result
  - Training: 160,000 (highest)
  - Test: 131,000 (highest)



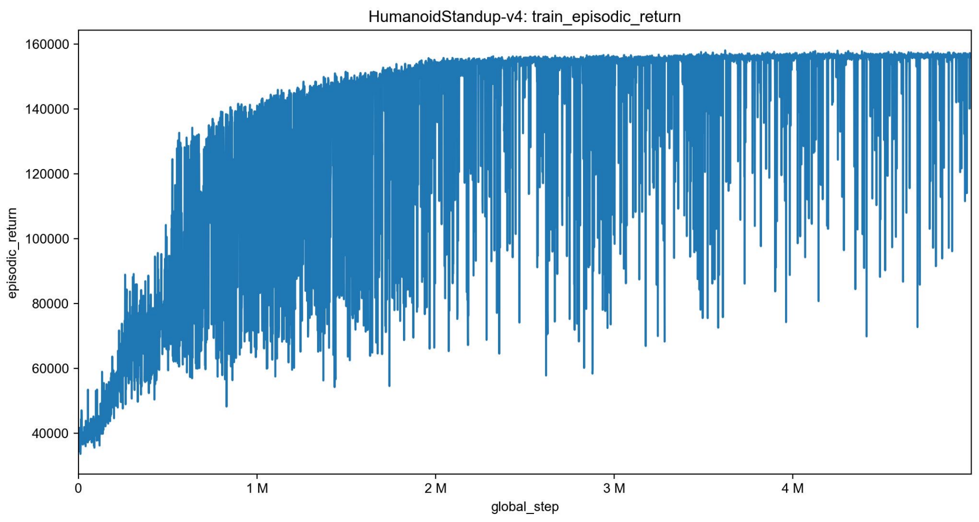


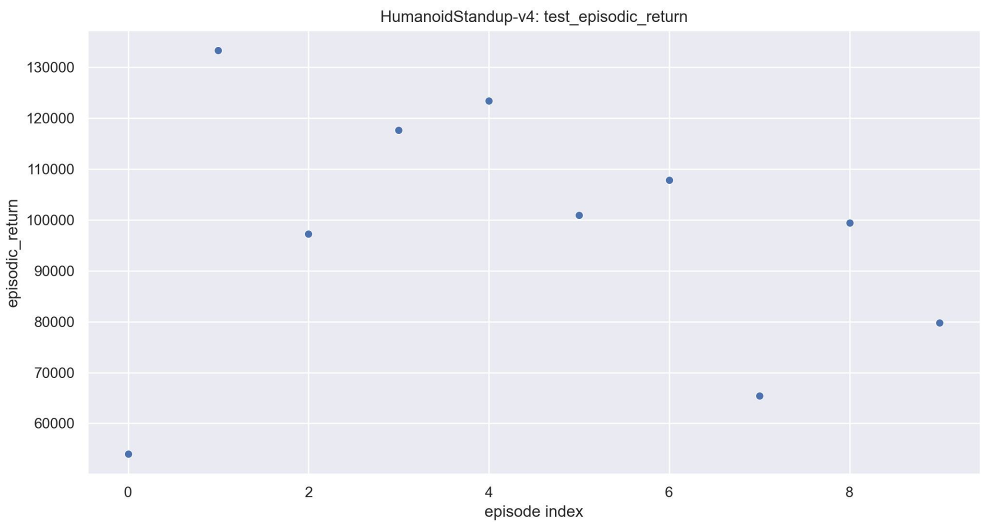


Hyperparameter	Value
Seed	0
Timesteps	5000000 (5M)
num_rollout_steps	2048
minibatch_size	64
Learning rate	2e-5
max_grad_norm	0.5
clip_coef	0.2
ent_coef	0.0001
vf_coef	0.5
gamma	0.99
gae_lambda	0.95
rpo_coef	0.5
sym_action_coef	0.02

Experiments (HumanoidStandup-v4)

- Best Result





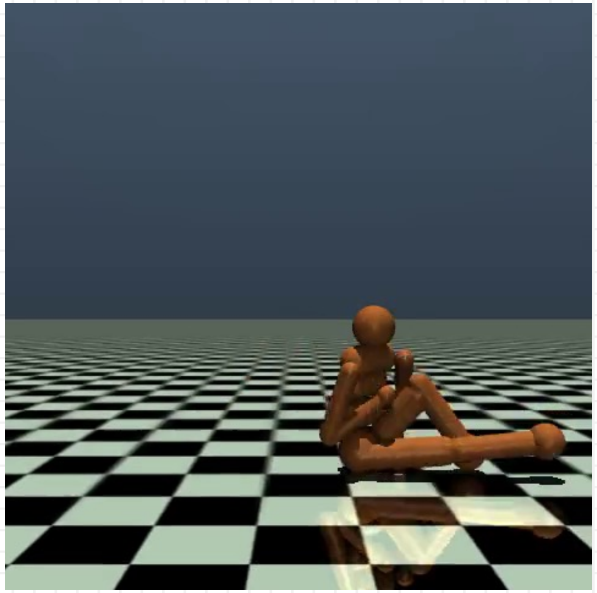
Analysis (HumanoidStandup-v4)

- RPO is better than PPO

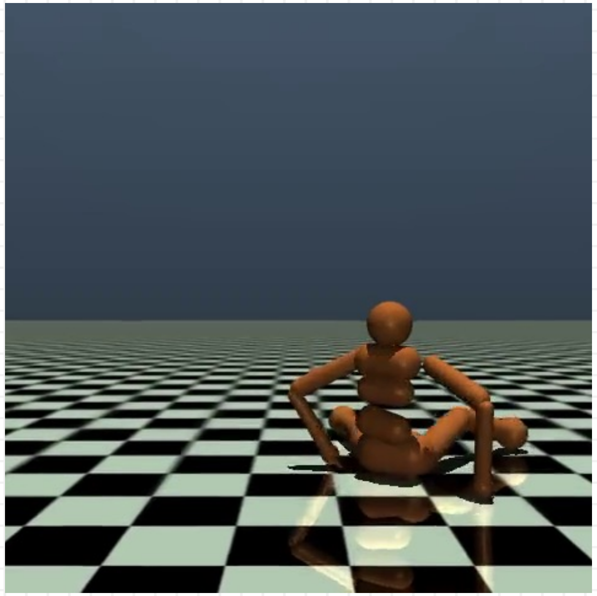
```
def get_action_and_value(self, x, action=None):
    ####################### Implement here : policy distribution ########################
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    action_mean = self.actor_mean(x)
    action_logstd = self.actor_logstd.expand_as(action_mean)
    action_std = torch.exp(action_logstd)
    probs = Normal(action_mean, action_std)
    if action is None:
        action = probs.sample()
    else: # RPO
        z = torch.FloatTensor(action_mean.shape).uniform_(-self.rpo_alpha, self.rpo_alpha).to(device)
        action_mean = action_mean + z
        probs = Normal(action_mean, action_std)
```

Analysis (HumanoidStandup-v4)

- Symmetric Action Loss guided the use of both arms and legs
- And Improve performance!

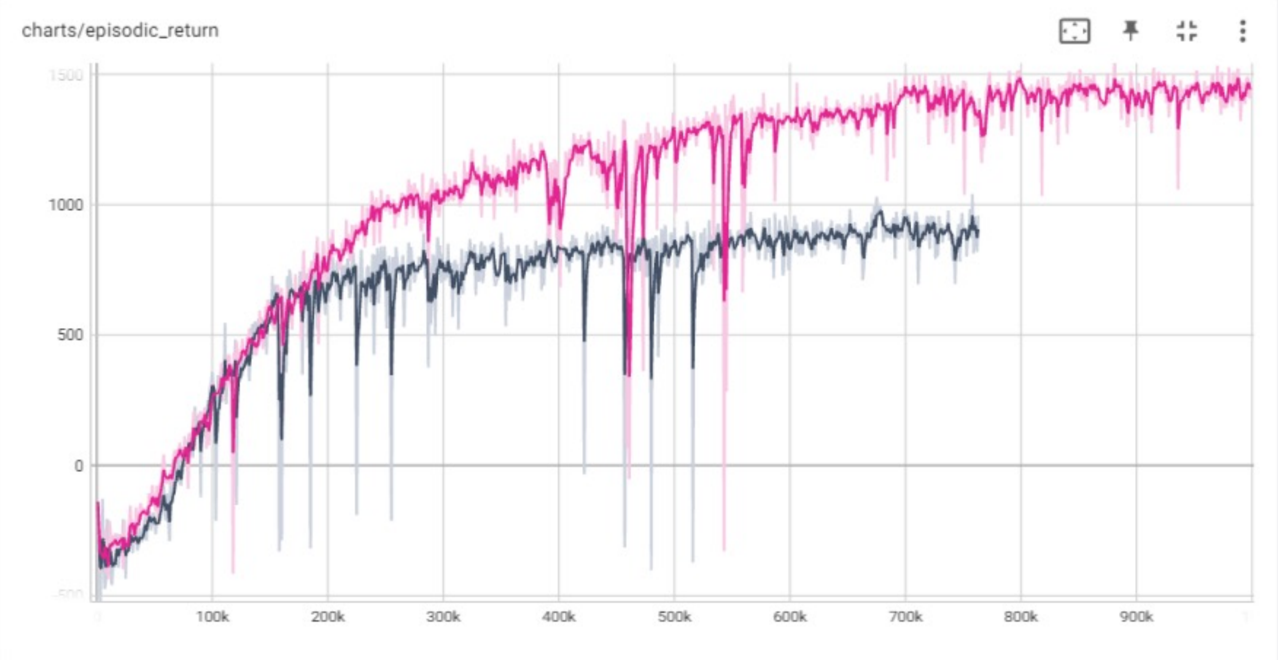


Symmetric Action Loss



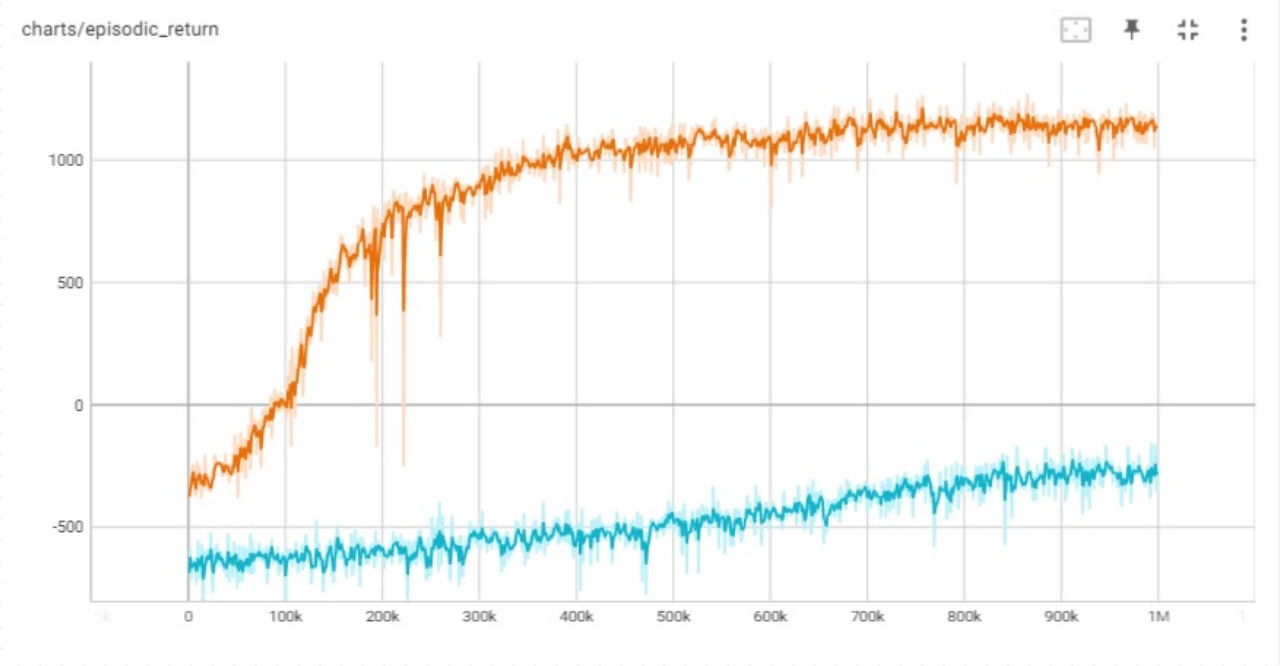
Experiments (HalfCheetah-v4)

- 1. Entropy Coefficient
  - **0.0 (pink)** vs. 0.0001 (black)



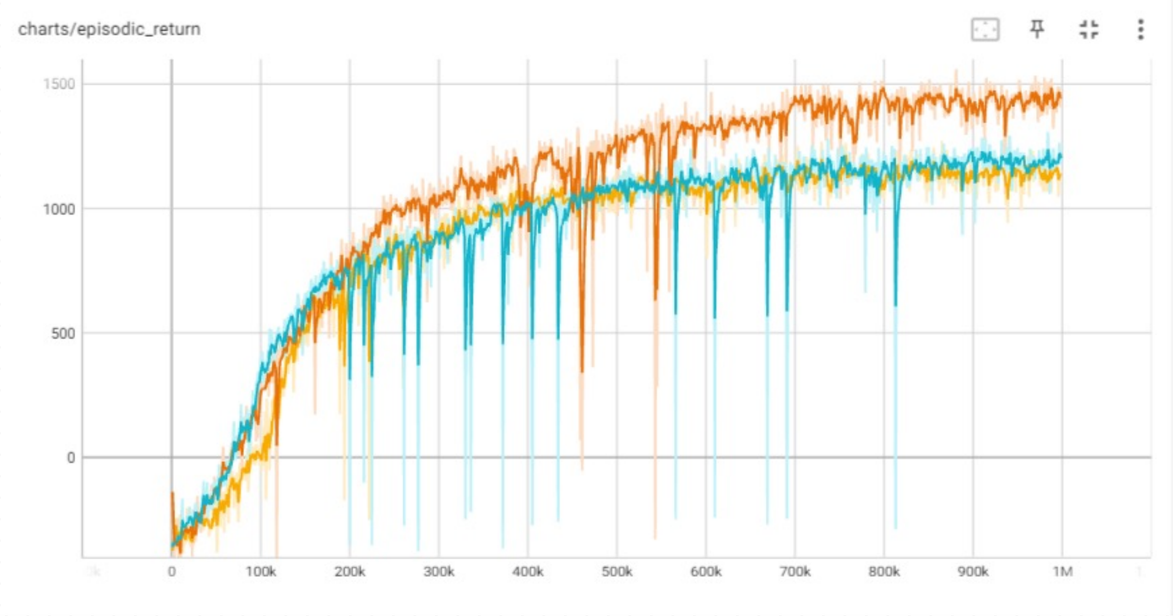
Experiments (HalfCheetah-v4)

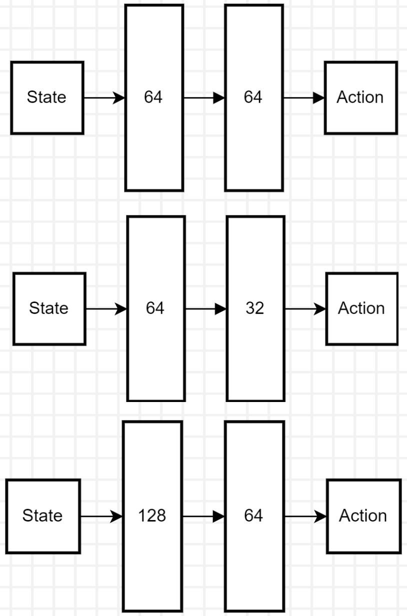
- 2. RPO vs PPO
  - **PPO (orange)** vs. RPO with coef=0.5 (cyan)



Experiments (HalfCheetah-v4)

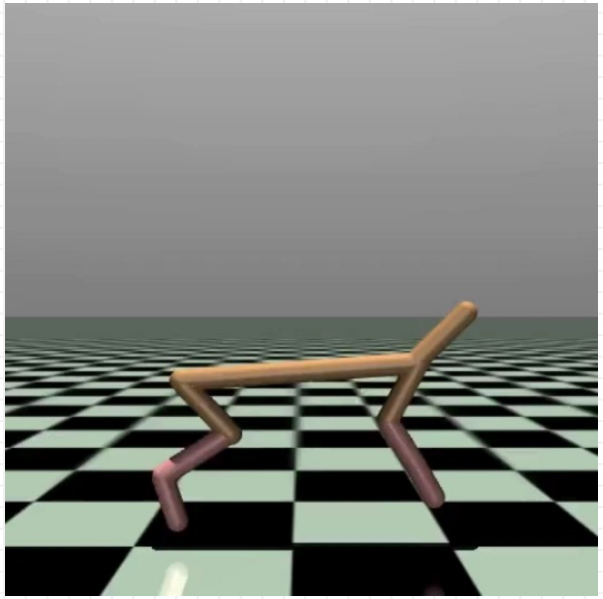
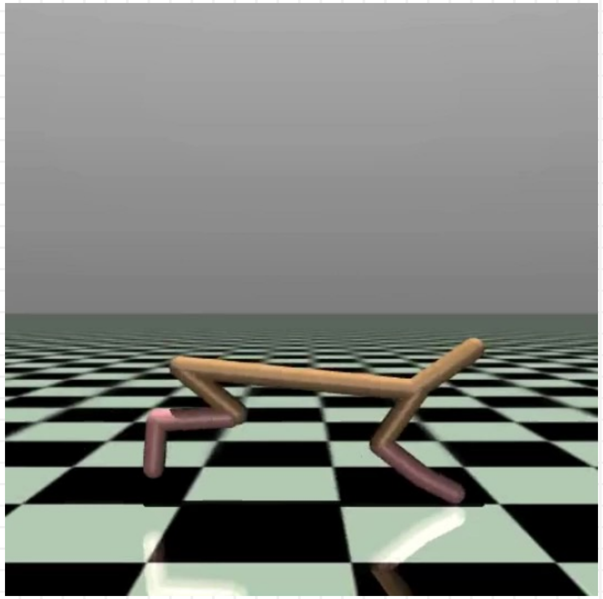
- 3. MLP
  - **64-64 (Orange)** vs. 64-32 (cyan) vs. 128-64 (yellow)





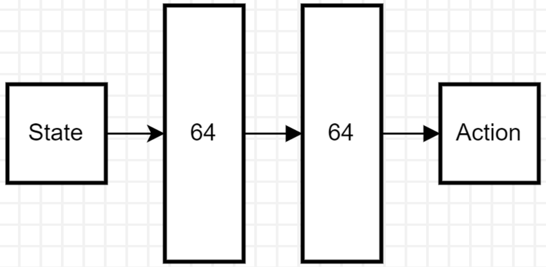
Experiments (HalfCheetah-v4)

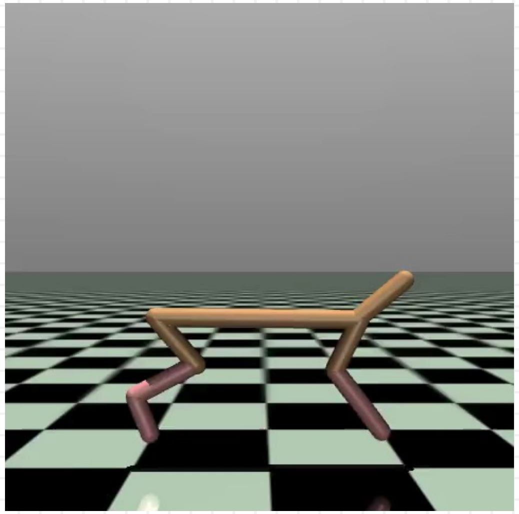
- 4. Seed
  - 0 (left) vs. 8 (right)

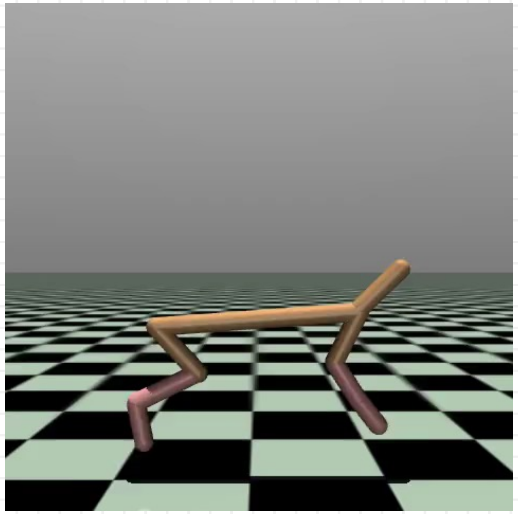



Experiments (HalfCheetah-v4)

- Best Performance
  - Training: 4,000 (highest)
  - Test: 3,250 (highest)



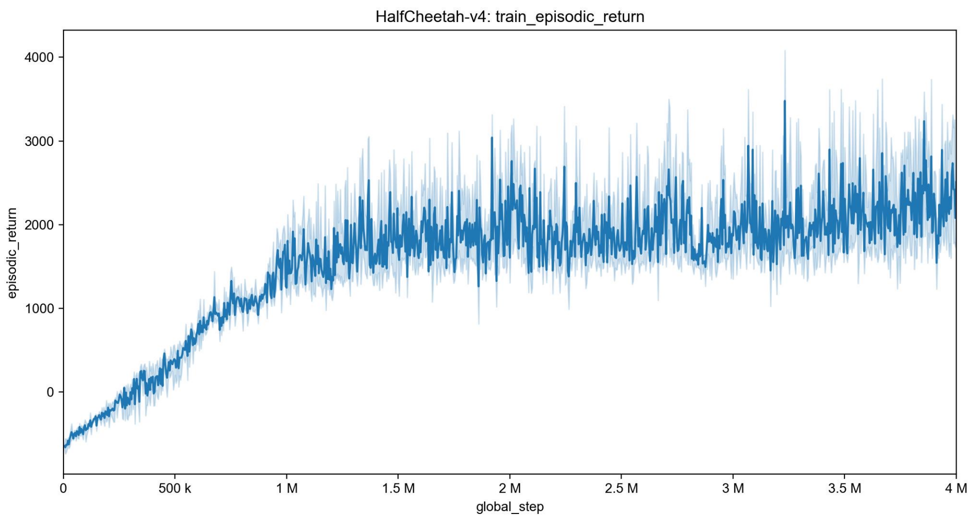


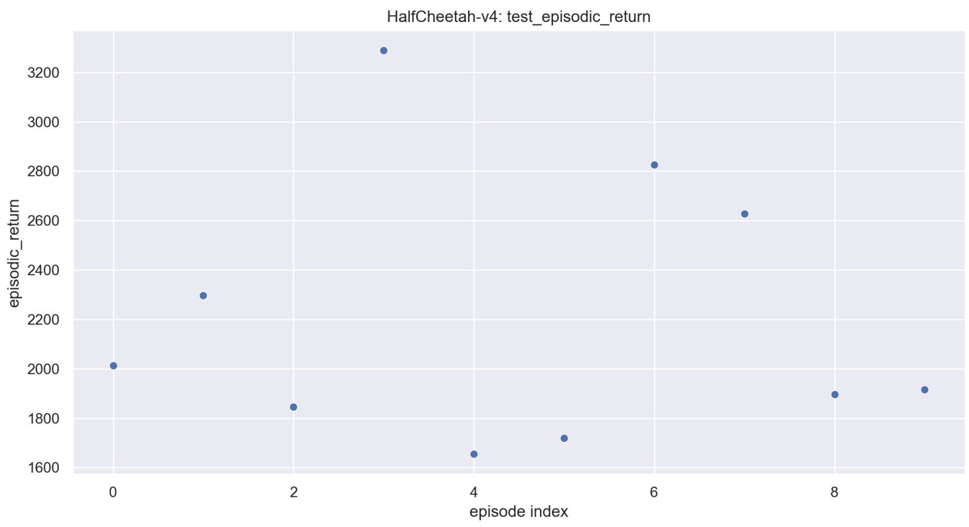


Hyperparameter	Value
Seed	8
Timesteps	1000000 (1M)
num_rollout_steps	128
minibatch_size	32
Learning rate	3e-5
max_grad_norm	2.0
clip_coef	0.2
ent_coef	0.5
vf_coef	0.5
gamma	0.99
gae_lambda	0.95

Experiments (HalfCheetah-v4)

- Best Result





Analysis (HalfCheetah-v4)

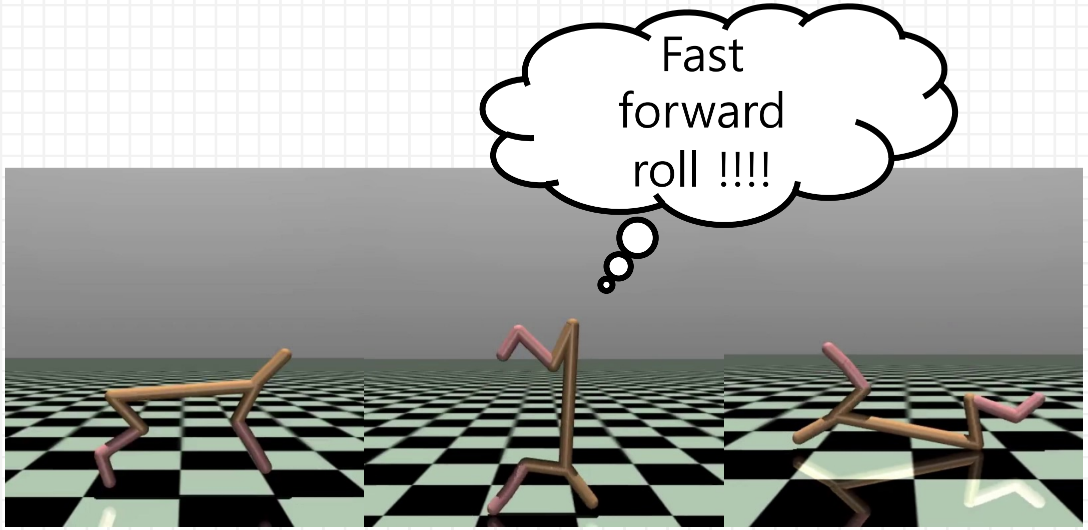
- Why does halfcheetah do a forward roll ?
  - Maybe halfcheetah get a big reward at once… → Stuck in local optima

Rewards

The reward consists of two parts:

- *forward_reward*: A reward of moving forward which is measured as `forward_reward_weight` * *(x-coordinate before action - x-coordinate after action)/dt*. *dt* is the time between actions and is dependent on the frame_skip parameter (fixed to 5), where the frametime is 0.01 - making the default *dt = 5 * 0.01 = 0.05*. This reward would be positive if the cheetah runs forward (right).
- *ctrl_cost*: A cost for penalising the cheetah if it takes actions that are too large. It is measured as `ctrl_cost_weight` * *sum(action^2)* where `ctrl_cost_weight` is a parameter set for the control and has a default value of 0.1

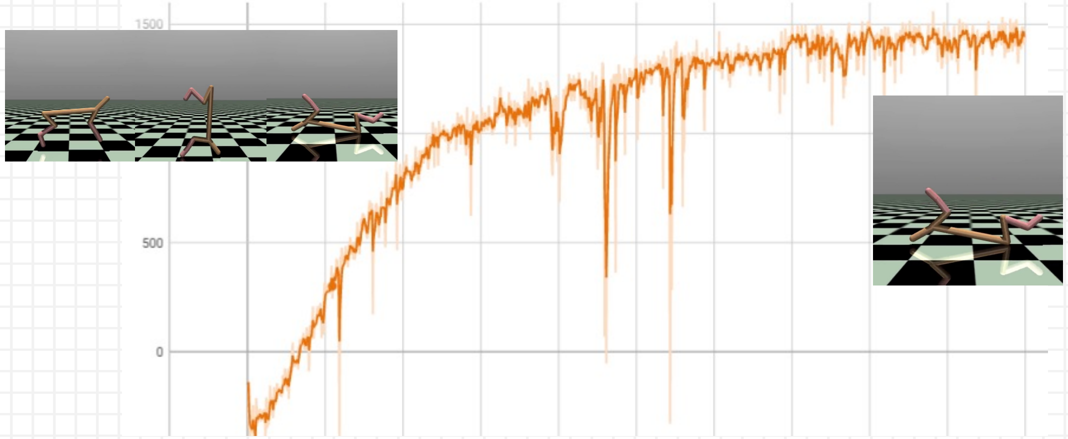
The total reward returned is ***reward*** = *forward_reward - ctrl_cost* and `info` will also contain the individual reward terms



Depending on the seed, Halfcheetah do or do not a forward roll

Analysis (HalfCheetah-v4)

- Why does halfcheetah do a forward roll ?
  - Maybe halfcheetah get a big reward at once… → Stuck in local optima
  - Forward roll: ~1500 (left) vs. Run fast: ~3000+ (right)



Conclusion

- Hyperparameters are **very important**
- Entropy loss is not useful in the two environments
- RPO worked depending on the environment
- Depending on the seed, results can vary a lot…
- It seems that other methods are needed to standup a humanoid.
- Still, PPO is an effective reinforcement learning algorithm!

Thanks ☺