HSLU Lucerne University of Applied Sciences and Arts

Advanced RL Methods

Reinforcement Learning
December 1, 2022

Trust Region Policy Optimization

In policy gradient methods, the update is calculated as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla J(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$$

The new values for θ should be near to the old values, as we use a small step size, however, the **policy** could still change significantly with a change of θ

We would like to make sure that the ***new and old policies*** are not too far apart (not that only the parameters are close)

Trust Region Policy Optimization

John Schulman JOSCHU@EECS.BERKELEY.EDU
Sergey Levine SLEVINE@EECS.BERKELEY.EDU
Philipp Moritz PCMORITZ@EECS.BERKELEY.EDU
Michael Jordan JORDAN@CS.BERKELEY.EDU
Pieter Abbeel PABBEEL@CS.BERKELEY.EDU
University of California, Berkeley, Department of Electrical Engineering and Computer Sciences

Trust Region Policy Optimization

- We would like to compare the policies, but these are distributions (probabilities for each action)
- We can use the Kullback-Leibler divergence (KL-divergence)

$$D_{KL}(P||Q) \doteq \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Trust Region Policy Optimization

We can write the cost function as a function of two policies and optimize that

$$J(\boldsymbol{\theta}, \boldsymbol{\theta}_t) = \mathbb{E}_{a \sim \pi}\left[\frac{\pi_{\boldsymbol{\theta}}(a|s)}{\pi_{\boldsymbol{\theta}_t}(a|s)} A(s,a)\right]$$

i.e. this measures how the new policy performs reative to the old policy, then we find

$$\boldsymbol{\theta}_{t+1} = \arg\max_{\boldsymbol{\theta}} J(\boldsymbol{\theta}, \boldsymbol{\theta}_t)$$

under the constraint

$$D_{KL}(\boldsymbol{\theta}_{t+1} || \boldsymbol{\theta}_t) \leq \delta$$

Trust Region Policy Optimization

- Actual implementation is mathematically a bit more complicated ☺
- The constraint problem of optimization is solved by approximate solutions (conjugate gradient)
- As it is only an approximation, the constraints might be violated anyway and a line search through different steps sizes is done to ensure the constraint

Proximal Policy Gradient (PPO)

- PPO tries to solve the same problem but uses clipping instead of a constraint on the KL-divergence
- Enforcing a constraint can also be viewed as imposing a penalty when the function gets near to it

Proximal Policy Optimization Algorithms

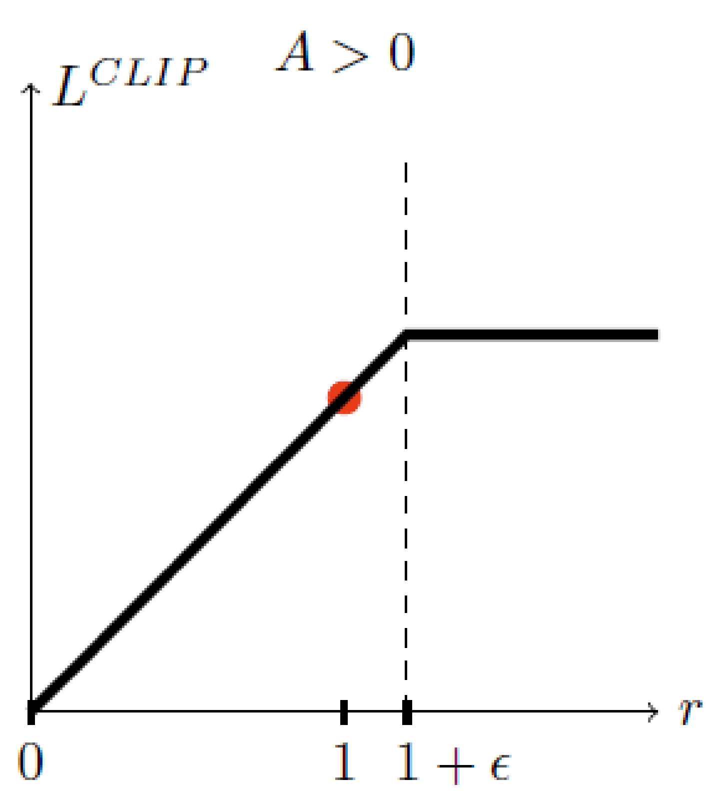
John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov
OpenAI
{joschu, filip, prafulla, alec, oleg}@openai.com

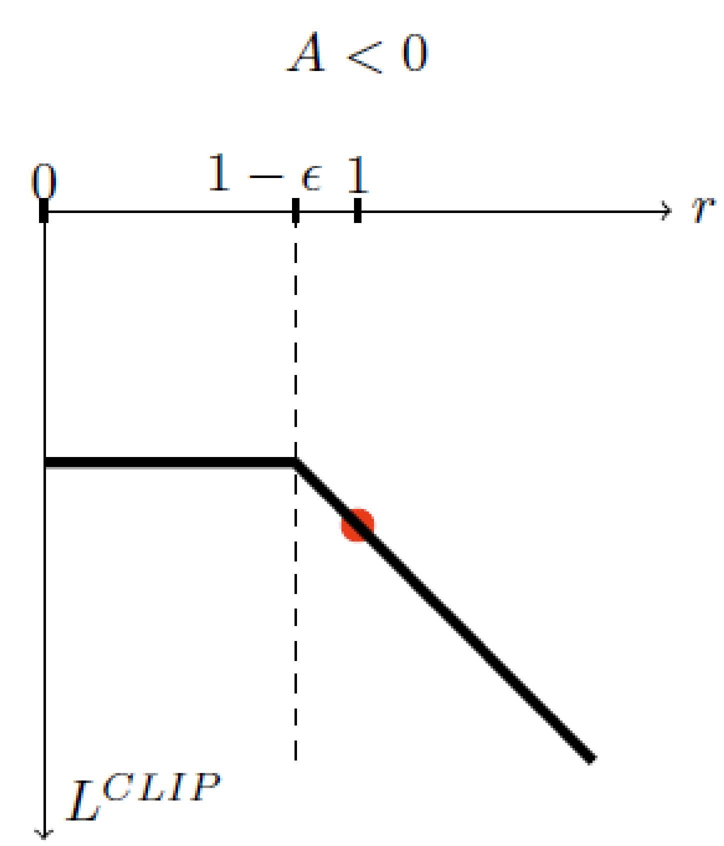
PPO: Clipped Oobjective

(CPI: Conservative Policy Iteration)

$$L^{CPI}(\boldsymbol{\theta}) = \mathbb{E}_t\left[\frac{\pi_{\boldsymbol{\theta}}(a_t, s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)} A_t\right] = \mathbb{E}_t[r_t(\boldsymbol{\theta}) A_t]$$

$$L^{CLIP}(\boldsymbol{\theta}) = \mathbb{E}_t[\min(r_t(\boldsymbol{\theta}) A_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon) A_t]$$





PPO: KL Penalty Coefficient

Calculate a penalty depending on the KL-divergence, but update the penalty parameter

$$L^{KLPEN}(\boldsymbol{\theta}) = \mathbb{E}_t\left[\frac{\pi_{\boldsymbol{\theta}}(a_t, s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}(a_t|s_t)}} A_t - \beta \text{KL}[\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot, s_t), \pi_{\boldsymbol{\theta}}(\cdot, s_t)]\right]$$

Calculate

$$d = \mathbb{E}_t[\text{KL}[\pi_{\boldsymbol{\theta}_{\text{old}}}(\cdot, s_t), \pi_{\boldsymbol{\theta}}(\cdot, s_t)]]$$

if d is small, decrease beta

if d is large, increase beta

Comparison of Algorithms

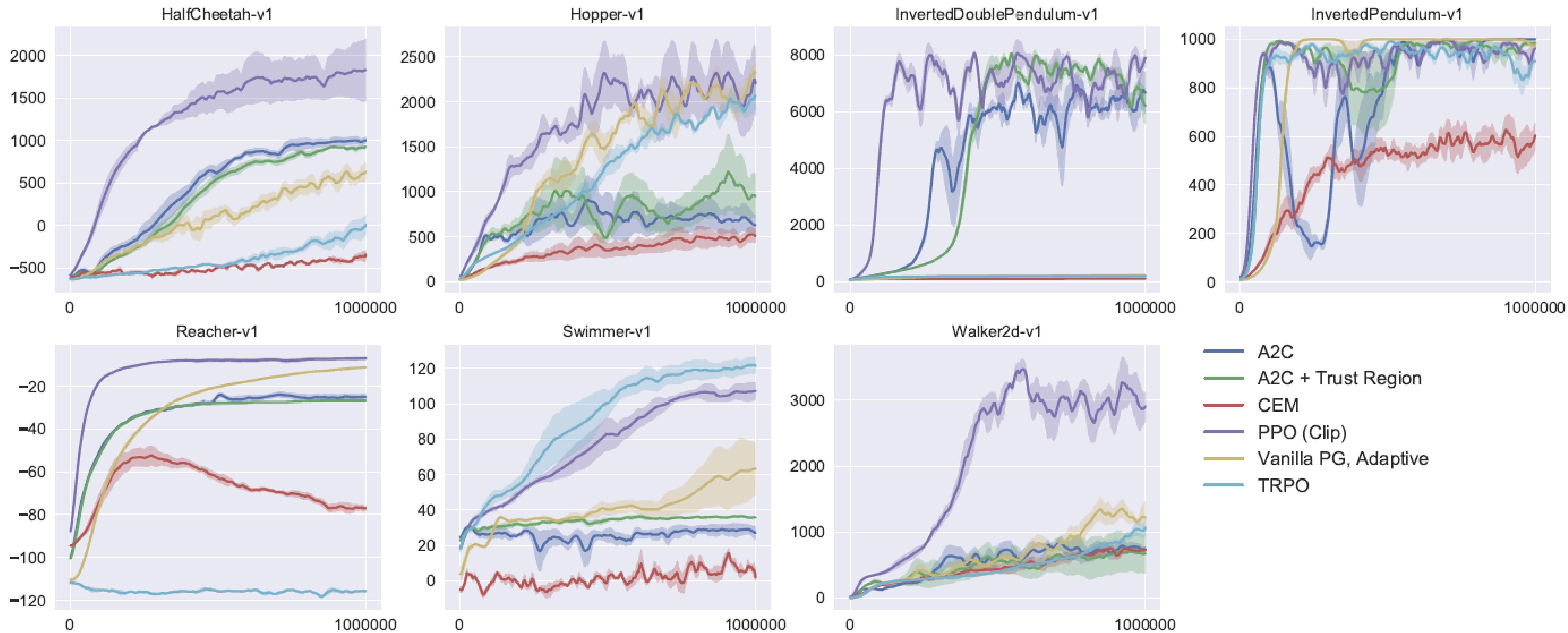


Figure 3: Comparison of several algorithms on several MuJoCo environments, training for one million timesteps.

Examples

https://openai.com/blog/openai-baselines-ppo/

Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradiant (DDPG) uses a combination of Deep Q-Learning and Policy Gradient

Q-Learning:

- Q-Learning works well for discrete action spaces but cannot be well adapted to continuing actions.
- In order to find the maximum action from a continuous action space, we would need an expensive global maximization at every step
- Furthermore, DQN learns a deterministic policy and cannot learn stochastic policies

DDPG

Policy Gradient:

- Learn stochastic policy
- it is an on-policy approach, so it cannot use a different policy for exploring than the one being optimized
- Furthermore, as the policy changes constantly the "old" trajectories are not relevant anymore, making policy gradient *sample inefficient*.

DDPG Algorithm

- Model free
- Off-policy
- Continuous and high-dimensional action space
- Actor-critic: policy network and action-value network
- Replay buffer: Store transitions and use them for training
- Target network: Use a target network, but using exponential averaging instead of copying the weights

DDPG

- Use an actor function that deterministically maps states to actions:

$$\mu(s|\boldsymbol{\theta})$$

- Use a critic that is learned using the Bellman equation (as in Q-learning):

$$Q(s,a|\mathbf{w})$$

- Update the actor by applying the chain rule to the expected return:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J &\approx \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_{\boldsymbol{\theta}} Q(s,a|\mathbf{w})|_{s=s_t,a=\mu(s_t|\boldsymbol{\theta})}\right] \\ &= \mathbb{E}_{s_t\sim\rho^\beta}\left[\nabla_a Q(s,a|\mathbf{w})|_{s=s_t,a=\mu(s_t)}\nabla_{\boldsymbol{\theta}}\mu(s|\boldsymbol{\theta})|_{s=s_t}\right]\end{aligned}$$

DDPG

- Actor and critic are updated by sampling a mini-batch from the replay buffer
- The targets weights are changed slowly:

$$\boldsymbol{\theta}' \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}'$$
$$\mathbf{w}' \leftarrow \tau\mathbf{w} + (1-\tau)\mathbf{w}'$$

- The behavior policy is constructed by adding noise to the actor function:

$$\mu'(s_t) = \mu(s_t|\boldsymbol{\theta}_t) + \mathcal{N}$$

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$, $\theta^{\mu'} \leftarrow \theta^\mu$
Initialize replay buffer R
for episode = 1, M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial observation state s_1
  for t = 1, T **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise
    Execute action a_t and observe reward r_t and observe new state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in R
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R
    Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$
    Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}$$

  end for
end for