

13강.Actor-Critic: DDPG,SAC

Contents

- Deep Deterministic Policy Gradient(DDPG)
- Code Ex.1
- Soft Actor Critic(SAC)
- Code Ex.2

DDPG

- Designed specifically for continuous action spaces
- Instead of a discrete action set: $a \in \{a_0, a_1, a_2, a_3, \dots\}$, the number of valid actions is infinite
- Example of a continuous action $a = [0.2, 1.5, 0.1]$

DDPG

- If $Q^*(s, a)$ is the optimal Q-value function, $Q^*(s, a)$ then the optimal actions are given by:

$$a^*(s) = \arg \max_a Q^*(s, a)$$

- But how do we find the action with the highest value among infinite possible actions?

DDPG

- In order to solve control tasks,
DDPG assume that $Q(s, a(s))$, is differentiable w.r.t $a(s)$
 - $Q(s, a(s))$ change very smoothly as the values of actions change
 - gradient ascent to get max. Q
- Similar actions should have similar values:
 - E.g., action $[1, -1]$, action $[1.0001, -1]$ will have very similar Q values.
 - E.g., In practice, setting steering boat wheel at 15 is very similar to the setting 15.01

DDPG

- DDPG updates θ of $\pi(s|\theta)$ via gradient ascent in the direction of maximum $Q(s, \pi_\theta(s))$ increase, so that it produces the highest Q-value action

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta Q(s, \pi_\theta(s))$$

$Q(s, \pi_\theta(s))$ is differentiable w.r.t θ

- Recall PG optimize policy $\pi(s|\theta)$ via gradient ascent so that it produces the most valuable actions:

$$\theta_{t+1} = \theta_t + \alpha \gamma^t G_t \nabla \ln \pi(A_t | S_t, \theta_t) \quad \text{REINFORCE}$$

$$\theta_{t+1} = \theta_t + \alpha \gamma^t \widehat{A} v_t \nabla \ln \pi(A_t | S_t, \theta_t) \quad \text{A2C}$$

DDPG

- Updates the policy through the Q-network:
Compute Q values by policy's chosen action

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} Q(s, \pi_{\theta}(s))$$

Deterministic policy
(argmax policy)

Find derivative w.r.t the neural net. parameters of the policy
(without involving the Q network)

Perform stochastic gradient ascent to improve the policy

DDPG

- DDPG belongs to Actor-Critic method:
 - Learn a policy and a value function at the same time
- While we update the policy to make better decision, we also train the Q-network to estimate the action values better

$$L(\theta) = (Q_{\theta}(s, a) - \hat{y})$$

,where $\hat{y} = r + \gamma Q(s', a')$

DDPG

- Problem: If we push our policy to take the actions we think are best (deterministic), how do we explore?
- DDPG Adds Gaussian noise:

$$\mu'(s) = \mu(s) + \varepsilon \quad \varepsilon \sim N(0, \sigma)$$

$$\text{e.g., } \mu(s) = [0.5, 2] \quad \varepsilon = [0.02, -0.3]$$

DDPG

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 gradient:

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

 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

Behavior policy is stochastic due to the added noise for exploration

Target policy is deterministic
($a_{t+1} = \text{argmax policy}$)

$$\nabla_{\theta} Q(s, \mu_{\theta}(s)) = \nabla_a Q(s, a) \nabla_{\theta} \mu(s)$$

Recall:

$$h(x) = (g \cdot f)(x)$$

$$h'(x) = g'(f(x))f'(x)$$

Let's play with the code

- Anaconda Install
- Virtual Env. Create: `conda create -n [이름] python=3.8`
- Pytorch Install: go <https://pytorch.kr/get-started/locally/> ,and copy pip the command line to install
- Gym Package Install:
 - `conda install -c conda-forge box2d-py`
 - `pip install gym==0.23.1`
 - `Pip install numpy==1.23.1`
 - `Pip install matplotlib`

Let's play with the code

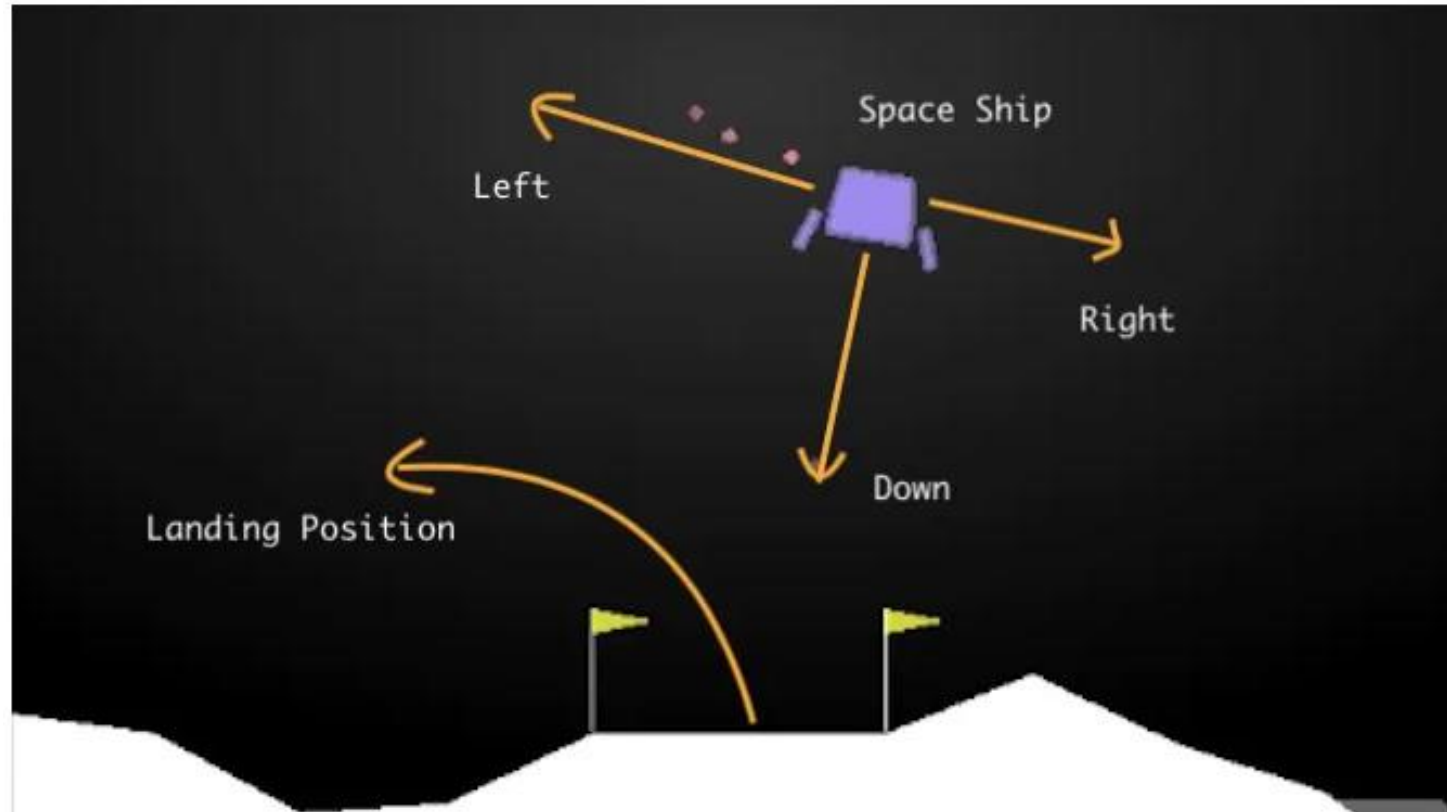
- In /DDPG
 - main_ddpg.py # main loop interacting the env., agent learning
 - ddpq_torch.py # defines agent obj.
 - networks.py # defines actor and critic network
 - buffer.py # defines replay buffer
 - noise.py # generates OU action noise
 - utils.py # plotting function
 - /temp # store parameters
 - /plots # store the score that agent get while training

Let's play with the code



Let's play with the code

- LunarLanderContinuous-v2



Let's play with the code

- $a \sim [-1,1] \times [-1,1]$
- a is `np.array([main, lateral])`
- the main engine will be turned off completely if `main < 0` and the throttle scales affinely from 50% to 100% for `0 ≤ main ≤ 1`
- `-0.5 < lateral < 0.5`, no lateral boosters fire
- `lateral < -0.5`, the left booster fire
- `lateral > 0.5`, the right booster fire

Action Space	<code>Box(-1, +1, (2,), dtype=np.float32)</code>
Observation Space	<code>Box([-1.5 -1.5 -5. -5. -3.1415927 -5. -0. -0.], [1.5 1.5 5. 5. 3.1415927 5. 1. 1.], (8,), float32)</code>
import	<code>gymnasium.make("LunarLander-v2")</code>

11강. Soft Actor-Critic(SAC)

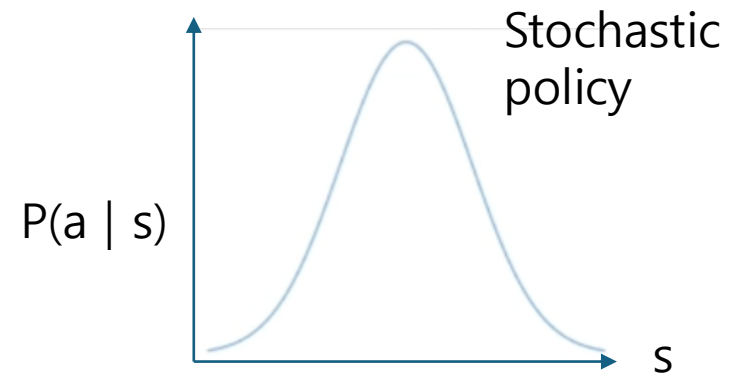
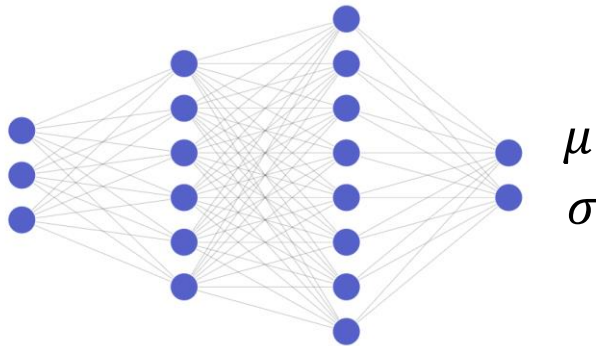
Problem with Deterministic Policy

- Deterministic policy:
 - always takes same action for a specific state s . It means that we need extra "tricks" to explore
 - (e.g., ϵ -greedy policy, target policy smoothing, adding random noise)
- Stochastic policy:
 - doesn't need to apply any of these tricks
 - it doesn't always choose the same action in the same state
 - Assigns a probability to each action and extract samples

Stochastic policies

- If we make the policy follow a normal distribution, then we just need to compute the mean and standard deviation

$$N(\mu, \sigma)$$



Entropy

- We want to maintain the agent's exploration but we do not have mechanisms such as ε -greedy policies
- Now the neural net. is our policy. How to incorporate an exploration mechanism into our neural network?
- We will incentivize the agent to keep the entropy of its policy as high as possible

$$H(X) = - \sum_{x \in X} p(x) \cdot \ln p(x)$$

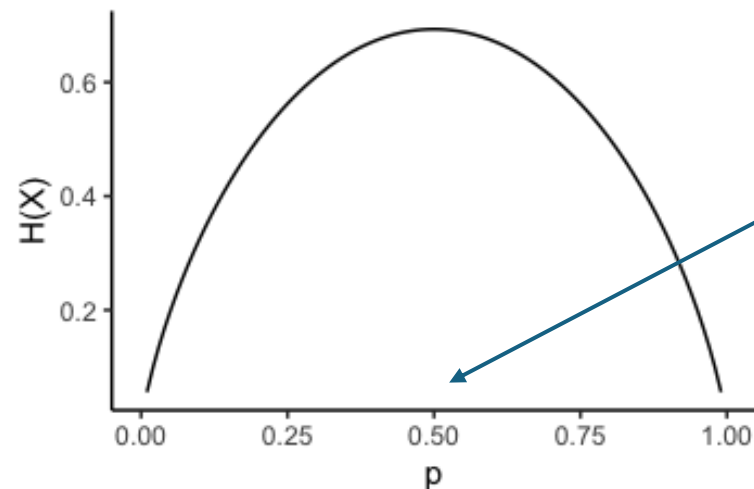
Entropy

- What is Entropy? the level of uncertainty of a random variable

e.g.,

if $p(X = x_1) = 1, p(X = x_2) = 0, H(X) = -[1 \cdot \ln(1) + 0 \cdot \ln(0)] = 0$

if $p(X = x_1) = 0.5, p(X = x_2) = 0.5, H(X) = -[0.5 \cdot \ln(0.5) + 0.5 \cdot \ln(0.5)] \approx 0.6931$



- Max. point where our confidence in our predictions will go down.
- Max. point where it surprises us

Entropy

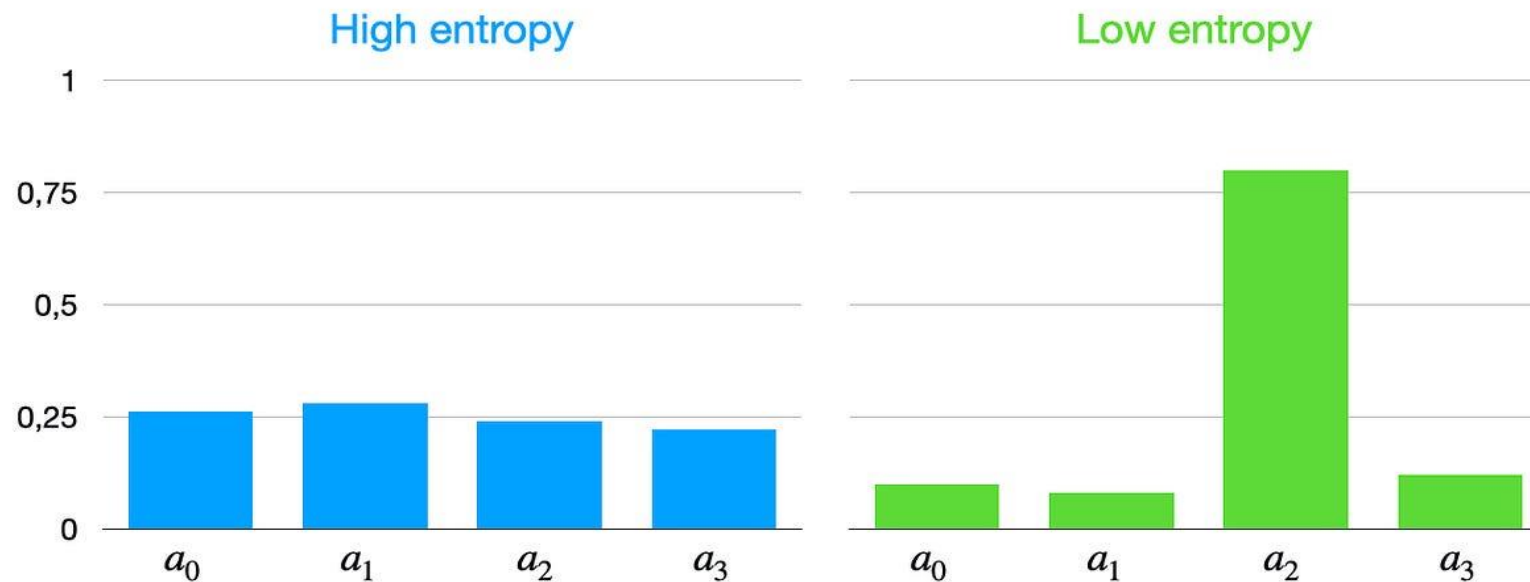
- What is the entropy of a policy?
- Uncertainty in the action to be selected in a state:

$$H_{\pi}(A_t) = - \sum_{a \in A_t} \pi(a|St) \cdot \ln \pi(a|St)$$

- Random variable(A_t) is the action the policy will choose in a state.
- The entropy is computed by multiplying the probability of choosing each action by its logarithm and adding up the results

Entropy

- What is the entropy of a policy?
- Imagine that we have 4 actions available



Entropy-regularized RL

- Maximize the cumulative sum of rewards while keeping the entropy of the policy as high as possible

- The entropy measures "how random" is our policy is

$$H(\pi) = E[-\ln \pi]$$

- Soft MDPs :

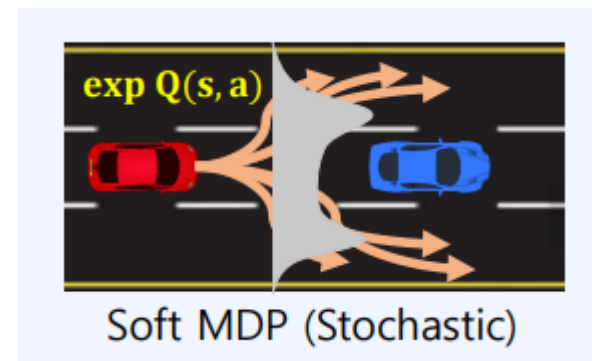
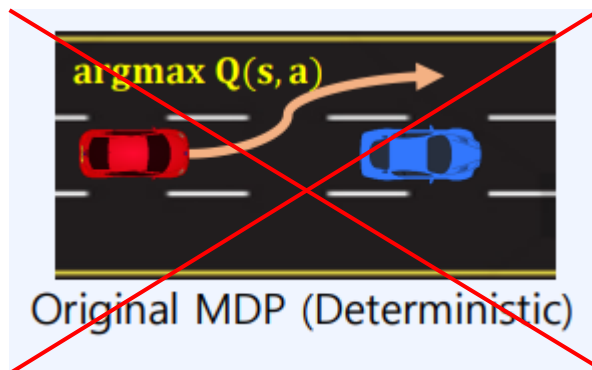
$$\max_{\pi} E\left[\sum_{t=0}^{\infty} \gamma^t r(st, at)\right] + \frac{\alpha H(\pi)}$$

Entropy Regularization:
Entropy of policy distribution

Softmax Policy

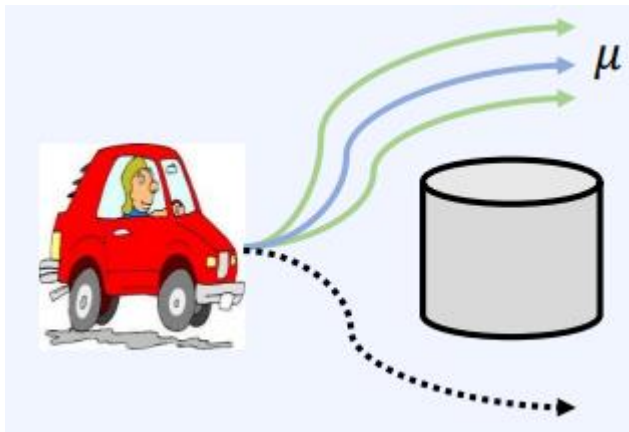
- A stochastic policy can provide multiple action choices
- **Softmax** dist. is the optimal solution of soft MDP (Max. Ent RL)
- Probability is exponentially proportional to the state action value $Q(s, a)$

$$\pi(a|s) = \frac{\exp Q(s, a)}{\sum_{a'} \exp Q(s, a')}$$

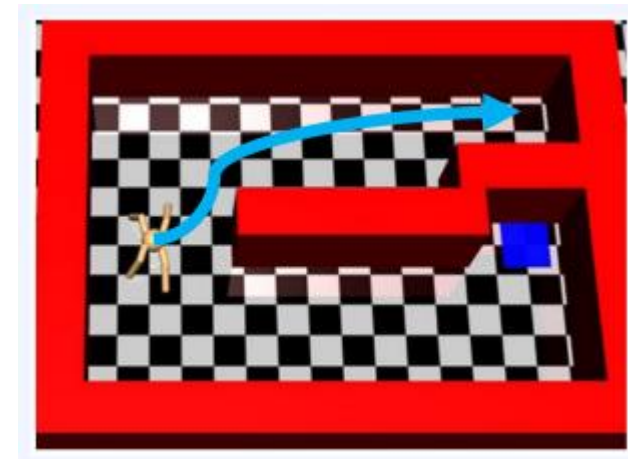


Softmax Policy

- Robustness
 - By finding multiple near optimal actions
- Avoid mode collapsing

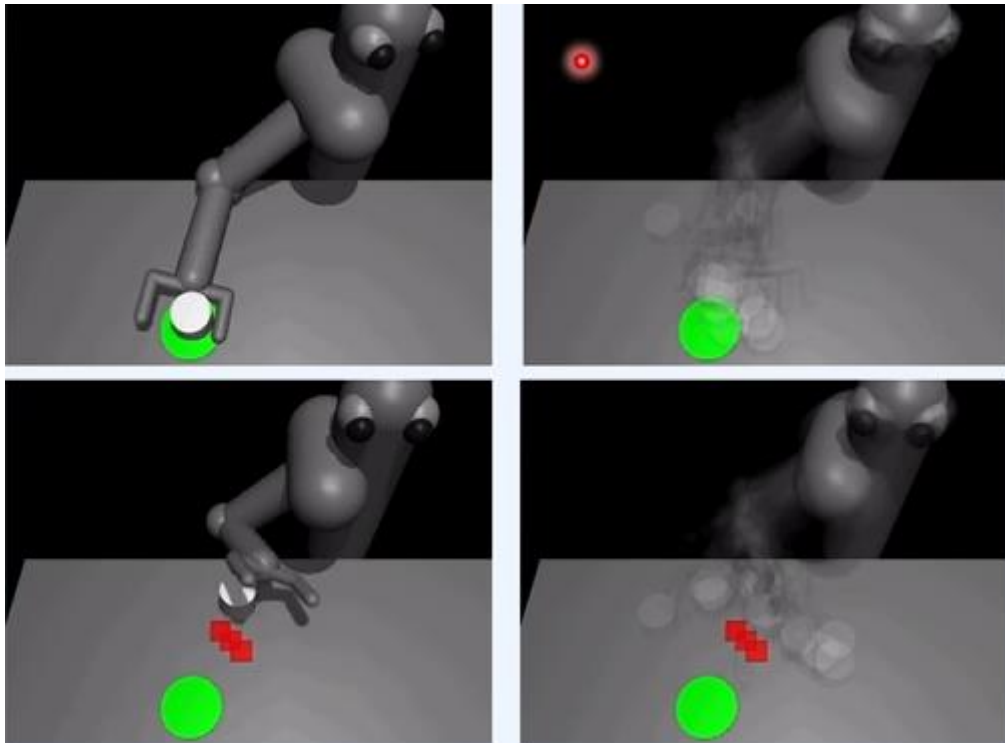


- Efficient exploration
 - Policy represents multiple promising actions till the end of learning
- Escape from local optima

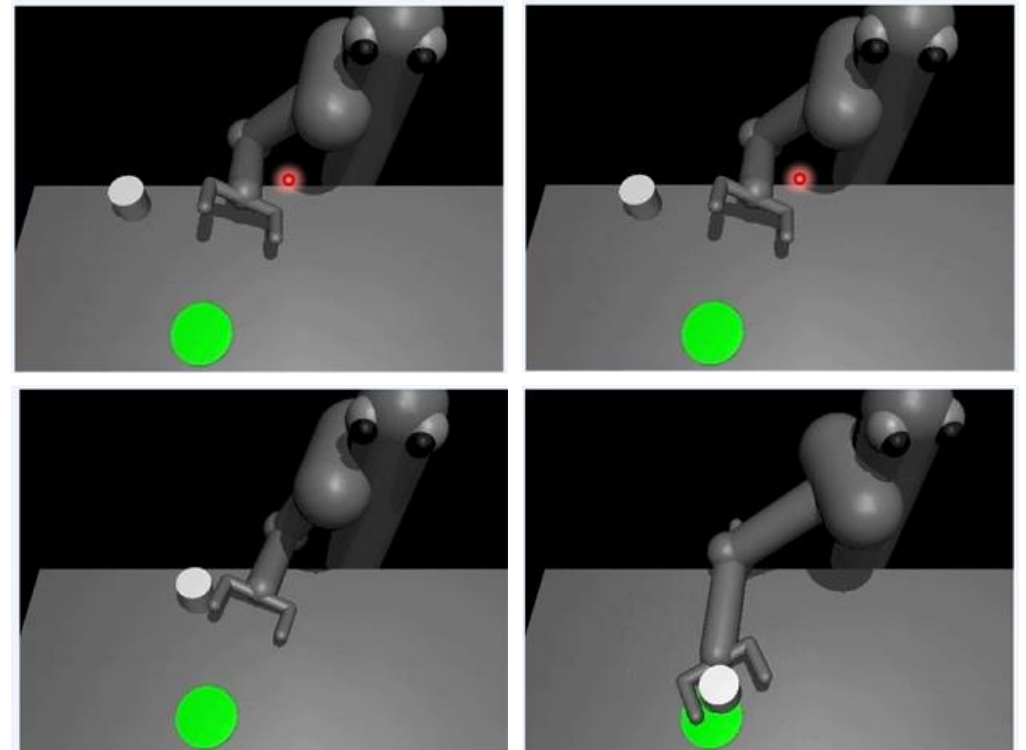


Softmax Policy

Std. RL vs. Max. Ent. RL

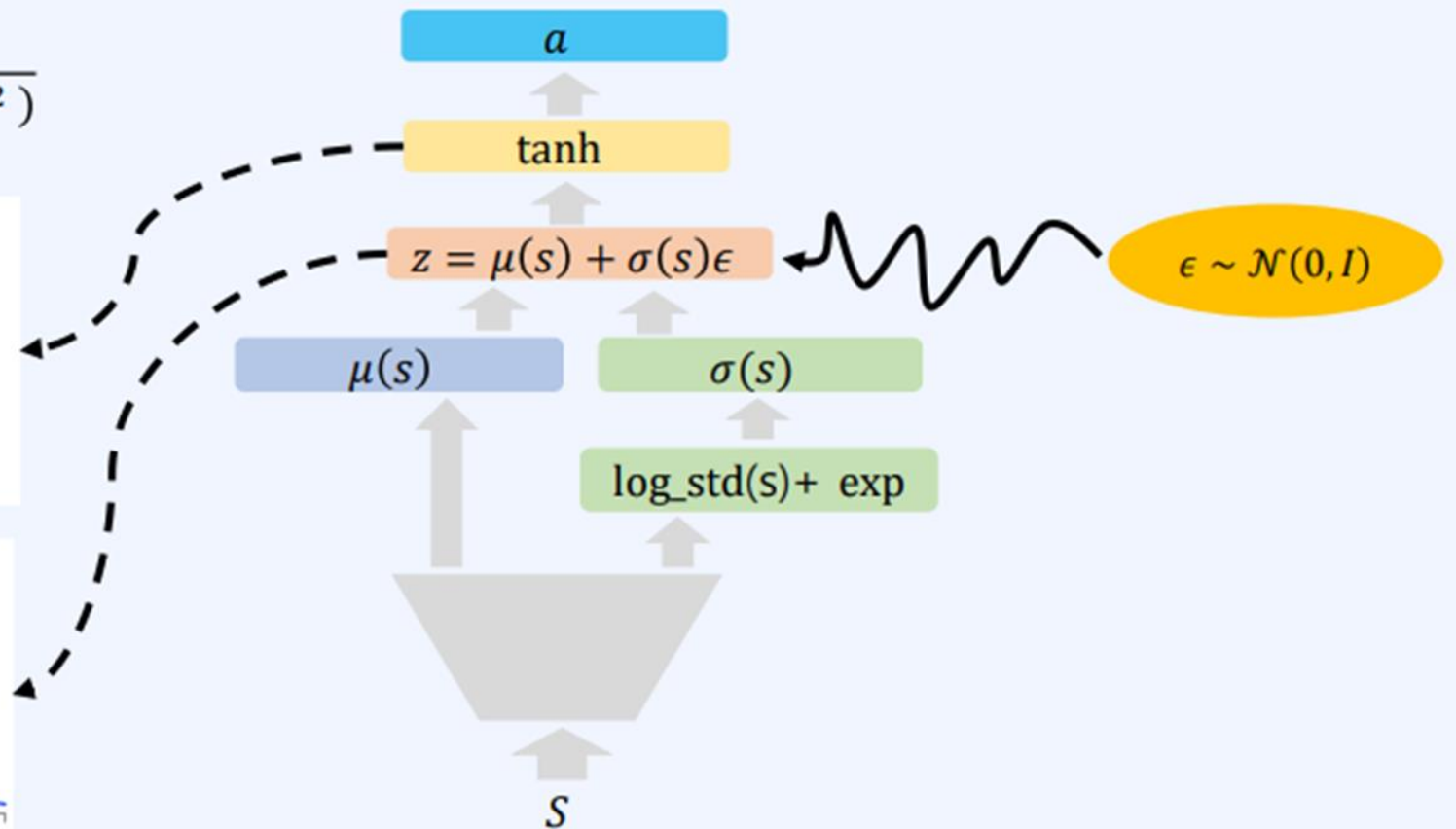
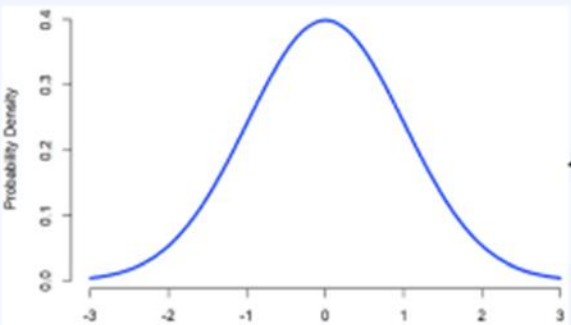
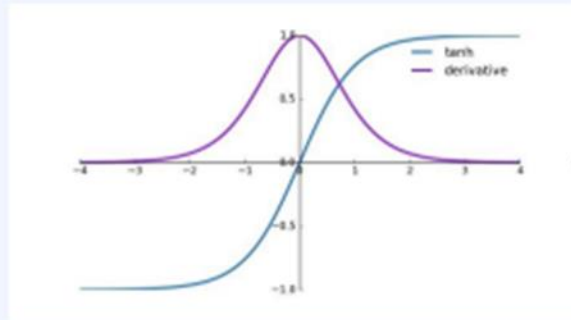


Std. RL vs. Max. Ent. RL



Soft Actor

$$\pi(a|s) = \frac{\mathcal{N}(\mu(s), \sigma(s))}{\prod_{i=1}^d (1 - (\tanh u_i)^2)}$$



Soft Actor

- $DKL(\pi(\cdot|s) \parallel \pi'(\cdot|s)) = \int \pi(a|s) \log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right) da$

$$\pi'(a|s) = \frac{\exp\left(\frac{1}{\alpha} Q(s, a)\right)}{Z^{\text{old}} Q(st)}, \quad \pi(a|s) = \frac{N(\mu(s), \sigma(s))}{\prod_{i=1}^d (1 - (\tanh u_i)^2)}$$

- Soft Actor Objective: 여기에 수식을 입력하십시오.

$$\arg \min_{\varphi} E_{a \sim \pi} \left[\log\left(\frac{\pi(a|s)}{\pi'(a|s)}\right) \right]$$

$$= E_{s \sim D} \left[E_{a \sim \pi} \left[\log \pi(a|s) - \left(\log \exp\left(\frac{1}{\alpha} Q(s, a)\right) - \cancel{\log Z(st)} \right) \right] \right]$$

$$= E_{s \sim D} \left[E_{a \sim \pi} \left[\underline{\alpha \log \pi(a|s)} - Q(s, a) \right] \right]$$

decreasing $\log \pi(a|s)$ = increasing entropy ($-\log \pi(a|s)$)

Can be ignored with no
impact on the policy
parameter differentiation

Double Q trick

- Recall the maximization bias in double Q-learning
- Use two independent Q-networks to estimate target values, and select most conservative target values.

$$\hat{y} = r + \gamma \max_{i=1,2} Q_i(s', a')$$

- Q1 and Q2 will be updated independently (each has its own target network):

$$L(\theta_i) = (Q_i(s, a) - \hat{y})$$

Double Q trick

- We must change the update rule of the Q-network:

$$L(\theta) = (Q_{\theta}(s, a) - \hat{y})$$

- Entropy of the policy:

$$H(\pi) = E[-\ln \pi]$$

- Now the target \hat{y} is:

$$\hat{y} = r + \gamma(\min_{i=1,2} Q_i(s', a') - \alpha \ln \pi(a' | s'))$$
$$a' \sim \pi(s)$$

SAC

Q target uses double-Q trick
Q target has policy entropy bonus
 $a' \sim$ the softmax policy
SGD for both Q update

Maximize future rewards sum (Q)
Maximize entropy of policy distribution

SGA for policy update

Algorithm 1 Soft Actor-Critic

- 1: Input: initial policy parameters θ , Q-function parameters ϕ_1, ϕ_2 , empty replay buffer \mathcal{D}
- 2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1, \phi_{\text{targ},2} \leftarrow \phi_2$
- 3: **repeat**
- 4: Observe state s and select action $a \sim \pi_\theta(\cdot|s)$
- 5: Execute a in the environment
- 6: Observe next state s' , reward r , and done signal d to indicate whether s' is terminal
- 7: Store (s, a, r, s', d) in replay buffer \mathcal{D}
- 8: If s' is terminal, reset environment state.
- 9: **if** it's time to update **then**
- 10: **for** j in range(however many updates) **do**
- 11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from \mathcal{D}
- 12: Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

- 13: Update Q-functions by one step of gradient descent using

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

- 14: Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} \left(\min_{i=1,2} Q_{\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s)|s) \right),$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.

- 15: Update target networks with

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

- 16: **end for**
- 17: **end if**
- 18: **until** convergence

Experiments

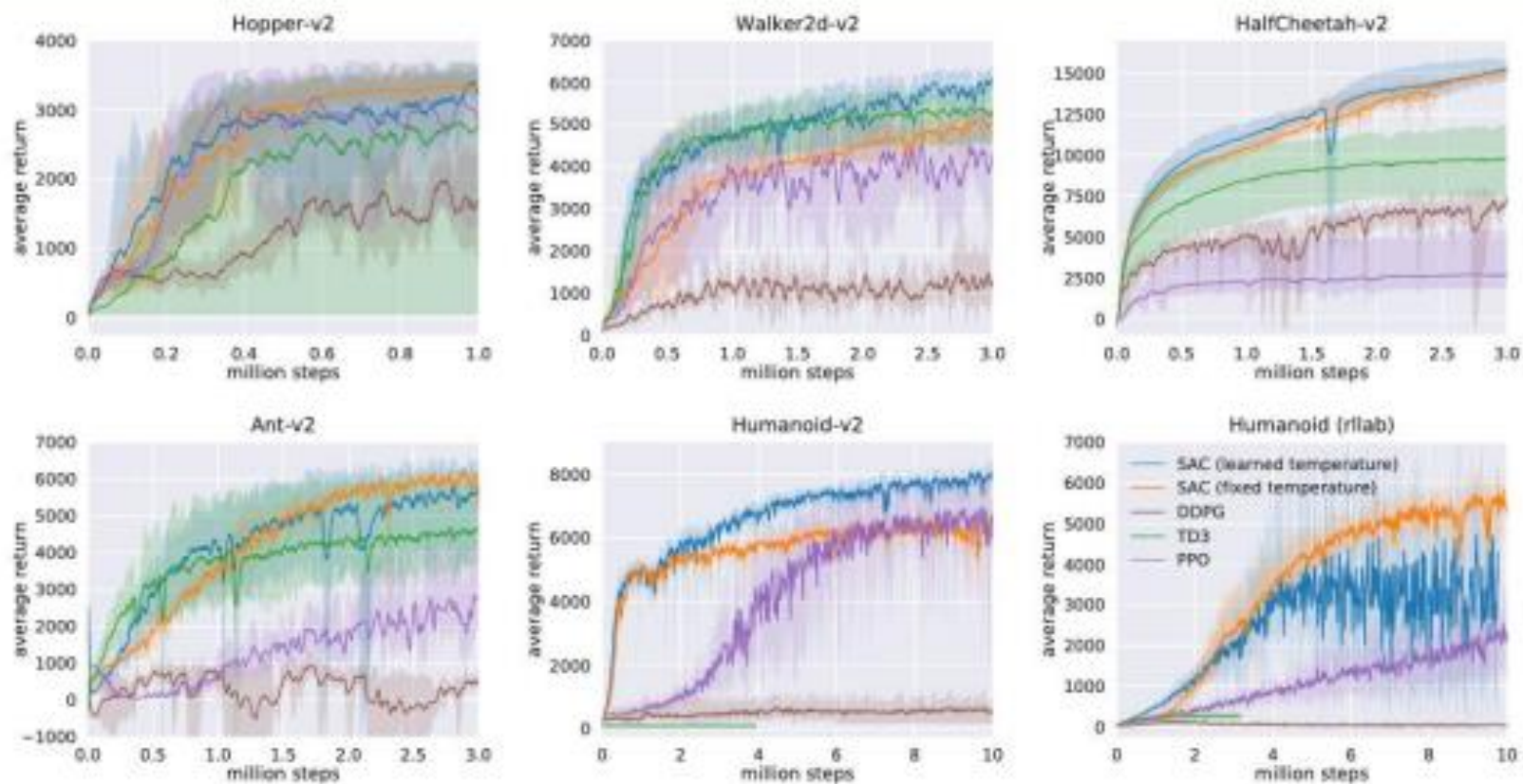
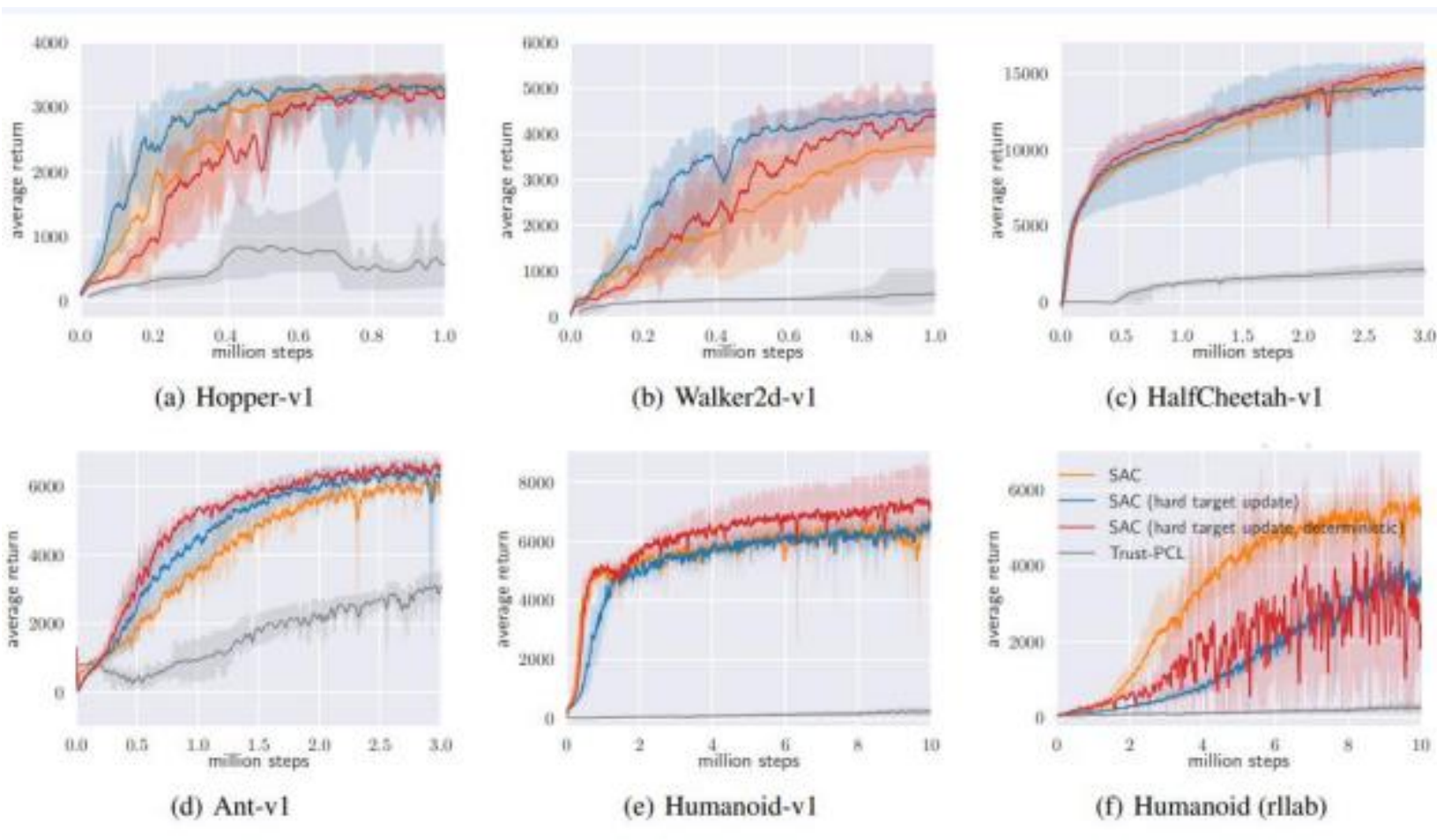


Figure 1: Training curves on continuous control benchmarks. Soft actor-critic (blue and yellow) performs consistently across all tasks and outperforming both on-policy and off-policy methods in the most challenging tasks.

Experiments



Code Ex.2

- Upload 'soft_actor_critic.ipynb' file onto Colab
- The notebook includes 'Mujoco' environment, 'Half Cheetah'
- State(17):
 - (body parts) position,
 - (roter, axis) angle or velocity
- Action(6):
 - Torque on 6 rotors (-1~1)
- Reward:
 - Forward_reward: moving-forward reward
 - ctrl_cost: penalty in taking too large action

