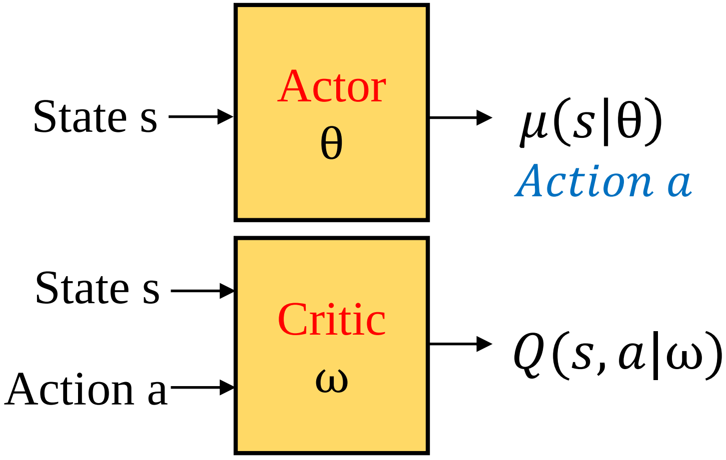
Deep Deterministic Policy Gradient (DDPG)
TD3
SAC (Soft Actor Critic)

Deep Deterministic Policy Gradient (DDPG)

Deterministic Policy Gradient

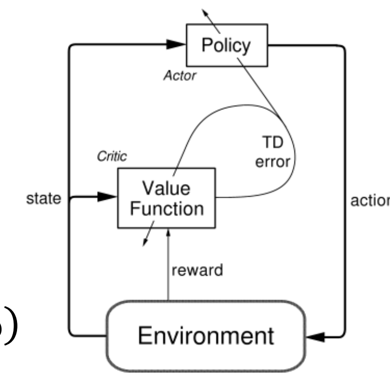
- Deterministic policy gradient can be estimated more efficiently, especially in high-dimensional continuous action spaces
  - Deterministic policy integrates over only states space
  - Use off-policy learning to ensure adequate exploration

[Lillicrap, et al., 2016] "Continuous control with deep reinforcement learning," in 4th International Conference on Learning Representations (ICLR 2016).



Deep Deterministic Policy Gradient (DDPG) (A Kind of Actor-Critic For Continuous Actions)

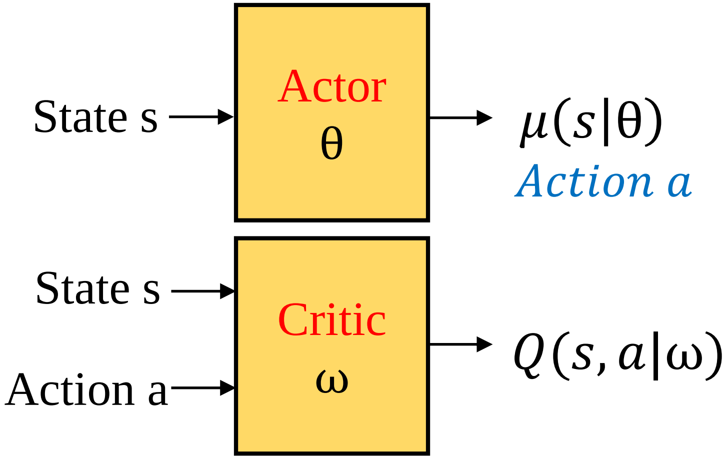
- Use two networks: an actor and a critic
  - Critic estimates value of current action by Q-learning



$$\nabla_\omega L_Q(s_t, a_t|\omega) = \left(\left(r_{t+1} + \gamma Q(s_{t+1}, \mu(s_{t+1}|\theta)|\omega)\right) - Q(s_t, a_t|\omega)\right) \nabla_\omega Q(s_t, a_t|\omega)$$

  - Actor updates policy in direction suggested by critic (DDPG):

$$\nabla_\theta J(\mu_\theta) \approx \mathbb{E}_\mu[\nabla_\theta Q(s_t, \mu(s_t|\theta)|\omega)] = \mathbb{E}_\mu\left[\nabla_a Q(s_t, a|\omega)\Big|_{a=\mu(s_t|\theta)} \nabla_\theta \mu(s_t|\theta)\right]$$



DDPG(1/2)

Behavior and target network

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 $t = 1, T$ **do**

Select action $a_t = \mu(s_t|\theta^\mu) + \boxed{N_t}$ A noise process

Execute action a_t and observe reward r_t and observe new state s_{t+1}

Experience replay

Store transition (s_t, a_t, r_t, s_{t+1}) in R
Sample random minibatch of M transitions (s_j, a_j, r_j, s_{j+1}) from R

Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$

Update critic by minimizing the loss: $L = \frac{1}{M}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$

Update the actor policy using the sampled gradient:

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

Update the target networks:

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

DDPG(2/2)

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 $t = 1, T$ **do**

Select action $a_t = \mu(s_t|\theta^\mu) + N_t$
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 random minibatch of M transitions (s_j, a_j, r_j, s_{j+1}) from R
Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'})$

Update the behavior networks (both actor and critic)

Update critic by minimizing the loss: $L = \frac{1}{M}\sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$
Update the actor policy using the sampled gradient:

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

Update the target networks:

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

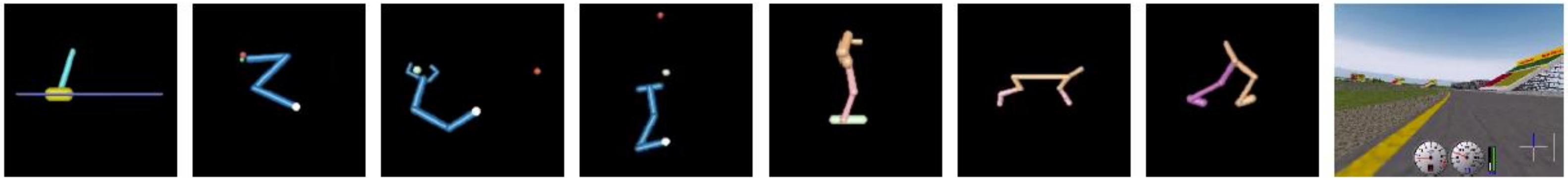
Apply “soft” target updates
$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tau \ll 1$

(0.001 in practice.)
(Note in DQN, θ is copied periodically.
Later, some DQN also used this way)

I-Chen Wu

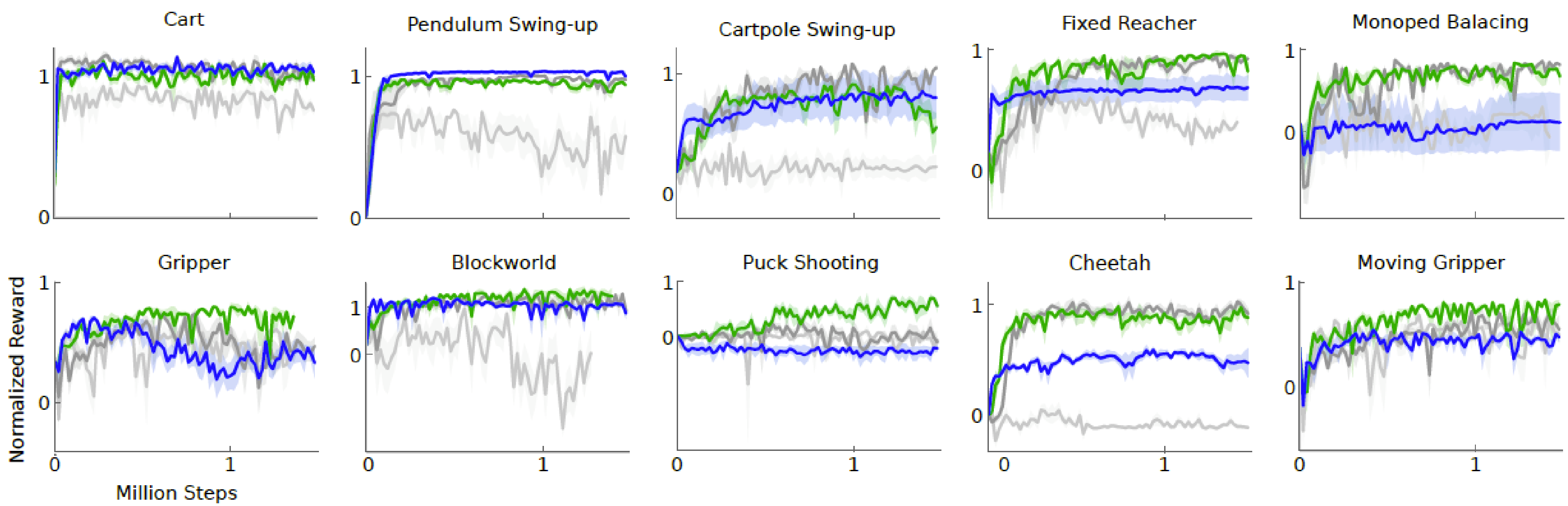
Experiment Settings

- Run experiments using both a low-dimensional state description and high-dimensional renderings of the environment
- The frames were downsampled to 64x64 pixels and the 8-bit RGB values were converted to floating point scaled to $[0, 1]$



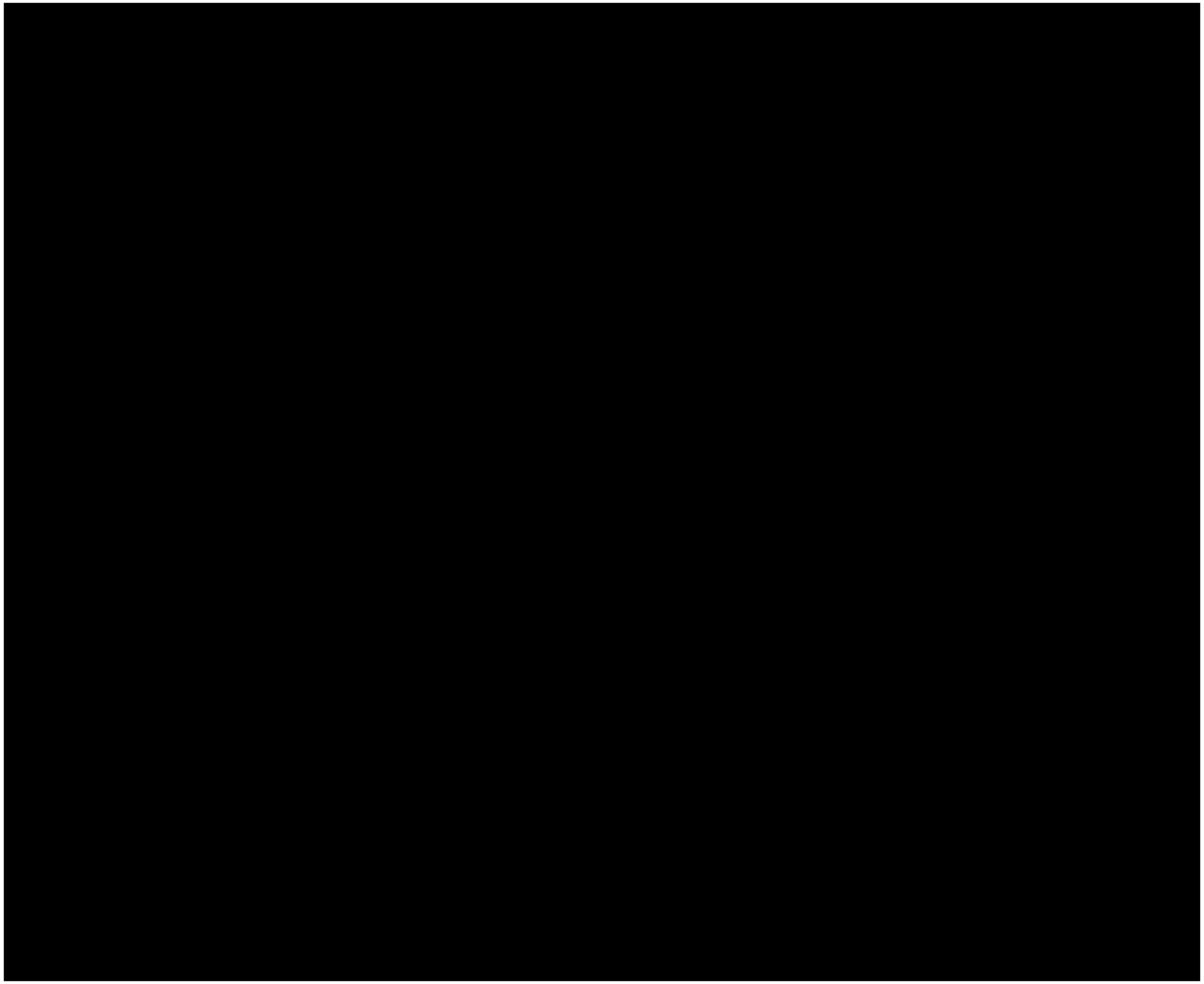
Example screenshots of a sample of environments to solve with DDPG.

Performance Curves for Those Using Variants of DPG



Light Gray: State Description + Batch Normalization
Dark Gray: State Description + Target Network
Green: State Description + Batch Normalization + Target Network
Blue: Pixels + Target Network

Demo



Twin Delayed DDPG (TD3)

Addressing Function Approximation Error in Actor-Critic Methods

Scott Fujimoto, Herke van Hoof and David Meger. “Addressing Function Approximation Error in Actor-Critic Methods.” ICML (2018).

DDPG Overview

initial $\theta, \theta', \phi, \phi'$, replay buffer B
for episode = 1~M **do**
  for t = 1~T **do**
    Select action using π_ϕ
    Play and store transition in B
    Sample a batch from B

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi'}(s'))$$

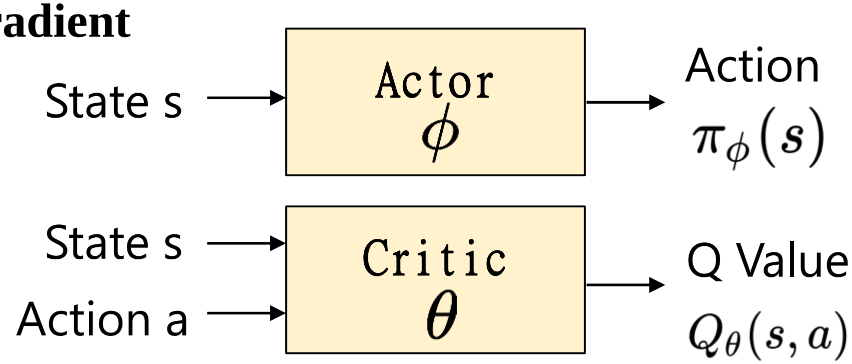
    Update Behavior Critic θ using y
    Update Behavior Actor ϕ using **policy gradient**
    Update Target

$$\theta' \to \tau\theta + (1-\tau)\theta'$$

$$\phi' \to \tau\phi + (1-\tau)\phi'$$

	Actor	Critic
Behavior	ϕ	θ
Target	ϕ'	θ'

Network Weight Notation



Method

- Twin Delayed DDPG (TD3)

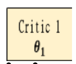
- TD3 = DDPG + 3 Tricks
  - Clipped Double Q-Learning
  - Delayed Policy Updates
  - Target Policy Smoothing

TD3 Overview

initial $\theta, \theta', \phi, \phi'$, replay buffer B

for episode = 1~M **do**

  for t = 1~T **do**

    Select action using 

    Play and store transition in B

    Sample a batch from B

Trick 1

$$y = r + \gamma \boxed{\min_{i=1,2} Q_{\theta'_i}}(s', \pi_{\phi'}(s') \boxed{+\, \epsilon})$$ Trick 3

    Update Behavior Critic θ_1, θ_2 using y

Trick 2 **if** t mod d **then**

      Update Behavior Actor ϕ using **policy gradient**

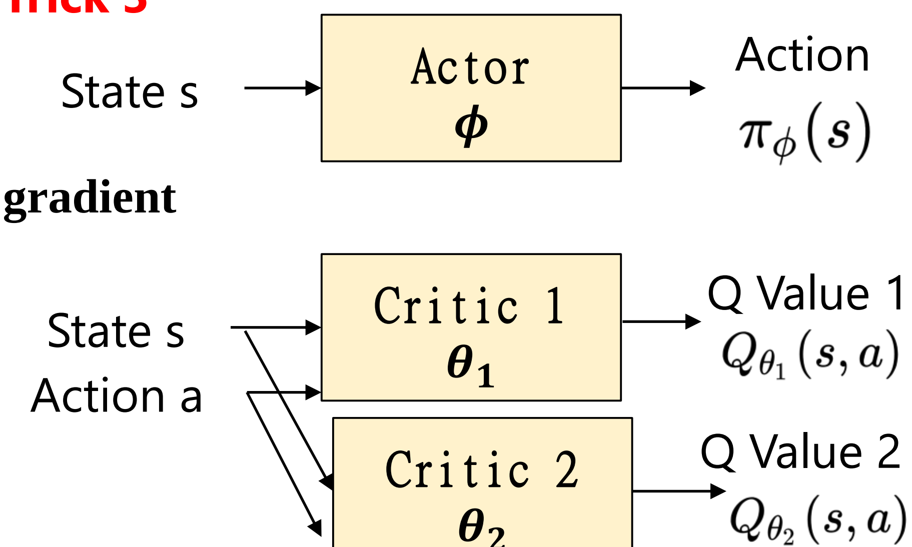
      Update Target

$$\theta'_i \to \tau\theta_i + (1-\tau)\theta'_i$$

$$\phi' \to \tau\phi + (1-\tau)\phi'$$

	Actor	Critic
Behavior	ϕ	θ_1, θ_2
Target	ϕ'	θ'_1, θ'_2

Network Weight Notation



I-Chen Wu

Trick 1 : Clipped Double-Q Learning

- Origin DDPG (Not Good)

$$y = r + \gamma Q_{\theta'}(s', \pi_{\boxed{\phi'}}(s'))$$

- Methods to solve overestimation problem
  - Double DQN (Not Good Enough)

$$y = r + \gamma Q_{\theta'}(s', \pi_{\boxed{\phi}}(s'))$$

  - Double-Q Learning (Not Good Enough)

$$y_1 = r + \gamma Q_{\theta'_2}(s', \pi_{\phi_1}(s'))$$
$$y_2 = r + \gamma Q_{\theta'_1}(s', \pi_{\phi_2}(s'))$$

	Actor	Critic
Behavior	ϕ	θ
Target	ϕ'	θ'

Network Weight Notation

(Recall) Overestimation Problem

- Q-Learning update

$$Q(s,a) = r + \gamma \max_{a'} Q(s',a')$$

Trick 1 : Clipped Double-Q Learning

- Methods to solve overestimation problem
  - Double DQN (Not Good Enough)

$$y = r + \gamma Q_{\theta'}(s', \pi_{\phi}(s'))$$

  - Double-Q Learning (Not Good Enough)

$$y_1 = r + \gamma Q_{\theta'_2}(s', \pi_{\phi_1}(s'))$$
$$y_2 = r + \gamma Q_{\theta'_1}(s', \pi_{\phi_2}(s'))$$

	Actor	Critic
Behavior	ϕ	θ_1, θ_2
Target	ϕ'	θ'_1, θ'_2

Network Weight Notation

- Clipped Double-Q Learning (Better)

$$\boxed{y} = r + \gamma \min[Q_{\theta'_1}(s', \boxed{\pi_{\phi}}(s')), Q_{\theta'_2}(s', \boxed{\pi_{\phi}}(s'))]$$

Only one Q target

Only one actor

Trick 2 : Delayed Policy Updates

- Use lower frequency to update behavior actor and target networks.

```
initial
for episode = 1~M do
    for t = 1~T do
        ...
        Update Behavior Critic
        Update Behavior Actor
        Update Targets Networks
```

```
initial
for episode = 1~M do
    for t = 1~T do
        ...
        Update Behavior Critic
        if t mod d then
            Update Behavior Actor
            Update Targets Networks
```

Hyperparameter

Trick 3 : Target Policy Smoothing

- Assumption
  - Similar actions have similar values

- Add noise to action value

$$y = r + \gamma Q(s', \pi(s') + \epsilon), \epsilon \sim clip(\mathcal{N}(0, \underbrace{\sigma), -c, c}_{\text{Hyperparameters}})$$

- Regularization

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

I-Chen Wu

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1, \theta'_2 \leftarrow \theta_2, \phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

2. Delayed Policy Updates

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{B}$
for $t = 1$ **to** T **do**
  Select action with exploration noise $a \sim \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma)$ and observe reward r and new state s'
  Store transition tuple (s, a, r, s') in $\mathcal{B}$

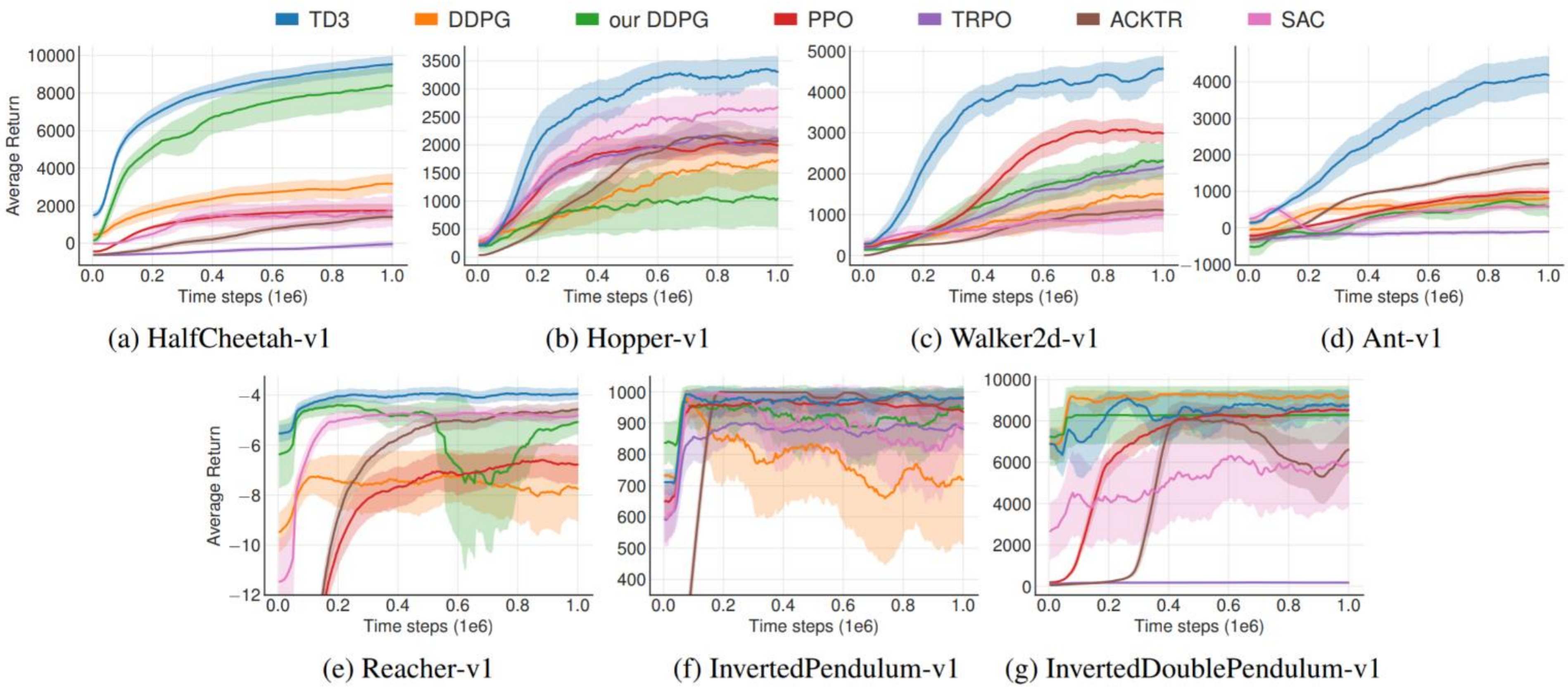
  Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
  $\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
  $y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta'_i}(s', \tilde{a})$
  Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s, a))^2$
  if t mod d **then**
    Update ϕ by the deterministic policy gradient:
    $\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s, a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$
    Update target networks:
    $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
  end if
end for

1. Clipped Double Q-Learning for Actor-Critic

2. Delayed Policy Updates

3. Target Policy Smoothing Regularization

Experiment



(a) HalfCheetah-v1 (b) Hopper-v1 (c) Walker2d-v1 (d) Ant-v1

(e) Reacher-v1 (f) InvertedPendulum-v1 (g) InvertedDoublePendulum-v1

Experiments: Compared to Others

Environment	TD3	DDPG	Our DDPG	PPO	TRPO	ACKTR	SAC
HalfCheetah	**9636.95 ± 859.065**	3305.60	8577.29	1795.43	-15.57	1450.46	2347.19
Hopper	**3564.07 ± 114.74**	2020.46	1860.02	2164.70	2471.30	2428.39	2996.66
Walker2d	**4682.82 ± 539.64**	1843.85	3098.11	3317.69	2321.47	1216.70	1283.67
Ant	**4372.44 ± 1000.33**	1005.30	888.77	1083.20	-75.85	1821.94	655.35
Reacher	**-3.60 ± 0.56**	-6.51	**-4.01**	-6.18	-111.43	-4.26	-4.44
InvPendulum	**1000.00 ± 0.00**	**1000.00**	**1000.00**	**1000.00**	985.40	**1000.00**	**1000.00**
InvDoublePendulum	**9337.47 ± 14.96**	**9355.52**	8369.95	8977.94	205.85	9081.92	8487.15

SAC (Soft Actor Critic)

Reference

- Haarnoja, T., Tang, H., Abbeel, P., & Levine, S. (2017). Reinforcement Learning with Deep Energy-Based Policies. ICML.
- Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. ArXiv, abs/1801.01290.
- Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018). Soft Actor-Critic Algorithms and Applications. ArXiv, abs/1812.05905.
- Open source:
  - https://github.com/haarnoja/sac (original author)
    https://github.com/rail-berkeley/softlearning

- Credit goes to Guo-Hao Ho for most of the slides.

Introduction

- SAC is
  - Open-source (by original authors)
    - https://sites.google.com/view/sac-and-applications
  - Perform well (as in realistic environment)
  - Key idea is easy to understand
    - Maximum entropy reinforcement learning

Introduction

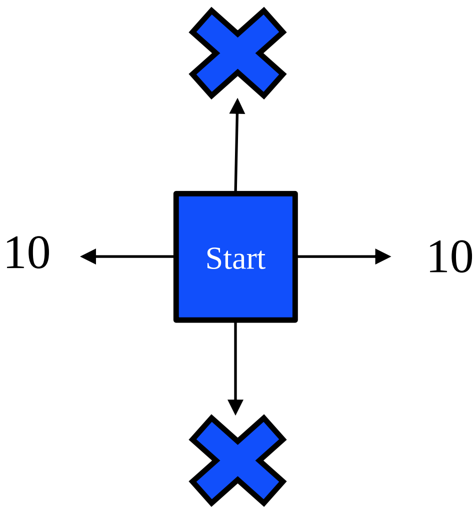
- Soft actor critic (SAC) train a policy that maximizes a trade-off between expected return and entropy
  – Still getting high performance while acting as random as possible
  – Augment the objective function with entropy term
- Evolution of SAC
  – Soft Q-learning (SQL)
  → Soft Actor-Critic (SAC)
  → Soft Actor-Critic with automating entropy adjustment(SAC)

Problem

- The above methods (PPO, DDPG) focus more on exploitation
  - The objective function is mainly based on the return
    - May be trapped in local optimum without exploration

Extremely simple case

Return	Up	Left	Down	Right
	0	10	0	10



Policy	Up	Left	Down	Right
T=0	0.25	0.25	0.25	0.25
T=1	0.2	0.4	0.2	0.2
…				
T=n	0	1	0	0

If we sampled “left” first

Without any exploration, the chance to sample the “right” is harder, resulting in the policy converges to “left” gradually

The agent will be

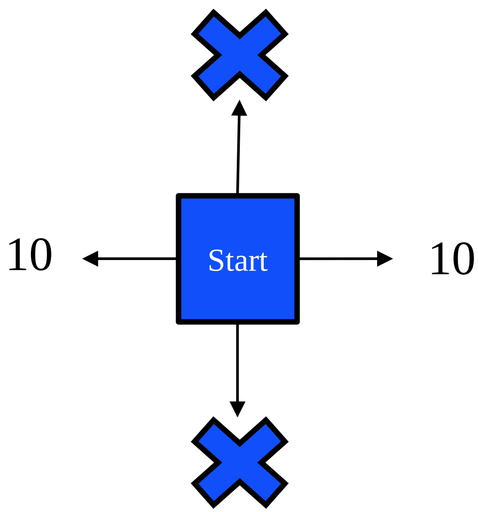
- either right or left with100%
- not right and left with 50%

Problem

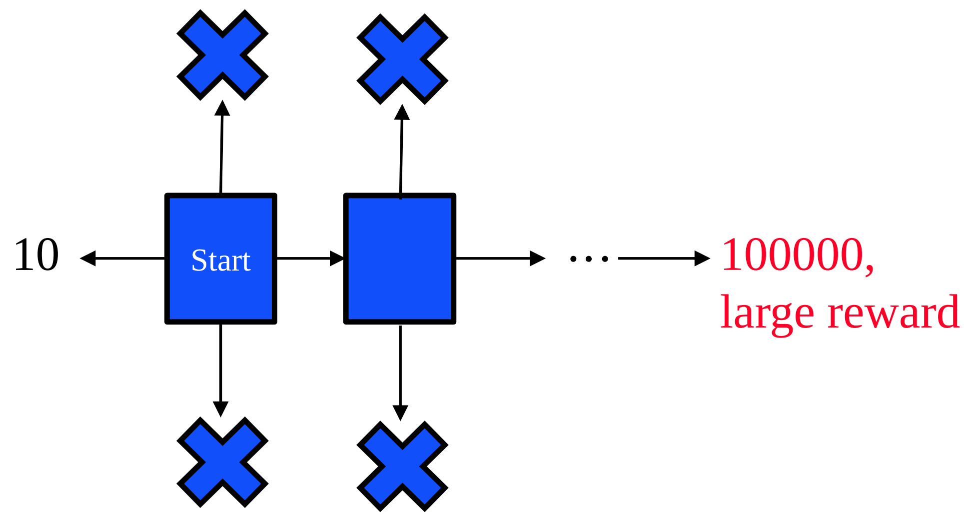
- Hard exploration case
  - Extend previous “extremely simple case”

The agent will be either right or left for 100%
But not right and left for 50%

Hard for agent to discover policy of “right”
May trap in policy of “left”



Extremely simple case



Hard exploration case

Problem-Solution

- The exploration ability relies on
  - Random noise in selected action
    E.g. DDPG
    - During training, the action is disturbed with the random noise

Algorithmus 4 : Deep Deterministic Policy-Gradient

Result : policy parameter $\boldsymbol{\theta}$ and action-value weights $\mathbf{w}$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and action-value weights $\mathbf{w} \in \mathbb{R}^{d}$;
Initialize target policy parameter $\boldsymbol{\theta}' \in \mathbb{R}^{d'}$ and target action-value weights $\mathbf{w}' \in \mathbb{R}^{d}$;
Initialize experience replay memory $\mathcal{D}$;
for *episode* $= 1,\ M$ **do**
  Observe initial state s_0 from environment ;
  for $t = 1, T$ **do**
    Select action $a_t = \tau(s, \boldsymbol{\theta}_t) + \mathcal{N}_t$;
    Observe reward r_t and next state s_{t+1} from environment ;
    Store (s_t, a_t, r_t, s_{t+1}) tupel in $\mathcal{D}$;
    Sample random batch (s_i, a_i, r_i, s_{i+1}) of size B from $\mathcal{D}$;
    $\delta_i \leftarrow r_i + \gamma \hat{q}(s_{i+1}, \tau(s_{i+1}, \boldsymbol{\theta}_t^{'}), \mathbf{w}_t^{'}) - \hat{q}(s_i, a_i, \mathbf{w}_t)$;
    $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \beta \frac{1}{B} \sum_i^B \delta_i \nabla_{\mathbf{w}} \hat{q}(s_i, a_i, \mathbf{w}_t)$;
    $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t + \alpha \frac{1}{B} \sum_i^B \nabla_{\boldsymbol{\theta}} \hat{q}(s_i, \tau(s_i, \boldsymbol{\theta}_t), \mathbf{w}_t) \nabla_{\boldsymbol{\theta}} \tau(s_i, \boldsymbol{\theta}_t)$;
    Update target networks by

$$\boldsymbol{\theta}_{t+1}^{'} \leftarrow \upsilon \boldsymbol{\theta}_t + (1-\upsilon)\boldsymbol{\theta}_t^{'}$$
$$\mathbf{w}_{t+1}^{'} \leftarrow \upsilon \mathbf{w}_t + (1-\upsilon)\mathbf{w}_t^{'}$$

  end
end

I-Chen Wu

Problem-Solution

- The exploration ability relies on
  - Random noise in selected action
    E.g. DDPG
  - Entropy regularization in objective
    E.g. PPO
    $L_t^{CLIP+VF+S}(\theta) = \widehat{E_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 \boxed{S[\pi_\theta](s_t)]}$
    - To maximum the objective, policy π_θ gets less entropy bonus $S[\pi_\theta]$ if π_θ is deterministic

Maximum Entropy Reinforcement Learning

- Standard reinforcement learning (RL) objective function:
  - Total expected rewards:

$$J(\pi_\theta) = \sum_t E_{(s_t, a_t) \sim \rho_{\pi_\theta}} [r(s_t, a_t)]$$

where ρ_{π_θ} is data distribution for policy π_θ

- Maximum entropy RL objective function:
  - Augment with entropy term:

$$J(\pi_\theta) = \sum_t E_{(s_t, a_t) \sim \rho_{\pi_\theta}} [r(s_t, a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

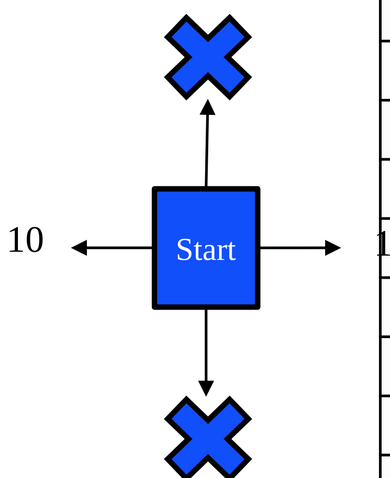
where α is temperature for importance of the entropy term

Maximum Entropy Reinforcement Learning

$$J(\pi_\theta) = \sum_t E_{(s_t, a_t) \sim \rho_{\pi_\theta}}[r(s_t, a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

Return	Up	Left	Down	Right
	0	10	0	10

- Example:
  $J(\pi_\theta) = r(s_t, a_t) - log(\pi_\theta(s_t, a_t))$,
  Assume α=1



Extremely simple case

Policy	Up	Left	Down	Right
T=0	0.25	0.25	0.25	0.25
$J(\pi_\theta)$=	0-log0.25	10-log0.25	0-log0.25	10-log0.25
T=1	0.2	0.4	0.2	0.2
$J(\pi_\theta)$=	0-log0.2	10-log0.4	0-log0.2	10-log0.2
…				
T=k	10^{-10}	≈1	10^{-10}	10^{-10}
$J(\pi_\theta)$=	0-log10^{-10}	10-log1	0-log10^{-10}	10+10
…				
T=n	0	0.5	0	0.5

Ideal convergence

If we sampled “left” first

- Encourage take this action (“right”) with entropy term
- The exploration bonus is vanish when the policy become deterministic

Maximum Entropy Reinforcement Learning

- Encourage exploration with entropy term
  - Entropy in loss function: Consider entropy as regularized term
    - E.g.: PPO

$$L_t^{CLIP+VF+S}(\theta) = \widehat{E_t}[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]$$

The entropy term only cares the current state

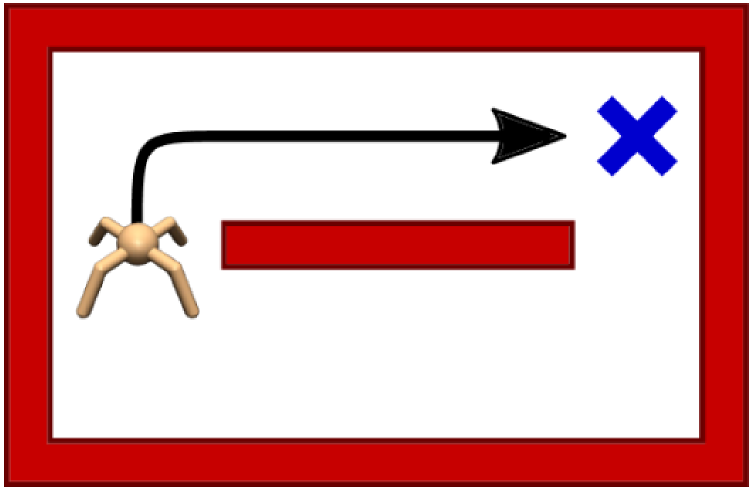
  - Entropy in objective function: Consider entropy as incentivized exploration reward
    - E.g.: SAC

$$J(\pi_\theta) = \sum_t E_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t, a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$

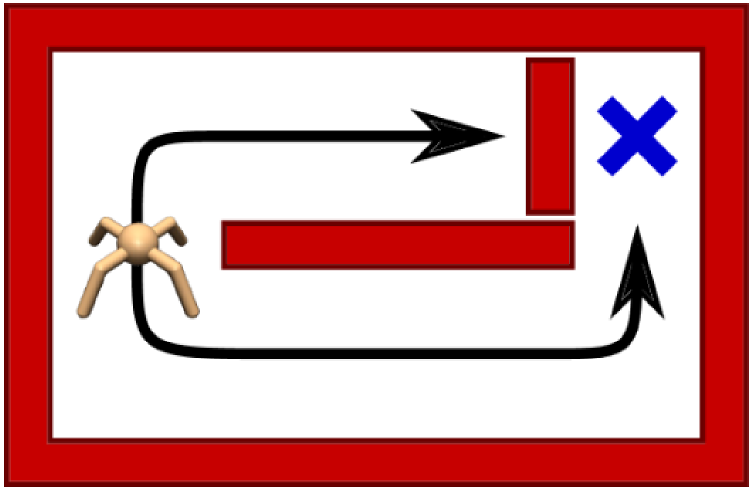
The entropy term affects following future states by accumulated return

Soft Actor Critic

- Soft Q-learning
- Soft actor critic
- Soft actor critic with automating entropy adjustment



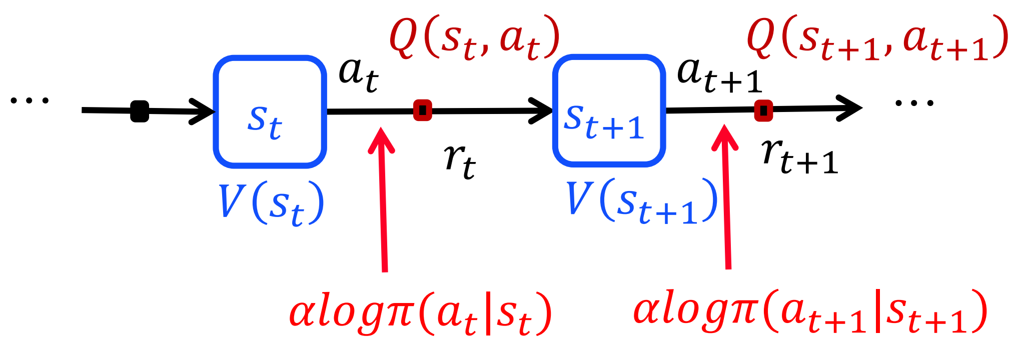
2a



2b

Soft Q-Learning

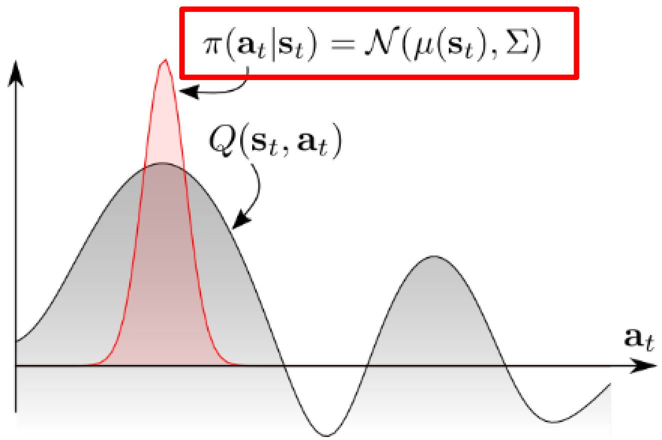
- Objective function: Maximum entropy RL
$$J(\pi_\theta) = \sum_t E_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha H(\pi_\theta(\,.\,|s_t))]$$
- Soft V-function:
$$V_{soft}(s_t) = E_{a_t\sim\pi_\theta}[Q_{soft}(s_t,a_t) - \alpha log\pi(a_t|s_t)]$$
- Soft Q-function:
$$Q_{soft}(s_t,a_t) = r_t + \gamma E_{s_{t+1}\sim\rho_{\pi_\theta}}[V_{soft}(s_{t+1})]$$
- Authors prove augment the entropy term still follow Bellman equation property
  - Policy evaluation
  - Policy improvement
  - Policy iteration



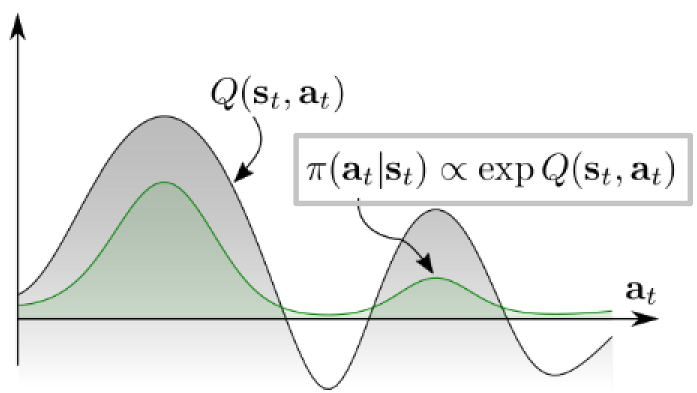
Soft Q-Learning

- Gaussian policy:
  - For convenient, usually assume the policy distribution is Gaussian distribution
  - Problem: Not suitable for multimodal case
- Energy-based policy:
  - Use Q value distribution to indicate the policy distribution
  - Assumption: $\pi(a_t|s_t) \propto \exp(Q(s_t, a_t))$

Gaussian policy



Energy-based policy:
Stochastic policy with multimodal

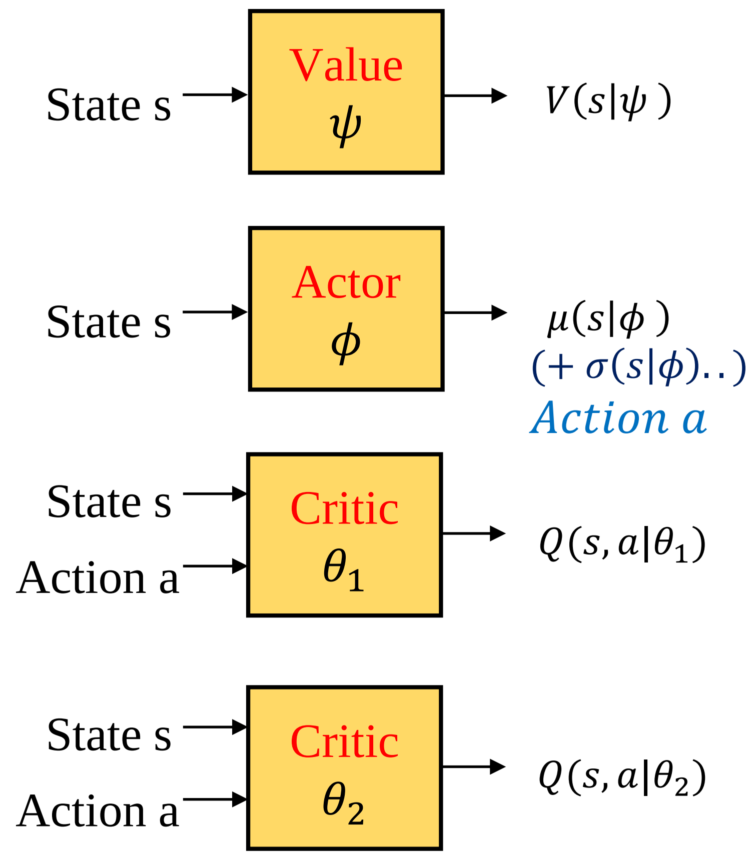


Soft Actor Critic

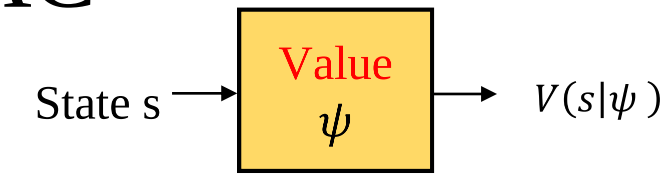
- Policy: (ideal)

  $\pi(a_t|s_t) = \exp(\frac{1}{\alpha}(Q_{soft}(s_t, a_t) - V_{soft}(s_t)))$

- Architecture
  - 1 state value (V_ψ) network
  - 1 policy network (π_ϕ)
  - 2 action-state value (Q-value) network (Q_θ)
    - Double Q trick: Prevent overestimated in Q
    - Like TD3



Training of SAC



- D is the distribution of sampled states and actions
  - Value network (V_ψ):

$$J_V(\psi) = E_{s_t\sim D}[\frac{1}{2}\left(V_\psi(s_t) - \widehat{V_\psi}(s_t)\right)^2]$$

where $\widehat{V_\psi}(s_t) = E_{a_t\sim\pi_\phi}[Q_\theta(s_t, a_t) - \alpha log\pi_\phi(a_t|s_t)]$

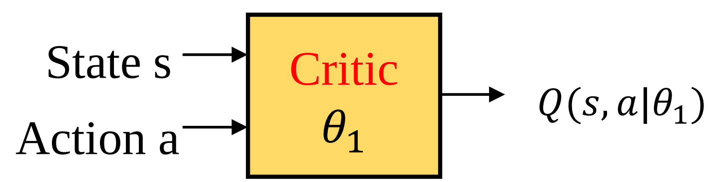
Trained by minimizing the squared residual error (TD error)

  - Q-Value network (Q_θ):

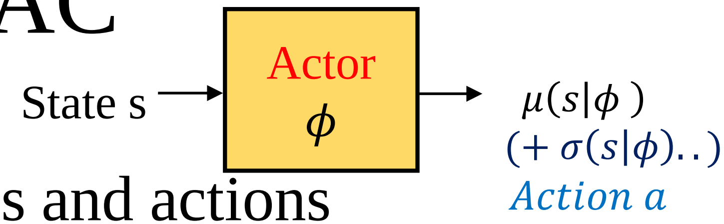
$$J_Q(\theta) = E_{(s_t,a_t)\sim D}[\frac{1}{2}\left(Q_\theta(s_t, a_t) - \widehat{Q_\theta}(s_t, a_t)\right)^2]$$

where $\widehat{Q_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma E_{s_{t+1}\sim p}[V_\psi(s_{t+1})]$

Trained by minimizing the soft Bellman residual error (TD error)



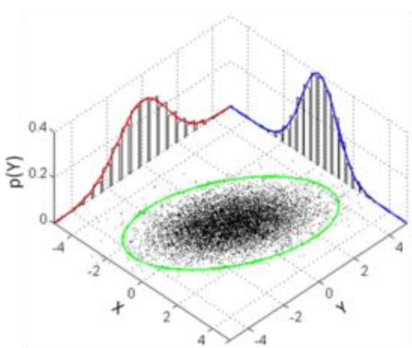
Training of SAC



- D is the distribution of sampled states and actions
- Policy network (π_ϕ)
  - Train by minimizing the KL-divergence
  - Use reparameterization trick, sample action from fixed distribution

    $$J_\pi(\phi) = E_{s_t \sim D, \epsilon_t \sim N}\left[log\pi_\phi\left(f_\phi(\epsilon_t; s_t)\middle|s_t\right) - Q_\theta(s_t, f_\phi(\epsilon_t; s_t))\right]$$

    - $a_t = f_\phi(\epsilon_t; s_t)$,
    - ϵ_t is a noise vector
    - E.g.: $f_\phi(\epsilon_t; s_t)$ as spherical Gaussian distribution
    - Take gradient $\nabla_\phi J_\pi(\phi)$



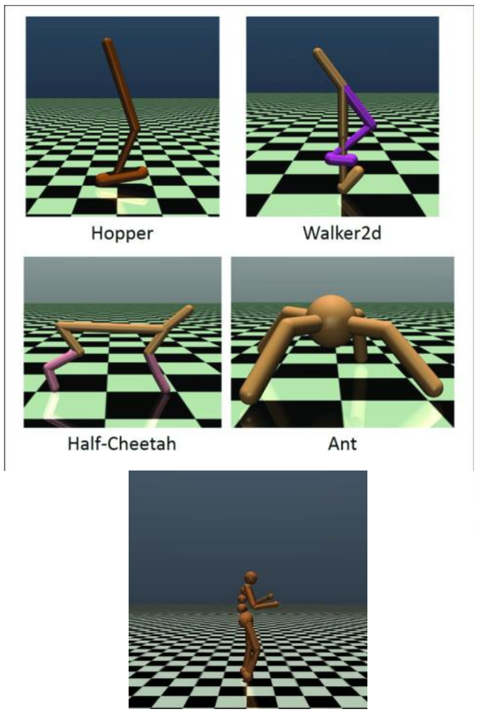
SAC Algorithm

Algorithm 1 Soft Actor-Critic

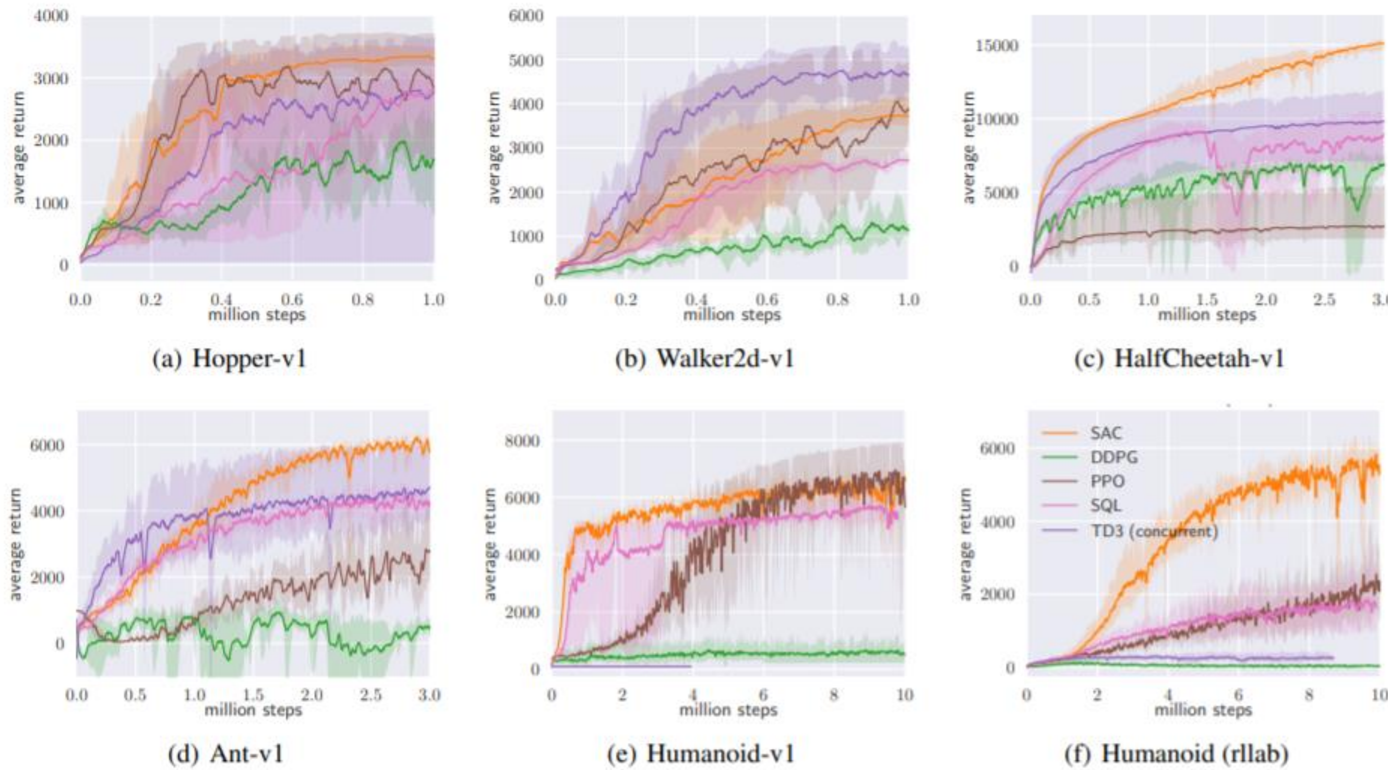
Initialize parameter vectors ψ, $\bar{\psi}$, θ, ϕ.
for each iteration **do**
  for each environment step **do**
    $\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t)$
    $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
  end for
  for each gradient step **do**
    $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$
    $\theta_i \leftarrow \underline{\theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i) \text{ for } i \in \{1, 2\}}$ Double Q trick
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$
    $\bar{\psi} \leftarrow \tau\psi + (1-\tau)\bar{\psi}$
  end for
end for

Result

- OpenAI gym v1



Humanoid



Conclusion

- Soft actor critic (SAC) train a policy that maximize a trade-off between expected return and entropy
  - Still getting high performance while acting as random as possible
- Evolution of SAC
  - Soft Q-learning (SQL)
    - Soft: $\pi \propto Q(s, a)$
  - Soft Actor-Critic (SAC)
    - Argument the objective function with entropy term
  - Soft Actor-Critic with auto-adjusted temperature (SAC)
    - Argument the objective function with entropy term
    - Auto-adjust temperature
      - By constrained policy optimization