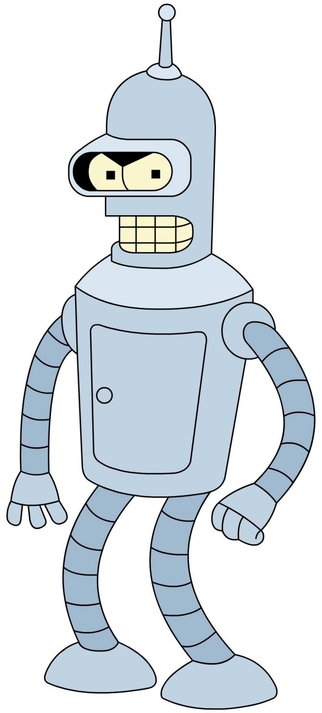
Reinforcement Learning

HSE, winter - spring 2023

Lecture 7: Continuous Control



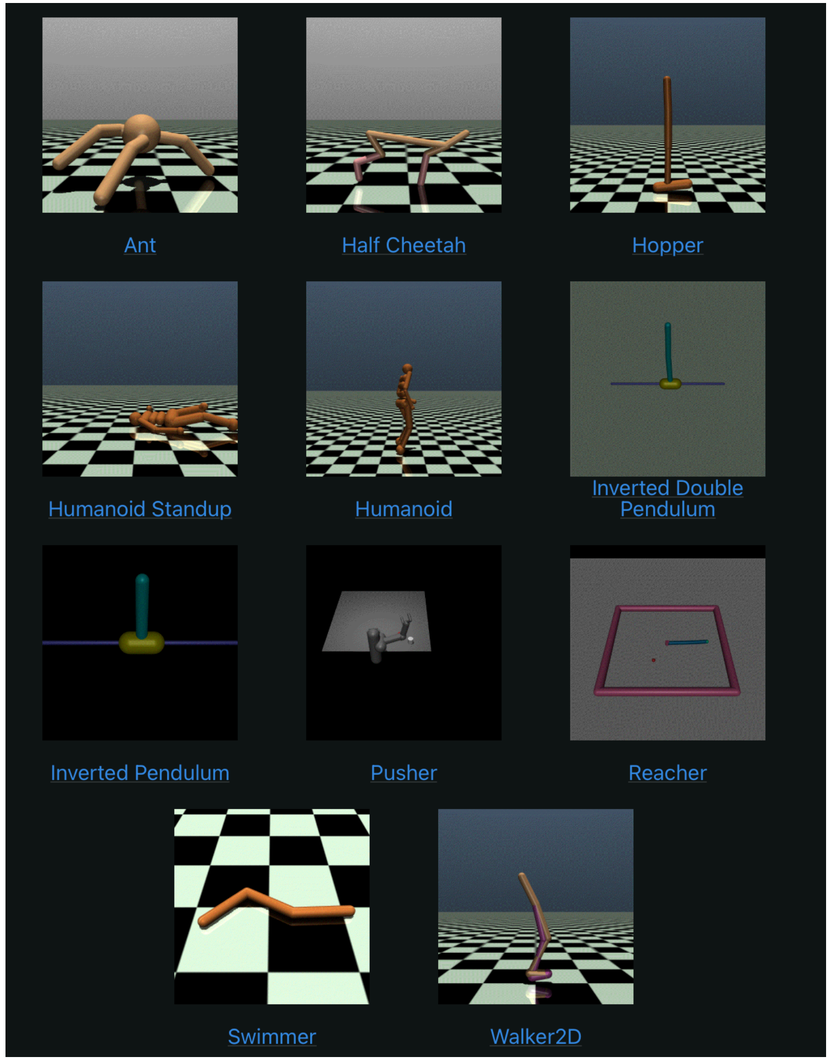
Sergei Laktionov
slaktionov@hse.ru
LinkedIn

Background

1. Reinforcement Learning Textbook (in Russian)

Continuous Control Tasks

- Action space $\mathscr{A} = [-1,1]^A$
- An obvious approach to adapting deep reinforcement learning methods such as DQN to continuous domains is to to simply discretize the action space. However such large action spaces are difficult to explore efficiently.
- Additionally, naive discretization of action spaces needlessly throws away information about the structure of the action domain, which may be essential for solving many problems.



Source

Recap: DQN vs Policy Gradient

- DQN:
  1. learn Q^* using Bellman target $r + \gamma \max_{a'} Q(s', a'; \theta)$
  2. Recover policy greedily w.r.t. $Q(s', a'; \theta)$

- Policy Gradient (REINFORCE, A2C, PPO):
  1. Learn policy directly calculating the gradient using log-derivative trick of $J(\pi_\theta)$ w.r.t. policy parameters θ (actor)
  2. Learn critic to estimate the quality of the current policy (critic)

Recap: DQN vs Policy Gradient

- DQN:
  1. learn Q^* using Bellman target $r + \gamma \max_{a'} Q(s', a'; \theta)$
  2. Recover policy greedily w.r.t. $Q(s', a'; \theta)$

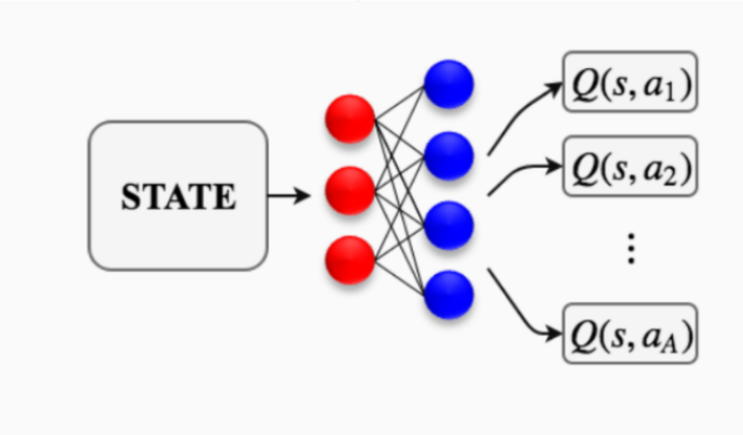
  - Only applicable to the discrete action space due to $argmax_a Q(s, a)$
  - Artificial exploration with ε-greedy policies
  - Off-policy algorithm, high sample efficiency thanks to Replay Buffer using
  - 1-step target, target network

- Policy Gradient (REINFORCE, A2C, PPO):
  1. Learn policy directly calculating the gradient using log-derivative trick of $J(\pi_\theta)$ w.r.t. policy parameters θ (actor)
  2. Learn critic to estimate the quality of the current policy (critic)

  - Applicable to both discrete and continuous action spaces
  - Natural exploration with stochastic policies
  - On-policy, lower sample efficiency, Replay Buffer can not be used
  - N-step target, target network can not be used

Policy Improvement as Optimisation

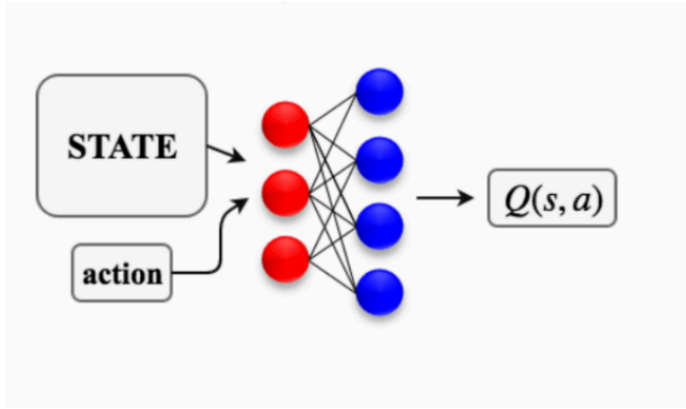
Source



$$\pi(s) = argmax_a Q(s, a)$$

DQN

Source



$\mu_\theta(s)$ is a deterministic parametrised policy

$$\mathbb{E}_s Q(s, \mu_\theta(s)) \rightarrow \max_\theta$$

Deep Deterministic Policy Gradient (DDPG)

Exploration

A major challenge of learning in continuous action spaces is exploration. An advantage of off- policies algorithms such as DDPG is that we can treat the problem of exploration independently from the learning algorithm. An exploration policy can be constructed by adding noise sampled from a noise process ε to our actor policy:

$a = \mu_\theta(s) + \varepsilon$, where:

1. Gaussian noise: $\varepsilon \sim \mathcal{N}(0,\sigma^2)$

2. Ornstein-Uhlenbeck process: $\varepsilon = \alpha\varepsilon + \nu, \nu \sim \mathcal{N}(0,\sigma^2)$

Deep Deterministic Policy Gradient

Original paper

Pseudocode

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

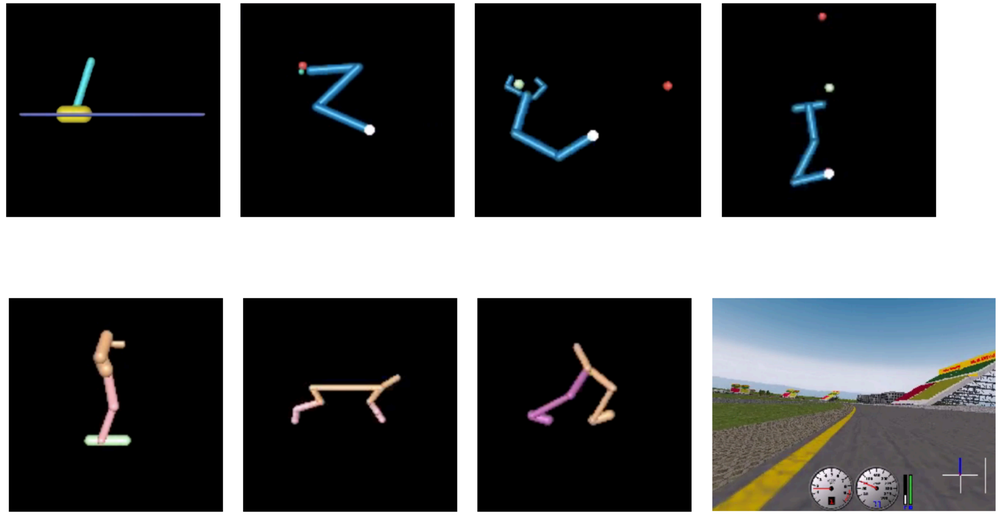
$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

    end for
  end if
until convergence

Source



- All improvements from DQN can be used here
- GAN-like interaction between actor and critic -> instability
- Overestimation bias problem

Twin Delayed DDPG

Clipped Double-Q Learning. TD3 learns *two* Q-functions instead of one (hence “twin”), and uses the smaller of the two Q-values to form the targets in the Bellman error loss functions.

“Delayed” Policy Updates. TD3 updates the policy (and target networks) less frequently than the Q-function. The paper recommends one policy update for every two Q-function updates.

Target Policy Smoothing. TD3 adds noise to the target action, to make it harder for the policy to exploit Q-function errors by smoothing out Q along changes in action.

Twin Delayed DDPG

Original paper

Algorithm 1 Twin Delayed DDPG

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for j in range(however many updates) **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute target actions

$$a'(s') = \text{clip}\left(\mu_{\theta_{\text{targ}}}(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right), \qquad \epsilon \sim \mathcal{N}(0, \sigma)$$

      Compute targets

$$y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$$

      Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2 \qquad \text{for } i = 1, 2$$

      if j mod `policy_delay` $= 0$ **then**
        Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_{\phi_1}(s, \mu_\theta(s))$$

        Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \qquad \text{for } i = 1, 2$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1 - \rho)\theta$$

      end if
    end for
  end if
until convergence

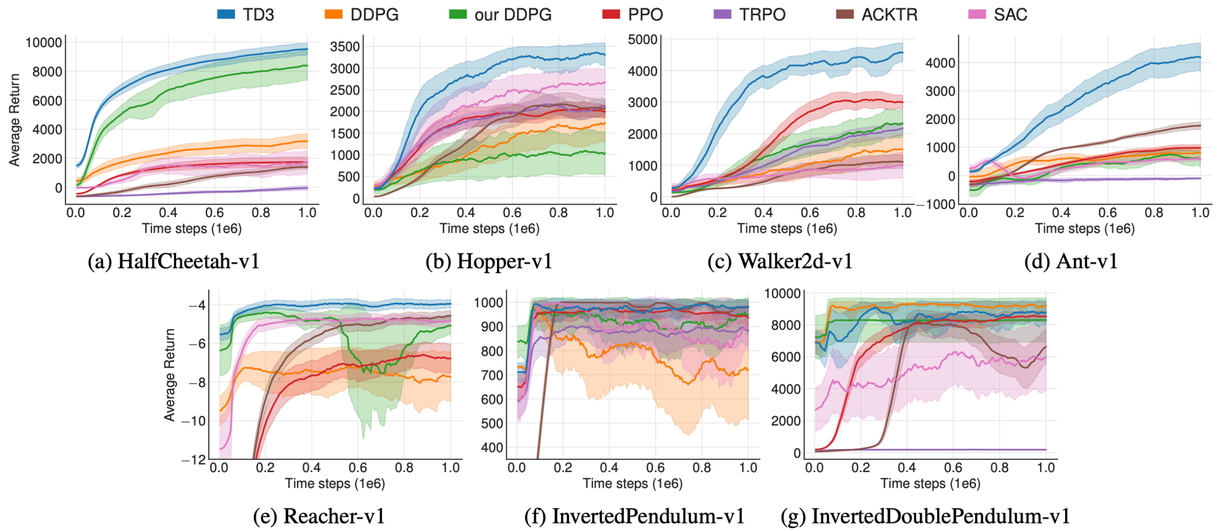


Figure 5. Learning curves for the OpenAI gym continuous control tasks. The shaded region represents half a standard deviation of the average evaluation over 10 trials. Curves are smoothed uniformly for visual clarity.

Not covered yet

1. Maximum Entropy RL
2. Soft Actor-Critic
3. Truncated Quantile Critics

Thank you for your attention!