Advanced Actor-Critic Methods (DPG, DDPG, Importance Sampling)

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Fall 2019

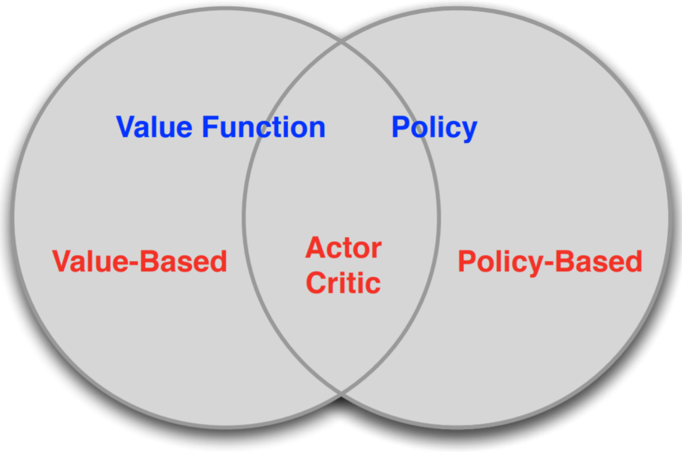
avereshc@buffalo.edu

November 5, 2019

*Slides are adopted from Deep Reinforcement Learning by Sergey Levine & Policy Gradients Algorithms by Lilian Weng

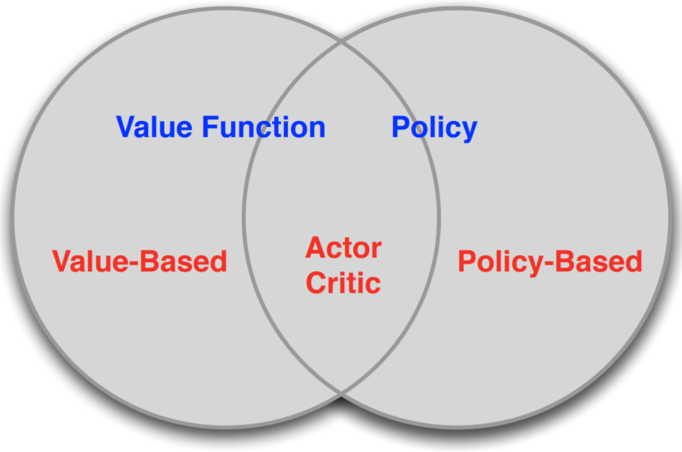
Table of Contents

Value Based and Policy-Based RL



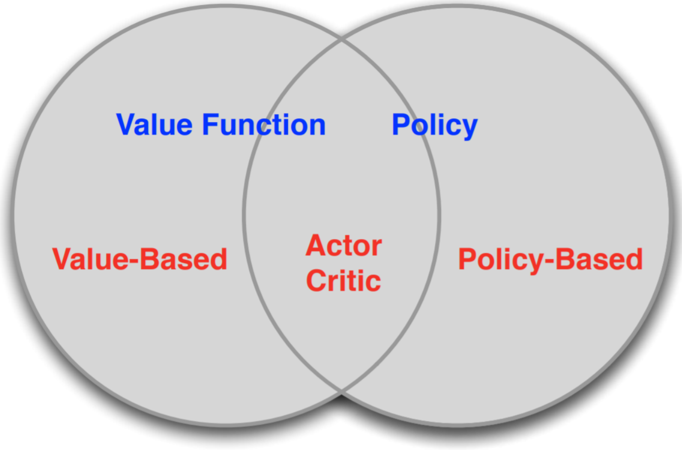
- Value Based

Value Based and Policy-Based RL



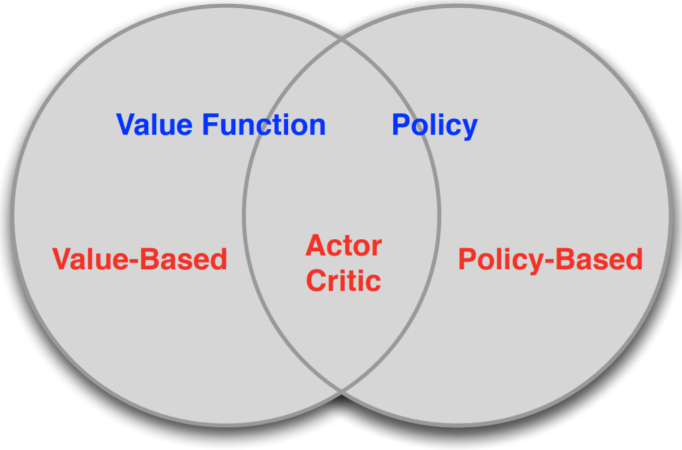
- Value Based
  - Learn Value Function
  - Implicit policy

Value Based and Policy-Based RL



- Value Based
  - Learn Value Function
  - Implicit policy
- Policy Based
  - No Value Function
  - Learn Policy

Value Based and Policy-Based RL



- Value Based
  - Learn Value Function
  - Implicit policy
- Policy Based
  - No Value Function
  - Learn Policy
- Actor-Critic
  - Learn Value Function
  - Learn Policy

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic Updates action-value function parameters w

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic

Actor-Critic

- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s,a) \approx Q_{\pi_\theta}(s,a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$$

Actor-Critic

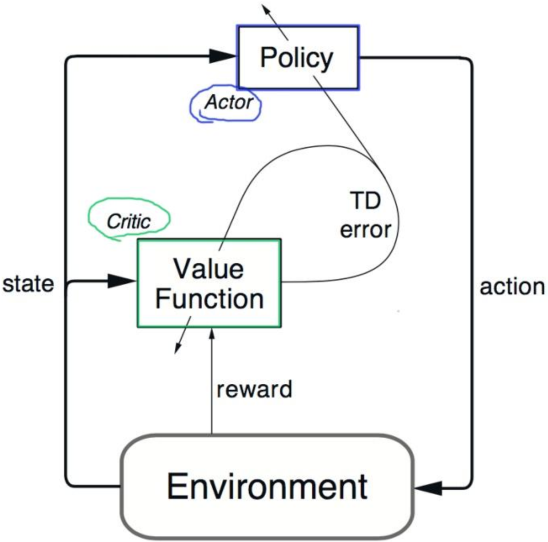
- Monte-Carlo policy gradient still has high variance
- We can use a critic to estimate the action-value function:

$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$

- Actor-critic algorithms maintain *two* sets of parameters
  - Critic Updates action-value function parameters w
  - Actor Updates policy parameters θ, in direction suggested by critic
- Actor-critic algorithms follow an approximate policy gradient

$$\nabla_\theta J(\theta) \approx E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)]$$
$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$$

Actor-Critic



Policy gradient methods maximize the expected total reward by repeatedly estimating the gradient $g := \nabla_\theta \mathbb{E}\left[\sum_{t=0}^{\infty} r_t\right]$. There are several different related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)\right], \tag{1}$$

where Ψ_t may be one of the following:

1. $\sum_{t=0}^{\infty} r_t$: total reward of the trajectory.
2. $\sum_{t'=t}^{\infty} r_{t'}$: reward following action a_t.
3. $\sum_{t'=t}^{\infty} r_{t'} - b(s_t)$: baselined version of previous formula.
4. $Q^\pi(s_t, a_t)$: state-action value function.
5. $A^\pi(s_t, a_t)$: advantage function.
6. $r_t + V^\pi(s_{t+1}) - V^\pi(s_t)$: TD residual.

The latter formulas use the definitions

$$V^\pi(s_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \qquad Q^\pi(s_t, a_t) := \mathbb{E}_{\substack{s_{t+1:\infty}, \\ a_{t+1:\infty}}}\left[\sum_{l=0}^{\infty} r_{t+l}\right] \tag{2}$$

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t), \quad \text{(Advantage function).} \tag{3}$$

*Schulman, John, et al. "High-dimensional continuous control using generalized advantage estimation."

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) G_t]$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] \\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]\end{aligned}$$

REINFORCE

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)]
\end{aligned}$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t] && \text{REINFORCE}\\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)] && \text{Q Actor-Critic}\\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)] && \text{Advantage Actor-Critic (A2C)}\\ &= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta]\end{aligned}$$

Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) G_t]$	REINFORCE
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)]$	Q Actor-Critic
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) A_w(s,a)]$	Advantage Actor-Critic (A2C)
$= \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) \delta]$	TD Actor-Critic

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q_\pi(s,a)$, $A_\pi(s,a)$ or $V_\pi(s)$.

Table of Contents

Recap: Policies

- Stochastic policy is defined as probability distribution over actions A

$$\pi(.|s)$$

Recap: Policies

- Stochastic policy is defined as probability distribution over actions A

$$\pi(.|s)$$

- Deterministic policy gradient (DPG) instead models the policy as a deterministic decision:

$$a = \mu(s)$$

Deterministic Policy Gradient: Notations [1]

- $\rho_0(s)$:

[1] Deterministic Policy Gradient Algorithms by David Silver et. al. 2014

Deterministic Policy Gradient: Notations [1]

- $\rho_0(s)$: The initial distribution over states

[1] Deterministic Policy Gradient Algorithms by David Silver et. al. 2014

Deterministic Policy Gradient: Notations [1]

- $\rho_0(s)$: The initial distribution over states
- $\rho^{\mu}(s \to s', k)$:

[1] Deterministic Policy Gradient Algorithms by David Silver et. al. 2014

Deterministic Policy Gradient: Notations [1]

- $\rho_0(s)$: The initial distribution over states
- $\rho^\mu(s \to s', k)$: Starting from state s, the visitation probability density at state s' after moving k steps by policy μ

[1] Deterministic Policy Gradient Algorithms by David Silver et. al. 2014

Deterministic Policy Gradient: Notations [1]

- $\rho_0(s)$: The initial distribution over states
- $\rho^\mu(s \to s', k)$: Starting from state s, the visitation probability density at state s' after moving k steps by policy μ
- $\rho^\mu(s')$: Discounted state distribution, defined as

$$\rho^\mu(s') = \int_{\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\mu(s \to s', k) ds$$

[1] Deterministic Policy Gradient Algorithms by David Silver et. al. 2014

Deterministic Policy Gradient: Notations [1]

- $\rho_0(s)$: The initial distribution over states
- $\rho^\mu(s \to s', k)$: Starting from state s, the visitation probability density at state s' after moving k steps by policy μ
- $\rho^\mu(s')$: Discounted state distribution, defined as

$$\rho^\mu(s') = \int_{\mathcal{S}} \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\mu(s \to s', k) ds$$

- The objective function to optimize is

$$J(\theta) = \int_{\mathcal{S}} \rho^\mu(s) Q(s, \mu_\theta(s)) ds$$

[1] Deterministic Policy Gradient Algorithms by David Silver et. al. 2014

Deterministic Policy Gradient (DPG)

Let's consider an example of on-policy actor-critic algorithm. In each iteration of on-policy actor-critic, two actions are taken deterministically $a = \mu_\theta(s)$ and the SARSA update on policy parameters relies on the new gradient that we just computed above:

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t) \qquad \text{; TD error in SARSA}$$

Deterministic Policy Gradient (DPG)

Let's consider an example of on-policy actor-critic algorithm. In each iteration of on-policy actor-critic, two actions are taken deterministically $a = \mu_\theta(s)$ and the SARSA update on policy parameters relies on the new gradient that we just computed above:

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t) \quad \text{; TD error in SARSA}$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t)$$

Deterministic Policy Gradient (DPG)

Let's consider an example of on-policy actor-critic algorithm. In each iteration of on-policy actor-critic, two actions are taken deterministically $a = \mu_\theta(s)$ and the SARSA update on policy parameters relies on the new gradient that we just computed above:

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t) \quad \text{; TD error in SARSA}$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t)$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_a Q_w(s_t, a_t) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)} \quad \text{; Deterministic policy gradient theorem}$$

Deterministic Policy Gradient (DPG)

However, unless there is sufficient noise in the environment, it is very hard to guarantee enough exploration due to the determinacy of the policy.

- We can either add noise into the policy (ironically this makes it nondeterministic!)
- Learn it off-policy-ly by following a different stochastic behavior policy to collect samples

Deterministic Policy Gradient (DPG)

Say, in the off-policy approach, the training trajectories are generated by a stochastic policy $\beta(a|s)$ and thus the state distribution follows the corresponding discounted state density ρ^β:

$$J_\beta(\theta) = \int_{\mathcal{S}} \rho^\beta Q^\mu(s, \mu_\theta(s)) ds$$
$$\nabla_\theta J_\beta(\theta) = \mathbb{E}_{s\sim\rho^\beta}[\nabla_a Q^\mu(s,a) \nabla_\theta \mu_\theta(s)|_{a=\mu_\theta(s)}]$$

Note that because the policy is deterministic, we only need $Q^\mu(s, \mu_\theta(s))$ rather than $\sum_a \pi(a|s) Q^\pi(s,a)$ as the estimated reward of a given state s.

Table of Contents

Deep Deterministic Policy Gradient (DDPG) [2]

- Deep Deterministic Policy Gradient (Lillicrap, et al., 2015) (DDPG) is an algorithm which concurrently learns a Q-function and a policy
- It is a model-free off-policy actor-critic algorithm, combining DPG with DQN

[2] Continuous Control With Deep Reinforcement Learning by Lillicrap et al, 2015

Deep Deterministic Policy Gradient (DDPG) [2]

- Deep Deterministic Policy Gradient (Lillicrap, et al., 2015) (DDPG) is an algorithm which concurrently learns a Q-function and a policy
- It is a model-free off-policy actor-critic algorithm, combining DPG with DQN
- Recall: How DQN stabilizes the learning of Q-function?

[2] Continuous Control With Deep Reinforcement Learning by Lillicrap et al, 2015

Deep Deterministic Policy Gradient (DDPG) [2]

- Deep Deterministic Policy Gradient (Lillicrap, et al., 2015) (DDPG) is an algorithm which concurrently learns a Q-function and a policy
- It is a model-free off-policy actor-critic algorithm, combining DPG with DQN
- Recall: How DQN stabilizes the learning of Q-function?
- By experience replay and the frozen target network

[2] Continuous Control With Deep Reinforcement Learning by Lillicrap et al, 2015

Deep Deterministic Policy Gradient (DDPG) [2]

- Deep Deterministic Policy Gradient (Lillicrap, et al., 2015) (DDPG) is an algorithm which concurrently learns a Q-function and a policy
- It is a model-free off-policy actor-critic algorithm, combining DPG with DQN
- Recall: How DQN stabilizes the learning of Q-function?
- By experience replay and the frozen target network
- Is DQN works in discrete or continuous space?

[2] Continuous Control With Deep Reinforcement Learning by Lillicrap et al, 2015

Deep Deterministic Policy Gradient (DDPG) [2]

- Deep Deterministic Policy Gradient (Lillicrap, et al., 2015) (DDPG) is an algorithm which concurrently learns a Q-function and a policy
- It is a model-free off-policy actor-critic algorithm, combining DPG with DQN
- Recall: How DQN stabilizes the learning of Q-function?
- By experience replay and the frozen target network
- Is DQN works in discrete or continuous space?
- The original DQN works in discrete space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy

[2] Continuous Control With Deep Reinforcement Learning by Lillicrap et al, 2015

Recap

- Optimal action in DQN

Recap

- Optimal action in DQN

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

Recap

- Optimal action in DQN

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

- Not practical for continuous action space
- Using deterministic policy, we can use:

$$a = \mu(s|\theta^\mu)$$

Recap

- Optimal action in DQN

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

- Not practical for continuous action space
- Using deterministic policy, we can use:

$$a = \mu(s|\theta^\mu)$$

- But deterministic policy gradient might not explore the full state and action space. To overcome this, we are adding noise $\mathcal{N}$

Recap

- Optimal action in DQN

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

- Not practical for continuous action space
- Using deterministic policy, we can use:

$$a = \mu(s|\theta^\mu)$$

- But deterministic policy gradient might not explore the full state and action space. To overcome this, we are adding noise $\mathcal{N}$

$$a = \mu(s|\theta^\mu) + \mathcal{N}$$

- Recall: How do we explore in DQN?

Recap

- Optimal action in DQN

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

- Not practical for continuous action space
- Using deterministic policy, we can use:

$$a = \mu(s|\theta^\mu)$$

- But deterministic policy gradient might not explore the full state and action space. To overcome this, we are adding noise $\mathcal{N}$

$$a = \mu(s|\theta^\mu) + \mathcal{N}$$

- Recall: How do we explore in DQN?
- In DQN we use ϵ-greedy approach to ensure exploration

Deep Deterministic Policy Gradient (DDPG)

DDPG does soft updates ("conservative policy iteration") on the parameters of both actor and critic

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

Deep Deterministic Policy Gradient (DDPG)

DDPG does soft updates (“conservative policy iteration”) on the parameters of both actor and critic

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

In this way, the target network values are constrained to change slowly, different from the design in DQN that the target network stays frozen for some period of time.

DDPG: Parameters

θ^Q: Q network

$\theta^{Q'}$: Target Q network

θ^μ: Deterministic policy function

$\theta^{\mu'}$: Target policy network

Deep Deterministic Policy Gradient (DDPG)

Actor directly maps states to actions instead of outputting the probability distribution across a discrete action space

Deep Deterministic Policy Gradient (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 $\boxed{a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t}$ 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 $\boxed{y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})}$
    Update $\boxed{\text{critic by minimizing the loss: } L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2}$
    Update the actor policy using the sampled policy gradient:

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

    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

- DDPG is an algorithm which concurrently learns a Q-function and a policy

Deep Deterministic Policy Gradient (DDPG)

- DDPG is an algorithm which concurrently learns a Q-function and a policy
- DDPG is an off-policy algorithm

Deep Deterministic Policy Gradient (DDPG)

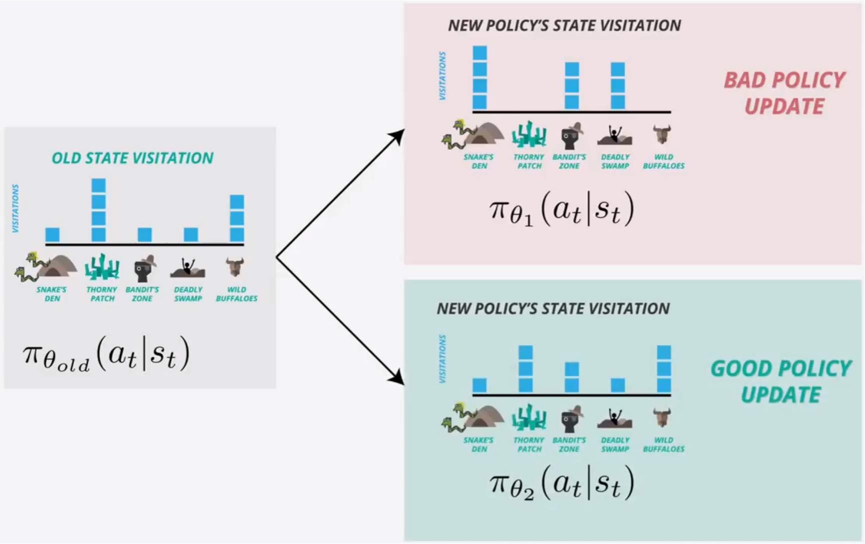
- DDPG is an algorithm which concurrently learns a Q-function and a policy
- DDPG is an off-policy algorithm
- DDPG can only be used for environments with continuous action spaces

Deep Deterministic Policy Gradient (DDPG)

- DDPG is an algorithm which concurrently learns a Q-function and a policy
- DDPG is an off-policy algorithm
- DDPG can only be used for environments with continuous action spaces
- DDPG can be thought of as being deep Q-learning for continuous action spaces

Table of Contents

Problems in Policy Gradient



On-policy Sampling

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

$$\nabla_\theta J(\theta) = \underline{E_{\tau\sim\pi_\theta(\tau)}}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$$

this is trouble...

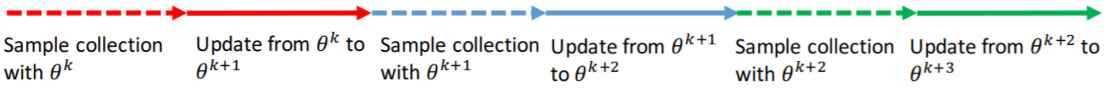
- Neural networks change only a little bit with each gradient step
- On-policy learning can be extremely inefficient!

can't just skip this!

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run it on the robot)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_t \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i)\right)\left(\sum_t r(\mathbf{s}_t^i, \mathbf{a}_t^i)\right)$
3. $\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta)$

On-policy Sampling



Off-policy Sampling

$$\theta^\star = \arg\max_\theta J(\theta)$$

$$J(\theta) = E_{\tau\sim\pi_\theta(\tau)}[r(\tau)]$$

what if we don't have samples from $\pi_\theta(\tau)$?

(we have samples from some $\bar\pi(\tau)$ instead)

$$J(\theta) = E_{\tau\sim\bar\pi(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar\pi(\tau)} r(\tau)\right]$$

$$\pi_\theta(\tau) = p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)$$

$$\frac{\pi_\theta(\tau)}{\bar\pi(\tau)} = \frac{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}}{\cancel{p(\mathbf{s}_1)}\prod_{t=1}^{T}\bar\pi(\mathbf{a}_t|\mathbf{s}_t)\cancel{p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar\pi(\mathbf{a}_t|\mathbf{s}_t)}$$

importance sampling

$$\begin{aligned} E_{x\sim p(x)}[f(x)] &= \int p(x)f(x)dx \\ &= \int \frac{q(x)}{q(x)}p(x)f(x)dx \\ &= \int q(x)\frac{p(x)}{q(x)}f(x)dx \\ &= E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right] \end{aligned}$$

Off-Policy Sampling