Advanced Actor-Critic Methods (A3C, DPG, DDPG, Importance Sampling)

Alina Vereshchaka

CSE4/510 Reinforcement Learning
Spring 2020
avereshc@buffalo.edu

April 13, 2020

*Slides are adopted from Deep Reinforcement Learning by Sergey Levine & Policy Gradients Algorithms by Lilian Weng
1 Recap: Actor-Critic

2 Asynchronous Advantage Actor Critic (A3C)

3 Deterministic Policy Gradient (DPG)

4 Deep Deterministic Policy Gradient (DDPG)
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
Value Based and Policy-Based RL

- **Value Based**
  - Learn Value Function
  - Implicit policy

- **Policy Based**
  - No Value Function
  - Learn Policy

Value-Based

Policy-Based

Value Function

Policy

Actor Critic
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 \( \theta \), 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 \( \theta \), 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 \( \theta \), in direction suggested by critic

- Actor-critic algorithms follow an approximate policy gradient

\[
\nabla_{\theta} J(\theta) \approx E_{\pi_{\theta}} \left[ \nabla_{\theta} \log \pi_{\theta}(s, a) Q_w(s, a) \right] \\
\Delta \theta = \alpha \nabla_{\theta} \log \pi_{\theta}(s, a) Q_w(s, a)
\]
Actor-Critic

![Diagram of Actor-Critic model]

- **Policy**
- **Actor**: Takes state as input and outputs an action.
- **Critic**: Receives the state and action as input and outputs a value function.
- **Value Function**: Computes the value of the state-action pair.
- **Environment**: Receives the action and produces the next state and reward.
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

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

where \( \Psi_t \) 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

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

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

The policy gradient has many equivalent forms

\[ \nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} [\nabla_\theta \log \pi_\theta(s, a) G_t] \]
The policy gradient has many equivalent forms

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

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

REINFORCE
Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

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

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

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

REINFORCE
Q Actor-Critic
Actor-Critic (A2C)
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 \( V_\pi(s) \).
Summary of Policy Gradient Algorithms

- The **policy gradient** has many equivalent forms

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

- **REINFORCE**
- Q Actor-Critic
- Advantage Actor-Critic (A2C)
Summary of Policy Gradient Algorithms

- The policy gradient has many equivalent forms

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

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

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

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

- REINFORCE  
- Q Actor-Critic  
- Advantage Actor-Critic (A2C)  
- 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 \( V_\pi(s) \).
Recap: Actor-Critic: Critic (Linear TD(0)) + Actor (Policy Gradient)

One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

- **Input:** a differentiable policy parameterization $\pi(a|s, \theta)$
- **Input:** a differentiable state-value function parameterization $\hat{v}(s, w)$
- **Parameters:** step sizes $\alpha^\theta > 0$, $\alpha^w > 0$
- **Initialize policy parameter** $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^{d}$ (e.g., to 0)
- **Loop forever** (for each episode):
  - Initialize $S$ (first state of episode)
  - $I \leftarrow 1$
  - Loop while $S$ is not terminal (for each time step):
    - $A \sim \pi(\cdot|S, \theta)$
    - Take action $A$, observe $S', R$
    - $\delta \leftarrow R + \gamma \hat{v}(S', w) - \hat{v}(S, w)$  
      (if $S'$ is terminal, then $\hat{v}(S', w) \equiv 0$)
    - $w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S, w)$
    - $\theta \leftarrow \theta + \alpha^\theta I \delta \nabla \ln \pi(A|S, \theta)$
    - $I \leftarrow \gamma I$
    - $S \leftarrow S'$
Recap: REINFORCE with Baseline

REINFORCE with Baseline (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s, \theta)$
Input: a differentiable state-value function parameterization $\hat{v}(s,w)$
Algorithm parameters: step sizes $\alpha^\theta > 0$, $\alpha^w > 0$
Initialize policy parameter $\theta \in \mathbb{R}^{d'}$ and state-value weights $w \in \mathbb{R}^d$ (e.g., to 0)

Loop forever (for each episode):
Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$, following $\pi(\cdot|\cdot, \theta)$
Loop for each step of the episode $t = 0, 1, \ldots, T - 1$:
\[ G \leftarrow \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \]  
\[ \delta \leftarrow G - \hat{v}(S_t, w) \]  
\[ w \leftarrow w + \alpha^w \delta \nabla \hat{v}(S_t, w) \]  
\[ \theta \leftarrow \theta + \alpha^\theta \gamma^t \delta \nabla \ln \pi(A_t|S_t, \theta) \]
Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient
Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating both $V_{\pi_\theta}(s)$ and $Q_{\pi_\theta}(s, a)$
Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating both $V_{\pi_\theta}(s)$ and $Q_{\pi_\theta}(s, a)$
- Using two function approximators and two parameter vectors,

$$V_v(s) \approx V_{\pi_\theta}(s)$$
Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating both $V_{\pi_\theta}(s)$ and $Q_{\pi_\theta}(s, a)$
- Using two function approximators and two parameter vectors,

$$V_v(s) \approx V_{\pi_\theta}(s)$$
$$Q_w(s, a) \approx Q_{\pi_\theta}(s, a)$$
Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating both $V_{\pi\theta}(s)$ and $Q_{\pi\theta}(s, a)$
- Using two function approximators and two parameter vectors,

\[
V_{\nu}(s) \approx V_{\pi\theta}(s) \\
Q_{w}(s, a) \approx Q_{\pi\theta}(s, a) \\
A(s, a) = Q_{w}(s, a) - V_{\nu}(s)
\]
Advantage Actor Critic (A2C)

- The advantage function can significantly reduce variance of policy gradient
- So the critic should really estimate the advantage function
- For example, by estimating both $V_{\pi\theta}(s)$ and $Q_{\pi\theta}(s, a)$
- Using two function approximators and two parameter vectors,

$V_v(s) \approx V_{\pi\theta}(s)$
$Q_w(s, a) \approx Q_{\pi\theta}(s, a)$

$A(s, a) = Q_w(s, a) - V_v(s)$

- And updating both value functions by e.g. TD learning
Table of Contents

1 Recap: Actor-Critic

2 Asynchronous Advantage Actor Critic (A3C)

3 Deterministic Policy Gradient (DPG)

4 Deep Deterministic Policy Gradient (DDPG)
A3C: Asynchronous Advantage Actor Critic

- **Asynchronous**: the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure. No experience replay is needed, though one could add it if desired.
A3C: Asynchronous Advantage Actor Critic

- **Asynchronous**: the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure. No experience replay is needed, though one could add it if desired.

- **Advantage**: the policy gradient updates are done using the advantage function $A(s, a)$
A3C: Asynchronous Advantage Actor Critic

- **Asynchronous**: the algorithm involves executing a set of environments in parallel to increase the diversity of training data, and with gradient updates performed in a Hogwild! style procedure. No experience replay is needed, though one could add it if desired.

- **Advantage**: the policy gradient updates are done using the advantage function $A(s, a)$

- **Actor**: this is an actor-critic method which involves a policy that updates with the help of learned state-value functions.
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent.
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
  - This parallelism decorrelates the agents’ data, so no Experience Replay Buffer needed
  - Even one can explicitly use different exploration policies in each actor-learner to maximize diversity
  - Asynchronism can be extended to other update mechanisms (SARSA, Q-learning, etc) but it works better in Advantage Actor critic setting
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - Use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents

- Asynchronism can be extended to other update mechanisms (SARSA, Q-learning, etc) but it works better in Advantage Actor critic setting
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents’ data, so no Experience Replay Buffer needed
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents’ data, so no Experience Replay Buffer needed
- Even one can explicitly use different exploration policies in each actor-learner to maximize diversity
Asynchronous Advantage Actor Critic (A3C)

- A3C (Mnih et al. 2016) idea: Sample for data can be parallelized using several copies of the same agent
  - use N copies of the agents (workers) working in parallel collecting samples and computing gradients for policy and value function
  - After some time, pass gradients to a main network that updates actor and critic using the gradients of all agents
  - After some time the worker copy the weights of the global network
- This parallelism decorrelates the agents’ data, so no Experience Replay Buffer needed
- Even one can explicitly use different exploration policies in each actor-learner to maximize diversity
- Asynchronism can be extended to other update mechanisms (SARSA, Q-learning, etc) but it works better in Advantage Actor critic setting
Asynchronous Advantage Actor Critic (A3C)
Asynchronous Advantage Actor Critic (A3C)

1. Worker resets to global network
2. Worker interacts with environment
3. Worker calculates value and policy loss
4. Worker gets gradients from losses
5. Worker updates global network with gradients
Asynchronous Advantage Actor Critic (A3C)

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors $\theta$ and $\theta_v$ and global shared counter $T = 0$
// Assume thread-specific parameter vectors $\theta'$ and $\theta'_v$
Initialize thread step counter $t \leftarrow 1$

repeat
  Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
  Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$.
  $t_{start} = t$
  Get state $s_t$

  repeat
    Perform $a_t$ according to policy $\pi(a_t|s_t; \theta')$
    Receive reward $r_t$ and new state $s_{t+1}$
    $t \leftarrow t + 1$
    $T \leftarrow T + 1$
  until terminal $s_t$ or $t - t_{start} \geq t_{max}$

  $R = \begin{cases} 
  0 & \text{for terminal } s_t \\
  V(s_{t+1}, \theta'_v) & \text{for non-terminal } s_t \end{cases}$

  for $i \in \{t_{start}, \ldots, t_{max}\}$ do
    $R \leftarrow r_i + \gamma R$
    Accumulate gradients wrt $\theta'$: $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$
    Accumulate gradients wrt $\theta'_v$: $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$
  end for

  Perform asynchronous update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_v$.
  until $T > T_{max}$
Table of Contents

1 Recap: Actor-Critic

2 Asynchronous Advantage Actor Critic (A3C)

3 Deterministic Policy Gradient (DPG)

4 Deep Deterministic Policy Gradient (DDPG)
Recap: Policies

- Stochastic policy is defined as probability distribution over actions $A$:

  $$\pi(\cdot | s)$$
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

- $\rho_0(s)$:

  - The initial distribution over states $ho_0(s)$:
  - Starting from state $s$, the visitation probability density at state $s'$ after moving $k$ steps by policy $\mu$.

  - Discounted state distribution, defined as $\rho_\mu(s') = \int \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho_\mu(s\rightarrow s',k) \, ds$.

  - The objective function to optimize is $J(\theta) = \int \rho_\mu(s) Q(s,\mu_\theta(s)) \, ds$. 

---

1Deterministic Policy Gradient Algorithms by David Silver et. al. 2014
Deterministic Policy Gradient: Notations

- $\rho_0(s)$: The initial distribution over states
Deterministic Policy Gradient: Notations

- $\rho_0(s)$: The initial distribution over states
- $\rho^\mu(s \rightarrow s', k)$:
Deterministic Policy Gradient: Notations

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

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

$$
\rho^\mu(s') = \int_S \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\mu(s \rightarrow s', k) ds
$$
Deterministic Policy Gradient: Notations

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

$$\rho^\mu(s') = \int_S \sum_{k=1}^{\infty} \gamma^{k-1} \rho_0(s) \rho^\mu(s \rightarrow s', k) ds$$

- The objective function to optimize is

$$J(\theta) = \int_S \rho^\mu(s) Q(s, \mu_\theta(s)) ds$$

---

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

$$
\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)
$$

; TD error in SARSA
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:

$$\delta_t = R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t)$$

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

; TD error in SARSA
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:

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

\[
w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t) \quad ; \text{Deterministic policy gradient theorem}
\]

\[
\theta_{t+1} = \theta_t + \alpha_\theta \nabla_a Q_w(s_t, a_t) \nabla_\theta \mu_\theta(s) |_{a=\mu_\theta(s)}
\]
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
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 \( \rho^{\beta} \):

\[
J_{\beta}(\theta) = \int_{S} \rho^{\beta} Q^{\mu}(s, \mu_\theta(s)) ds
\]

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

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 \( s \).
1 Recap: Actor-Critic

2 Asynchronous Advantage Actor Critic (A3C)

3 Deterministic Policy Gradient (DPG)

4 Deep Deterministic Policy Gradient (DDPG)
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) 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?
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.

²Continuous Control With Deep Reinforcement Learning by Lillicrap et al, 2015
Deep Deterministic Policy Gradient (DDPG) (Lillicrap, et al., 2015) 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.

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