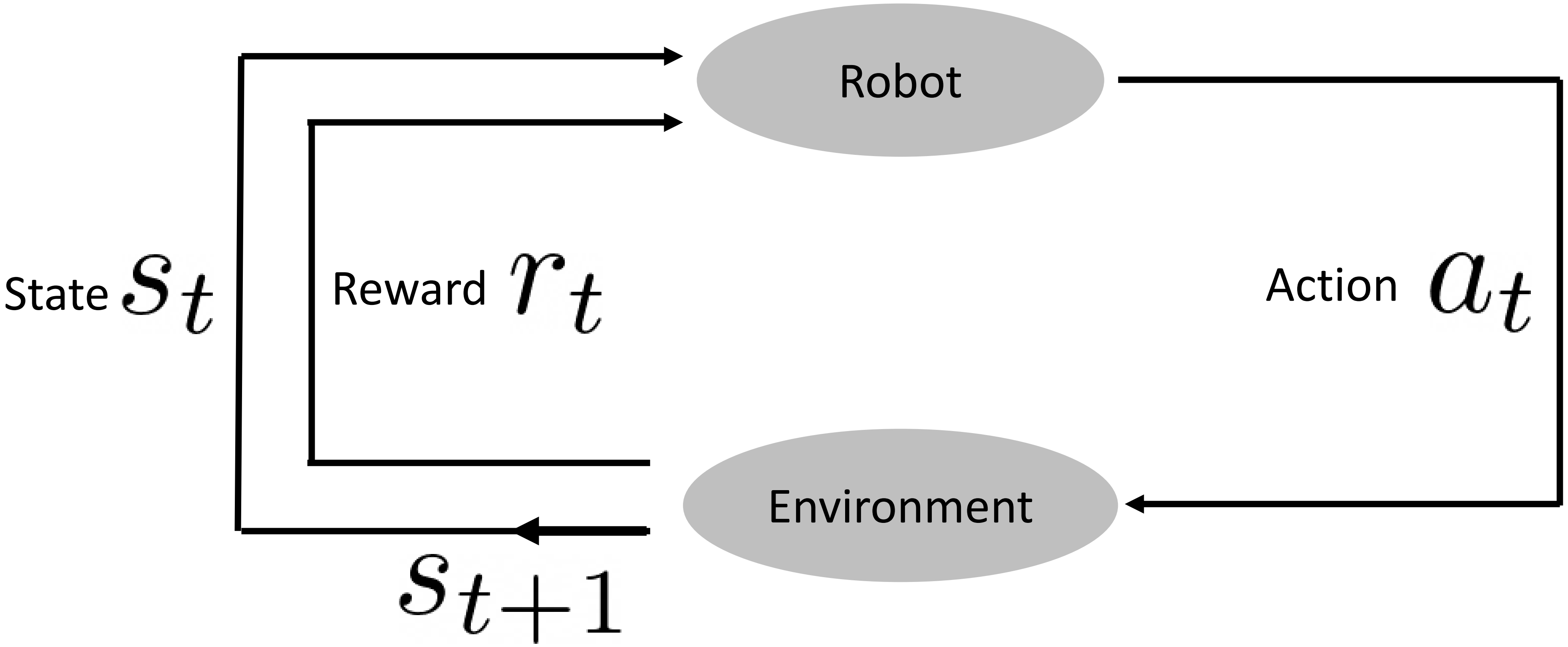
Reinforcement Learning: Algorithms

CS 6301 Special Topics: Introduction to Robot Manipulation and Navigation

Professor Yu Xiang

The University of Texas at Dallas

Reinforcement Learning



Reinforcement Learning:
Imitation Learning:

$$a_t = \pi(s_t)$$

Policy Gradient

- Maximize expected return $J(\pi_\theta) = \underset{\tau \sim \pi_\theta}{\mathrm{E}}[R(\tau)]$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

- Gradient ascent $\theta_{k+1} = \theta_k + \alpha \left. \nabla_\theta J(\pi_\theta) \right|_{\theta_k}$

Policy gradient

- How to compute the policy gradient?

$$\begin{aligned} \nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau \sim \pi_\theta}{\mathrm{E}}[R(\tau)] \\ &= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) \\ &= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) \end{aligned}$$

Probability of a Trajectory

$$P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t) \pi_\theta(a_t|s_t)$$

Policy Gradient

- The Log-Derivative Trick $\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)$

$$\log P(\tau|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} \Bigg(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t) \Bigg)$$

$$\begin{aligned}\nabla_\theta \log P(\tau|\theta) &= \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T} \Bigg(\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t) \Bigg) \\ &= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t).\end{aligned}$$

Policy Gradient

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) && \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) && \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) && \text{Log-derivative trick} \\
&= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\nabla_\theta \log P(\tau|\theta)R(\tau)] && \text{Return to expectation form}
\end{aligned}
$$

$$
\nabla_\theta J(\pi_\theta) = \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] \quad \text{Expression for grad-log-prob}
$$

Policy Gradient

- Collect a set of trajectories using the policy π_θ

$$\mathcal{D} = \{\tau_i\}_{i=1,\dots,N}$$

- Estimate policy gradient $$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)$$

Categorical policy for discrete actions $$\log \pi_\theta(a|s) = \log \left[P_\theta(s)\right]_a$$

Diagonal Gaussian policy $$\log \pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2\log\sigma_i\right) + k\log 2\pi\right)$$

Policy Gradient

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

Agents should really only reinforce actions on the basis of their *consequences*.

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})\right]$$

$$\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

reward-to-go

Vanilla Policy Gradient

- Key idea: push up the probabilities of actions that lead to higher return, and push down probabilities of actions that lead to lower return
- The expected finite-horizon undiscounted return of the policy $J(\pi_\theta)$

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]$$

Advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$

Stochastic gradient ascent $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k})$

Vanilla Policy Gradient

Algorithm 1 Vanilla Policy Gradient Algorithm

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0

2: **for** $k = 0, 1, 2, \ldots$ **do**

3: Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

4: Compute rewards-to-go $\hat{R}_t$.

5: Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

6: Estimate policy gradient as

$$\hat{g}_k = \frac{1}{|\mathcal{D}_k|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)|_{\theta_k} \hat{A}_t.$$

7: Compute policy update, either using standard gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha_k \hat{g}_k,$$

or via another gradient ascent algorithm like Adam.

8: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_\phi \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

9: **end for**

reward-to-go

$$\hat{R}_t \doteq \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1})$$

Advantage function

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

$$= r + V^\pi(s') - V^\pi(s)$$

Trust Region Policy Optimization (TRPO)

- TRPO update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta)$$
$$\text{s.t. } \bar{D}_{KL}(\theta || \theta_k) \leq \delta$$

taking the largest step possible to improve performance

- *surrogate advantage*

$$\mathcal{L}(\theta_k, \theta) = \mathop{\mathrm{E}}_{s,a \sim \pi_{\theta_k}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) \right]$$

A measure of how the policy performs related to the old policy

- *KL-divergence*

$$\bar{D}_{KL}(\theta || \theta_k) = \mathop{\mathrm{E}}_{s \sim \pi_{\theta_k}} \left[D_{KL}\left(\pi_\theta(\cdot|s) || \pi_{\theta_k}(\cdot|s) \right) \right]$$

- *Approximation*

$$\mathcal{L}(\theta_k, \theta) \approx g^T(\theta - \theta_k)$$
$$\bar{D}_{KL}(\theta || \theta_k) \approx \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k)$$

$$\theta_{k+1} = \arg\max_{\theta} g^T(\theta - \theta_k)$$
$$\text{s.t. } \frac{1}{2}(\theta - \theta_k)^T H (\theta - \theta_k) \leq \delta.$$

$$\theta_{k+1} = \theta_k + \sqrt{\frac{2\delta}{g^T H^{-1} g}} H^{-1} g$$

Proximal Policy Optimization (PPO)

- PPO-clip updates

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta) \right]$$

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; \operatorname{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon \right) A^{\pi_{\theta_k}}(s,a) \right)$$

Avoid stepping so far that we accidentally cause performance collapse

PPO methods are significantly simpler to implement, and empirically seem to perform at least as well as TRPO

- A simpler version

$$L(s,a,\theta_k,\theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a), \;\; g(\epsilon, A^{\pi_{\theta_k}}(s,a)) \right)$$

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0. \end{cases}$$

Proximal Policy Optimization (PPO)

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

Deep Deterministic Policy Gradient (DDPG)

- DDPG currently learns a Q-function and a policy
  - Uses off-policy data and the Bellman equation to learn the Q-function
  - Uses the Q-function to learn the policy
- Q-learning $$Q^*(s,a) = \mathop{\mathrm{E}}_{s' \sim P}\left[r(s,a) + \gamma \max_{a'} Q^*(s',a')\right]$$

Approximator $Q_\phi(s,a)$ Collect a set of transitions (s,a,r,s',d)

mean-squared Bellman error (MSBE)

$$L(\phi, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d) \sim \mathcal{D}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)\max_{a'} Q_\phi(s',a')\right)\right)^2\right]$$

$\max_a Q(s,a) \approx Q(s,\mu(s))$ a policy $\mu(s)$

Deep Deterministic Policy Gradient (DDPG)

- Trick one: replay buffers
  - Large enough to contain a wide range of experiences
- Trick two: target networks
  - The term is called target $r + \gamma(1-d)\max_{a'} Q_{\phi}(s', a')$
  - The target depends on the same parameters ϕ but with a time delay
  - Target network ϕ_{targ}

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

  - Target policy network $\mu_{\theta_{\text{targ}}}$

Deep Deterministic Policy Gradient (DDPG)

- Q-learning in DDPG

$$L(\phi, \mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}}\left[\left(Q_\phi(s,a) - \left(r + \gamma(1-d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))\right)\right)^2\right]$$

- Policy learning in DDPG

$$\max_\theta \underset{s\sim\mathcal{D}}{\mathrm{E}}\left[Q_\phi(s, \mu_\theta(s))\right]$$

Deep Deterministic Policy Gradient (DDPG)

Algorithm 1 Deep Deterministic Policy Gradient

```
Input: initial policy parameters θ, Q-function parameters φ, empty replay buffer D
Set target parameters equal to main parameters θ_targ ← θ, φ_targ ← φ
repeat
  Observe state s and select action a = clip(μ_θ(s) + ε, a_Low, a_High), where ε ~ 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 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 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
```

Twin Delayed DDPG (TD3)

- Trick one: clipped double-Q learning
  - TD3 learns two Q functions
  - uses the smaller of the two Q-values to form the targets in the Bellman error loss functions

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

- Trick two: “delayed” policy updates
  - Updates the policy (and target networks) less frequently than the Q-function
- Trick three: target policy smoothing
  - 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

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

Twin Delayed DDPG (TD3)

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

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

      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

Soft Actor-Critic (SAC)

- An algorithm that optimizes a stochastic policy in an off-policy way
- Entropy-regularized RL

$$\pi^* = \arg\max_{\pi} \mathop{\mathrm{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big) \right]$$

Entropy $H(P) = \mathop{\mathrm{E}}_{x \sim P} [-\log P(x)]$

increasing entropy results in more exploration, which can accelerate learning later on

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{\tau \sim \pi} \left[\left. \sum_{t=0}^{\infty} \gamma^t \Big(R(s_t, a_t, s_{t+1}) + \alpha H\left(\pi(\cdot|s_t)\right) \Big) \right| s_0 = s \right]$$

$$V^{\pi}(s) = \mathop{\mathrm{E}}_{a \sim \pi} [Q^{\pi}(s,a)] + \alpha H\left(\pi(\cdot|s)\right)$$

$$Q^{\pi}(s,a) = \mathop{\mathrm{E}}_{\tau \sim \pi} \left[\left. \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t, s_{t+1}) + \alpha \sum_{t=1}^{\infty} \gamma^t H\left(\pi(\cdot|s_t)\right) \right| s_0 = s, a_0 = a \right]$$

Soft Actor-Critic (SAC)

- SAC learns a policy and two Q-functions
  - Uses entropy regularization
  - Train a stochastic policy

$$
\begin{aligned}
Q^{\pi}(s,a) &= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') + \alpha H\left(\pi(\cdot|s')\right) \right) \right] \\
&= \underset{\substack{s' \sim P \\ a' \sim \pi}}{\mathrm{E}} \left[R(s,a,s') + \gamma \left(Q^{\pi}(s',a') - \alpha \log \pi(a'|s') \right) \right]
\end{aligned}
$$

Approximate expectation with samples $Q^{\pi}(s,a) \approx r + \gamma \left(Q^{\pi}(s',\tilde{a}') - \alpha \log \pi(\tilde{a}'|s') \right), \quad \tilde{a}' \sim \pi(\cdot|s')$

Soft Actor-Critic (SAC)

- Q-learning

$$L(\phi_i, \mathcal{D}) = \mathop{\mathrm{E}}_{(s,a,r,s',d)\sim\mathcal{D}} \left[\left(Q_{\phi_i}(s,a) - y(r,s',d) \right)^2 \right]$$

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

- Policy learning

maximize

$$\begin{aligned} V^\pi(s) &= \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^\pi(s,a)\right] + \alpha H\left(\pi(\cdot|s)\right) \\ &= \mathop{\mathrm{E}}_{a\sim\pi}\left[Q^\pi(s,a) - \alpha \log \pi(a|s)\right] \end{aligned}$$

reparameterization trick

$$\tilde{a}_\theta(s,\xi) = \tanh\left(\mu_\theta(s) + \sigma_\theta(s) \odot \xi\right), \qquad \xi \sim \mathcal{N}(0, I)$$

Soft Actor-Critic (SAC)

- Policy learning

$$\mathop{\mathrm{E}}_{a\sim\pi_\theta}\left[Q^{\pi_\theta}(s,a) - \alpha \log \pi_\theta(a|s)\right] = \mathop{\mathrm{E}}_{\xi\sim\mathcal{N}}\left[Q^{\pi_\theta}(s,\tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right]$$

$$\max_\theta \mathop{\mathrm{E}}_{\substack{s\sim\mathcal{D}\\ \xi\sim\mathcal{N}}}\left[\min_{j=1,2} Q_{\phi_j}(s,\tilde{a}_\theta(s,\xi)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi)|s)\right]$$

Soft Actor-Critic (SAC)

Algorithm 1 Soft Actor-Critic

Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$

Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$

repeat

Observe state s and select action $a \sim \pi_\theta(\cdot|s)$

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 targets for the Q functions:

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

Update Q-functions by one step of gradient descent using

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

Update policy by one step of gradient ascent using

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

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

Update target networks with

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

end for

end if

until convergence

Summary

- Vanilla Policy Gradient
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)
- Deep Deterministic Policy Gradient (DDPG)
- Twin Delayed DDPG (TD3)
- Soft Actor-Critic (SAC)

Further Reading

- OpenAI Spinning Up in Deep RL
  https://spinningup.openai.com/en/latest/index.html