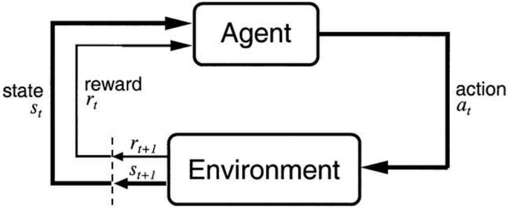
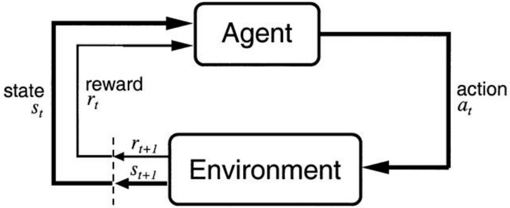
Reinforcement Learning
Policy Gradient Methods

Reinforcement Learning (RL) - Introduction



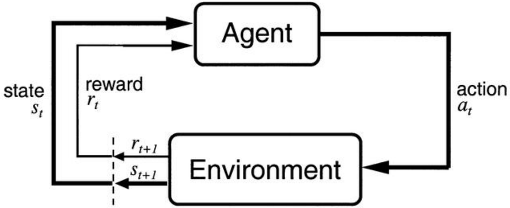
- RL methods apply to problems where an agent interacts with an environment in discrete time step.

Reinforcement Learning (RL) - Introduction



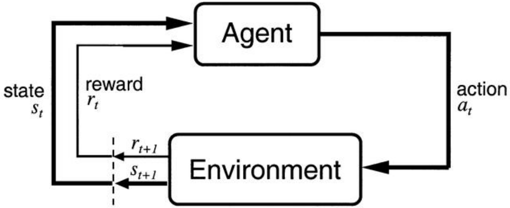
- RL methods apply to problems where an agent interacts with an environment in discrete time step.
- At time t, the agent is in state s_t, and performs an action a_t.

Reinforcement Learning (RL) - Introduction



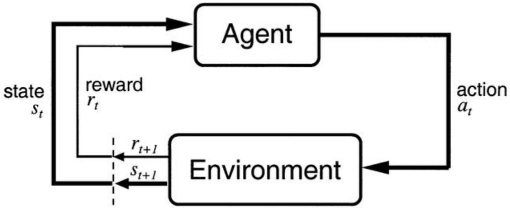
- RL methods apply to problems where an agent interacts with an environment in discrete time step.
- At time t, the agent is in state s_t, and performs an action a_t.
- At the next time step $t+1$, it arrives in next state s_{t+1} and obtain reward r_{t+1}.

Reinforcement Learning (RL) - Introduction



- RL methods apply to problems where an agent interacts with an environment in discrete time step.
- At time t, the agent is in state s_t, and performs an action a_t.
- At the next time step $t+1$, it arrives in next state s_{t+1} and obtain reward r_{t+1}.
- The goal of the agent is to maximize the reward obtained **on the long term**.

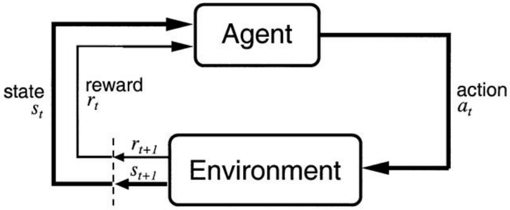
Markov Decision Process (MDP) - Introduction



RL problems are often described as MDPs:

- A state space $\mathcal{S}$.

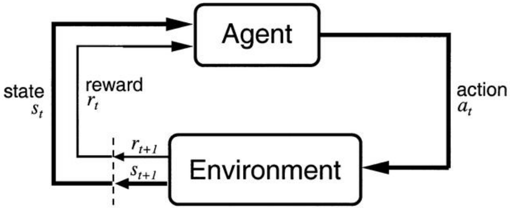
Markov Decision Process (MDP) - Introduction



RL problems are often described as MDPs:

- A state space $\mathcal{S}$.
- An action space $\mathcal{A}$ of actions a.

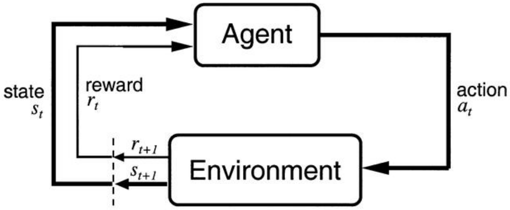
Markov Decision Process (MDP) - Introduction



RL problems are often described as MDPs:

- A state space $\mathcal{S}$.
- An action space $\mathcal{A}$ of actions a.
- A transition model $p(s'|s,a)$: the probability of arriving in state s' at time $t+1$ when being in state s and performing action a.

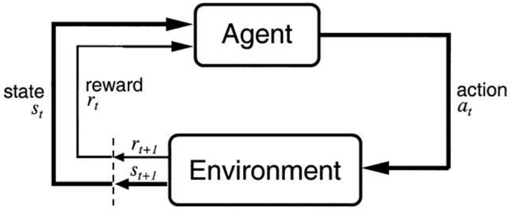
Markov Decision Process (MDP) - Introduction



RL problems are often described as MDPs:

- A state space $\mathcal{S}$.
- An action space $\mathcal{A}$ of actions a.
- A transition model $p(s'|s,a)$: the probability of arriving in state s' at time $t+1$ when being in state s and performing action a.
- A reward function $r(s,a,s')$ defines the (stochastic) reward obtained after performing a in state s and arriving in s'.

Markov Decision Process (MDP) - Introduction



RL problems are often described as MDPs:

- A state space $\mathcal{S}$.
- An action space $\mathcal{A}$ of actions a.
- A transition model $p(s'|s,a)$: the probability of arriving in state s' at time $t+1$ when being in state s and performing action a.
- A reward function $r(s,a,s')$ defines the (stochastic) reward obtained after performing a in state s and arriving in s'.
- An initial state distribution $p_0(s_0)$ from which states is the agent likely to start.

Markov Decision Process (MDP) - Introduction

- The behavior of the agent over time is a trajectory (also called episode, history, or roll-out)

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_H, a_H)$$

Markov Decision Process (MDP) - Introduction

- The behavior of the agent over time is a trajectory (also called episode, history, or roll-out)

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_H, a_H)$$

- Each transition occurs with a probability $p(s'|s, a)$ and provides a certain amount of reward $r(s, a, s')$.

Markov Decision Process (MDP) - Introduction

- The behavior of the agent over time is a trajectory (also called episode, history, or roll-out)

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_H, a_H)$$

- Each transition occurs with a probability $p(s'|s, a)$ and provides a certain amount of reward $r(s, a, s')$.
- In episodic tasks, the horizon H is finite, while in continuing tasks, H is infinite ∞.

Markov Decision Process (MDP) - Introduction

- The behavior of the agent over time is a trajectory (also called episode, history, or roll-out)

$$\tau = (s_0, a_0, s_1, a_1, \dots, s_H, a_H)$$

- Each transition occurs with a probability $p(s'|s,a)$ and provides a certain amount of reward $r(s,a,s')$.
- In episodic tasks, the horizon H is finite, while in continuing tasks, H is infinite ∞.
- The Markov property states that:

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots, s_0, a_0)$$

Markov Decision Process (MDP) - Introduction

- The behavior of the agent over time is a trajectory (also called episode, history, or roll-out)

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_H, a_H)$$

- Each transition occurs with a probability $p(s'|s,a)$ and provides a certain amount of reward $r(s,a,s')$.
- In episodic tasks, the horizon H is finite, while in continuing tasks, H is infinite ∞.
- The Markov property states that:

$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \ldots, s_0, a_0)$$

- N.B. We need to provide enough information to the description of a state. If a transition depends on what happened in the past, put that information in the state description.

Policy and value functions

- The policy defines the behavior of the agent: which action is performed in each state.

[1] Thomas Simonini - An introduction to Policy Gradients with Cartpole and Doom

Policy and value functions

- The policy defines the behavior of the agent: which action is performed in each state.
- A deterministic policy $\mu(s_t)$ is a mapping of: $\mathcal{S} \to \mathcal{A}$.

[1] Thomas Simonini - An introduction to Policy Gradients with Cartpole and Doom

Policy and value functions

- The policy defines the behavior of the agent: which action is performed in each state.
- A deterministic policy $\mu(s_t)$ is a mapping of: $\mathcal{S} \rightarrow \mathcal{A}$.
- A stochastic policy $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$ defines the probability distribution $P(\mathcal{A})$ of performing an action.

[1] Thomas Simonini - An introduction to Policy Gradients with Cartpole and Doom

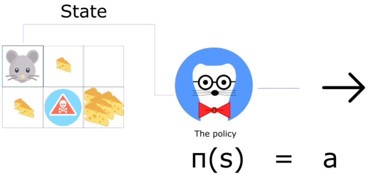
Policy and value functions

- The policy defines the behavior of the agent: which action is performed in each state.
- A deterministic policy $\mu(s_t)$ is a mapping of: $\mathcal{S} \to \mathcal{A}$.
- A stochastic policy $\pi : \mathcal{S} \to P(\mathcal{A})$ defines the probability distribution $P(\mathcal{A})$ of performing an action.

[1] Thomas Simonini - An introduction to Policy Gradients with Cartpole and Doom

Policy and value functions

- The policy defines the behavior of the agent: which action is performed in each state.
- A deterministic policy $\mu(s_t)$ is a mapping of: $\mathcal{S} \to \mathcal{A}$.
- A stochastic policy $\pi : \mathcal{S} \to P(\mathcal{A})$ defines the probability distribution $P(\mathcal{A})$ of performing an action.

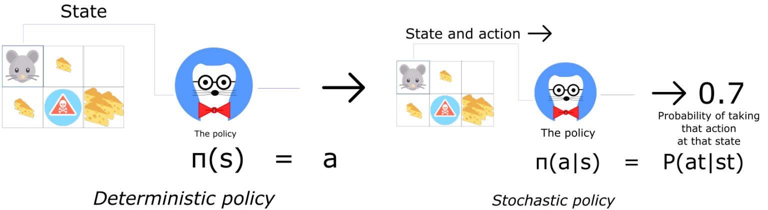


Deterministic policy

[1] Thomas Simonini - An introduction to Policy Gradients with Cartpole and Doom

Policy and value functions

- The policy defines the behavior of the agent: which action is performed in each state.
- A deterministic policy $\mu(s_t)$ is a mapping of: $\mathcal{S} \rightarrow \mathcal{A}$.
- A stochastic policy $\pi : \mathcal{S} \rightarrow P(\mathcal{A})$ defines the probability distribution $P(\mathcal{A})$ of performing an action.



[1]

[1] Thomas Simonini - An introduction to Policy Gradients with Cartpole and Doom

Policy and value functions

- The performance of a policy is determined by estimating the discounted return, i.e., the sum of all rewards received from time step t onwards:

$$R_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{1}$$

where $0 < \gamma \leq 1$ is the discount rate and r_{t+1} represents the reward obtained during the transition from s_t to s_{t+1}.

Policy and value functions

- The performance of a policy is determined by estimating the discounted return, i.e., the sum of all rewards received from time step t onwards:

$$R_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{1}$$

  where $0 < \gamma \leq 1$ is the discount rate and r_{t+1} represents the reward obtained during the transition from s_t to s_{t+1}.
- If the task is episodic (H is finite), γ can be set to 1.

Policy and value functions

- The performance of a policy is determined by estimating the discounted return, i.e., the sum of all rewards received from time step t onwards:

$$R_t = \sum_{k=0}^{H} \gamma^k r_{t+k+1} \tag{1}$$

  where $0 < \gamma \leq 1$ is the discount rate and r_{t+1} represents the reward obtained during the transition from s_t to s_{t+1}.
- If the task is episodic (H is finite), γ can be set to 1.
- If the task is continuing ($H = \infty$), γ must be chosen smaller than 1.

Policy and value function

- The Q-value of a state-action pair (s, a) is defined as the **expected** discounted return received if the agent takes action a from a state s and follows the policy π thereafter:

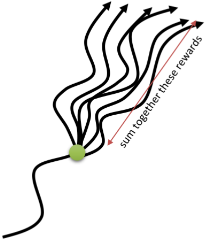
$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] \tag{2}$$

Policy and value function

- The Q-value of a state-action pair (s, a) is defined as the **expected** discounted return received if the agent takes action a from a state s and follows the policy π thereafter:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] \tag{2}$$

- The Q-value of a pair (s, a) is the expectation of the return over all trajectories starting in (s, a) defined by the policy π.

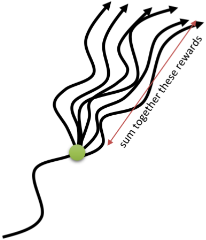


Policy and value function

- The Q-value of a state-action pair (s, a) is defined as the **expected** discounted return received if the agent takes action a from a state s and follows the policy π thereafter:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a] \tag{2}$$

- The Q-value of a pair (s, a) is the expectation of the return over all trajectories starting in (s, a) defined by the policy π.



- The value of a state s is the **expected** discounted return received if the agent starts in s and follows its policy π.

$$V^{\pi}(s) = \mathbb{E}_{\pi}[R_t | s_t = s]$$

Bellman Equations

- The value of a state depends on the values of the actions possible in that state, modulated by the probability the action will be taken by the policy π:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s, a) \tag{3}$$

Bellman Equations

- The value of a state depends on the values of the actions possible in that state, modulated by the probability the action will be taken by the policy π:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s, a) \tag{3}$$

- For deterministic policy, the value of a state is the value of the action that will be taken by the policy μ:

$$V^{\mu}(s) = Q^{\mu}(s, \mu(s))$$

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- The Q-value function can be written as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- The Q-value function can be written as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of an action depends on:

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- The Q-value function can be written as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of an action depends on:
  - which state we arrive in s'

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- The Q-value function can be written as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of an action depends on:
  - which state we arrive in s'
  - which probability this transition occurs $p(s'|s,a)$

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- The Q-value function can be written as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of an action depends on:
  - which state we arrive in s'
  - which probability this transition occurs $p(s'|s,a)$
  - how much reward we receive immediately $r(s,a,s')$

Bellman Equations

- The return at time step t is the sum of the immediate reward r_{t+1} and the return at the next state R_{t+1} (discounted by γ):

$$R_t = r_{t+1} + \gamma R_{t+1}$$

- The Q-value function can be written as:

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t | s_t = s, a_t = a] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of an action depends on:
  - which state we arrive in s'
  - which probability this transition occurs $p(s'|s,a)$
  - how much reward we receive immediately $r(s,a,s')$
  - how much we will receive later $V^\pi(s')$.

Bellman Equations

- We have the Bellman equation for the value of a state:

$$
\begin{aligned}
V^\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s, a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} p(s'|s, a)[r(s, a, s') + \gamma V^\pi(s')]
\end{aligned}
$$

Bellman Equations

- We have the Bellman equation for the value of a state:

$$\begin{aligned} V^\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of state depends on:

Bellman Equations

- We have the Bellman equation for the value of a state:

$$
\begin{aligned}
V^{\pi}(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) Q^{\pi}(s,a) \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^{\pi}(s')]
\end{aligned}
$$

- The value of state depends on:
  - the values of all other states

Bellman Equations

- We have the Bellman equation for the value of a state:

$$\begin{aligned} V^\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a) \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of state depends on:
  - the values of all other states
  - the current policy π

Bellman Equations

- We have the Bellman equation for the value of a state:

$$\begin{aligned} V^\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) Q^\pi(s,a) \\ &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of state depends on:
  - the values of all other states
  - the current policy π
  - the dynamics of the MDP $p(s'|a,s)$ and $r(s,a,s')$.

Bellman Equations

- We have the Bellman equation for the value of a state:

$$\begin{aligned} V^\pi(s) &= \sum_{a\in\mathcal{A}} \pi(a|s) Q^\pi(s,a) \\ &= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')] \end{aligned}$$

- The value of state depends on:
  - the values of all other states
  - the current policy π
  - the dynamics of the MDP $p(s'|a,s)$ and $r(s,a,s')$.
- We have the Bellman equation for the action at a state.

$$\begin{aligned} Q^\pi(s,a) &= \mathbb{E}_\pi[R_t|s_t = s, a_t = a] \\ &= \sum_{s'} p(s'|a,s)[r(s,a,s') + \gamma V^\pi(s')] \\ &= \sum_{s'} p(s'|a,s)[r(s,a,s') + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') Q^\pi(s',a')] \end{aligned}$$

Policy search

- **Policy search** methods directly learn the policy π_θ with a parameterized function estimator (e.g., a neural network).

Policy search

- **Policy search** methods directly learn the policy π_θ with a parameterized function estimator (e.g., a neural network).
- The goal of the neural network is to maximize an objective function representing the *return* (i.e., sum of rewards $R(\tau)$) of the trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots, s_H)$ selected by the policy π_θ.

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \rho_\theta}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1})\right] \tag{4}$$

Policy search

- **Policy search** methods directly learn the policy π_θ with a parameterized function estimator (e.g., a neural network).
- The goal of the neural network is to maximize an objective function representing the *return* (i.e., sum of rewards $R(\tau)$) of the trajectories $\tau = (s_0, a_0, s_1, a_1, \ldots, s_H)$ selected by the policy π_θ.

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \mathbb{E}_{\tau \sim \rho_\theta}\left[\sum_{t=0}^{H} \gamma^t r(s_t, a_t, s_{t+1})\right] \tag{4}$$

- Policy π_θ should generate trajectories τ with high returns $R(\tau)$ and avoid those with lose return.

Policy search

- The **likelihood** that a trajectory is generated by policy π_θ is:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \ldots, s_H) = p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{5}$$

Policy search

- The **likelihood** that a trajectory is generated by policy π_θ is:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \ldots, s_H) = p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{5}$$

- $p_0(s_0)$ is the initial probability of starting in s_0 (independent from the policy).

Policy search

- The **likelihood** that a trajectory is generated by policy π_θ is:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \ldots, s_H) = p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{5}$$

- $p_0(s_0)$ is the initial probability of starting in s_0 (independent from the policy).
- $p(s_{t+1}|s_t, a_t)$ is the transition probability defining the MDP.

Policy search

- The **likelihood** that a trajectory is generated by policy π_θ is:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \ldots, s_H) = p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{5}$$

- $p_0(s_0)$ is the initial probability of starting in s_0 (independent from the policy).
- $p(s_{t+1}|s_t, a_t)$ is the transition probability defining the MDP.
- The expectation in the objective function can be expanded:

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \int_\tau \rho_\theta(\tau) R(\tau) d\tau \tag{6}$$

Policy search

- The **likelihood** that a trajectory is generated by policy π_θ is:

$$\rho_\theta(\tau) = p_\theta(s_0, a_0, \ldots, s_H) = p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{5}$$

- $p_0(s_0)$ is the initial probability of starting in s_0 (independent from the policy).
- $p(s_{t+1}|s_t, a_t)$ is the transition probability defining the MDP.
- The expectation in the objective function can be expanded:

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \int_\tau \rho_\theta(\tau) R(\tau) d\tau \tag{6}$$

- Monte-Carlo sampling could be used to estimate $J(\theta)$:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau_i) \tag{7}$$

Policy search

Using Monte-Carlo sampling, we sample multiple trajectories $\{\tau_i\}$ and average their obtained returns:

$$J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau_i)$$

This approach suffers from several problems:

- **High variance**: the trajectory space is huge, we need a lot of sampled trajectories to properly estimate $J(\theta)$.
- **Sample complexity**: due to stability, only small changes can be made to the policy at each iteration, so we need a lot of episodes.
- For continuing tasks ($T = \infty$), the return cannot be estimated as the episode never ends.

Policy gradient

- In policy gradient methods, we apply gradient ascent on the weights θ in order to maximize $J(\theta)$.

Policy gradient

- In policy gradient methods, we apply gradient ascent on the weights θ in order to maximize $J(\theta)$.
- All we need is the gradient $\nabla_\theta J(\theta)$ of the objective function w.r.t. the weights:

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta} \tag{8}$$

Policy gradient

- In policy gradient methods, we apply gradient ascent on the weights θ in order to maximize $J(\theta)$.
- All we need is the gradient $\nabla_\theta J(\theta)$ of the objective function w.r.t. the weights:

$$\nabla_\theta J(\theta) = \frac{\partial J(\theta)}{\partial \theta} \tag{8}$$

- When a proper estimation of this policy gradient is obtained, we can perform gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta) \tag{9}$$

REINFORCE - Estimating the policy gradient

- Considering that the return $R(\tau)$ of a trajectory does not depend on the parameters θ, we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) d\tau = \int_\tau (\nabla_\theta \rho_\theta(\tau)) R(\tau) d\tau \quad (10)$$

REINFORCE - Estimating the policy gradient

- Considering that the return $R(\tau)$ of a trajectory does not depend on the parameters θ, we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) d\tau = \int_\tau (\nabla_\theta \rho_\theta(\tau)) R(\tau) d\tau \quad (10)$$

- We now use the **log-trick**: $\frac{d \log f(x)}{dx} = \frac{f'(x)}{f(x)}$ to rewrite the policy gradient of a single tracjectory:

$$\nabla_\theta \rho_\theta(\tau) = \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau)$$

REINFORCE - Estimating the policy gradient

- Considering that the return $R(\tau)$ of a trajectory does not depend on the parameters θ, we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) d\tau = \int_\tau (\nabla_\theta \rho_\theta(\tau)) R(\tau) d\tau \quad (10)$$

- We now use the **log-trick**: $\frac{d \log f(x)}{dx} = \frac{f'(x)}{f(x)}$ to rewrite the policy gradient of a single tracjectory:

$$\nabla_\theta \rho_\theta(\tau) = \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau)$$

- The policy gradient becomes:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) R(\tau) d\tau && (11) \\ &= \mathbb{E}_{\tau \sim \rho_\theta} [\nabla_\theta \log \rho_\theta(\tau) R(\tau)] && (12) \end{aligned}$$

REINFORCE - Estimating the policy gradient

- Considering that the return $R(\tau)$ of a trajectory does not depend on the parameters θ, we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \int_\tau \rho_\theta(\tau) R(\tau) d\tau = \int_\tau (\nabla_\theta \rho_\theta(\tau)) R(\tau) d\tau \quad (10)$$

- We now use the **log-trick**: $\frac{d \log f(x)}{dx} = \frac{f'(x)}{f(x)}$ to rewrite the policy gradient of a single tracjectory:

$$\nabla_\theta \rho_\theta(\tau) = \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau)$$

- The policy gradient becomes:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) R(\tau) d\tau && (11) \\ &= \mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) R(\tau)] && (12) \end{aligned}$$

- We can obtain an estimate of the policy gradient by sampling different trajectories $\{\tau_i\}$ and averaging $\nabla_\theta \log \rho_\theta(\tau_i) R(\tau_i)$.

REINFORCE - Estimating the policy gradient

- How to compute the gradient of the log-likelihood of a trajectory $\log \rho_\theta(\tau)$?

$$\log \rho_\theta(\tau) = \log \left(p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right)$$

$$= \log p_0(s_0) + \sum_{t=0}^{H} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{H} \log p(s_{t+1}|s_t, a_t)$$

REINFORCE - Estimating the policy gradient

- How to compute the gradient of the log-likelihood of a trajectory $\log \rho_\theta(\tau)$?

$$
\begin{aligned}
\log \rho_\theta(\tau) &= \log \left(p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right) \\
&= \log p_0(s_0) + \sum_{t=0}^{H} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{H} \log p(s_{t+1}|s_t, a_t)
\end{aligned}
$$

- $\log p_0(s_0)$ and $\log p(s_{t+1}|s_t, a_t)$ do not depend on the parameters θ (they are defined by the MDP), so the gradient of the log-likelihood is simply:

$$
\nabla_\theta \log \rho_\theta(\tau) = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{13}
$$

REINFORCE - Estimating the policy gradient

- How to compute the gradient of the log-likelihood of a trajectory $\log \rho_\theta(\tau)$?

$$\log \rho_\theta(\tau) = \log \left(p_0(s_0) \prod_{t=0}^{H} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) \right)$$
$$= \log p_0(s_0) + \sum_{t=0}^{H} \log \pi_\theta(a_t|s_t) + \sum_{t=0}^{H} \log p(s_{t+1}|s_t, a_t)$$

- $\log p_0(s_0)$ and $\log p(s_{t+1}|s_t, a_t)$ do not depend on the parameters θ (they are defined by the MDP), so the gradient of the log-likelihood is simply:

$$\nabla_\theta \log \rho_\theta(\tau) = \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{13}$$

- $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is called the **score function**.

REINFORCE - Estimating the policy gradient

- The policy gradient is independent from the MDP dynamics, allowing **model-free learning**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta} \left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \right] \tag{14}$$

$$= \mathbb{E}_{\tau \sim \rho_\theta} \left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t=0}^{H} \gamma^t r_{t+1} \right) \right] \tag{15}$$

REINFORCE - Estimating the policy gradient

- The policy gradient is independent from the MDP dynamics, allowing **model-free learning**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] \tag{14}$$

$$= \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t=0}^{H}\gamma^t r_{t+1}\right)\right] \tag{15}$$

- Estimating the policy gradient can now be done using Monte-Carlo sampling.

REINFORCE - Estimating the policy gradient

- The policy gradient is independent from the MDP dynamics, allowing **model-free learning**:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right] \tag{14}$$

$$= \mathbb{E}_{\tau\sim\rho_\theta}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t=0}^{H} \gamma^t r_{t+1}\right)\right] \tag{15}$$

- Estimating the policy gradient can now be done using Monte-Carlo sampling.
- The resulting algorithm is called the **REINFORCE** algorithm (Williams, 1992).

REINFORCE algorithm

While not converged:

1. Sample N trajectories $\{\tau_i\}$ using the current policy π_θ and observe the returns $\{R(\tau_i)\}$
2. Estimate the policy gradient as an average over the trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i)$$

3. Update the policy using gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

REINFORCE - Reducing the variance - Reward scaling

While not converged:

1. Sample N trajectories $\{\tau_i\}$ using the current policy π_θ and observe the returns $\{R(\tau_i)\}$
2. Compute the mean return:

$$\hat{R} = \frac{1}{N} \sum_{i=1}^{N} R(\tau_i)$$

3. Estimate the policy gradient as an average over the trajectories:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)(R(\tau_i) - \hat{R})$$

4. Update the policy using gradient ascent:

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$$

This algorithm is called **REINFORCE with baseline**.

REINFORCE - Reducing the variance - Reward scaling

- Subtracting a constant b from the returns still leads to an unbiased estimation of the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau)(R(\tau) - b)] \tag{16}$$

REINFORCE - Reducing the variance - Reward scaling

- Subtracting a constant b from the returns still leads to an unbiased estimation of the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau)(R(\tau) - b)] \tag{16}$$

- We have:

$$\begin{aligned}
\mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) b] &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) b d\tau \\
&= \int_\tau \nabla_\theta \rho_\theta(\tau) b d\tau \\
&= b \nabla_\theta \int_\tau \rho_\theta(\tau) d\tau = b \nabla_\theta 1 = 0
\end{aligned}$$

REINFORCE - Reducing the variance - Reward scaling

- Subtracting a constant b from the returns still leads to an unbiased estimation of the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau)(R(\tau) - b)] \tag{16}$$

- We have:

$$\begin{aligned}
\mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) b] &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) b d\tau \\
&= \int_\tau \nabla_\theta \rho_\theta(\tau) b d\tau \\
&= b \nabla_\theta \int_\tau \rho_\theta(\tau) d\tau = b \nabla_\theta 1 = 0
\end{aligned}$$

- If b does not depend on θ, the estimator is unbiased.

REINFORCE - Reducing the variance - Reward scaling

- Subtracting a constant b from the returns still leads to an unbiased estimation of the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau)(R(\tau) - b)] \tag{16}$$

- We have:

$$\begin{aligned}
\mathbb{E}_{\tau \sim \rho_\theta}[\nabla_\theta \log \rho_\theta(\tau) b] &= \int_\tau \rho_\theta(\tau) \nabla_\theta \log \rho_\theta(\tau) b d\tau \\
&= \int_\tau \nabla_\theta \rho_\theta(\tau) b d\tau \\
&= b \nabla_\theta \int_\tau \rho_\theta(\tau) d\tau = b \nabla_\theta 1 = 0
\end{aligned}$$

- If b does not depend on θ, the estimator is unbiased.
- **Advantage actor-critic** methods replace the constant b with *an estimate of the value of each state* $\hat{V}(s_t)$.

Policy Gradient Theorem

- REINFORCE estimate of the policy gradient after sampling:

$$
\begin{aligned}
\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i) \\
&= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=0}^{H}\gamma^k r(s_k, a_k, s_{k+1})\right)
\end{aligned}
$$

Policy Gradient Theorem

- REINFORCE estimate of the policy gradient after sampling:

$$\begin{aligned}\nabla_\theta J(\theta) &\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i) \\ &= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{H}\gamma^k r(s_k, a_k, s_{k+1})\right)\end{aligned}$$

- For each transition (s_t, a_t) the gradient of its log-likelihood $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is multiplied with the return of the whole episode $R(\tau) = \sum_{k=0}^{H}\gamma^k r(s_k, a_k, s_{k+1})$.

Policy Gradient Theorem

- REINFORCE estimate of the policy gradient after sampling:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i)$$
$$= \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=0}^{H} \gamma^k r(s_k, a_k, s_{k+1}) \right)$$

- For each transition (s_t, a_t) the gradient of its log-likelihood $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is multiplied with the return of the whole episode $R(\tau) = \sum_{k=0}^{H} \gamma^k r(s_k, a_k, s_{k+1})$.
- Causality principle: The reward received at $k = 0$ does not depend on actions taken in the future.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=t}^{H} \gamma^{k-t} r(s_k, a_k, s_{k+1}) \right) \tag{17}$$

Policy Gradient Theorem

- REINFORCE estimate of the policy gradient after sampling:

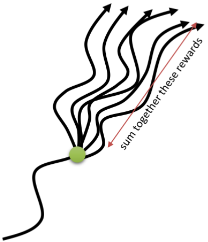
$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i)$$
$$= \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{H}\gamma^k r(s_k,a_k,s_{k+1})\right)$$

- For each transition (s_t, a_t) the gradient of its log-likelihood $\nabla_\theta \log \pi_\theta(a_t|s_t)$ is multiplied with the return of the whole episode $R(\tau) = \sum_{k=0}^{H}\gamma^k r(s_k,a_k,s_{k+1})$.
- Causality principle: The reward received at $k=0$ does not depend on actions taken in the future.

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{H}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=t}^{H}\gamma^{k-t} r(s_k,a_k,s_{k+1})\right) \tag{17}$$

- $\sum_{k=t}^{H}\gamma^{k-t} r(s_k,a_k,s_{k+1})$ is called the **reward to go** from the transition (s_t, a_t).

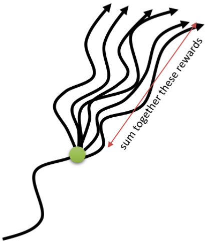
Policy Gradient Theorem



$$Q^{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots | S_t = s, A_t = a]$$

$$= \mathbb{E}_{\pi_\theta}[\sum_{k=t}^{H} \gamma^{k-t} R_{t+1} | S_t = s, A_t = a]$$

Policy Gradient Theorem



$$Q^{\pi_\theta}(s,a) = \mathbb{E}_{\pi_\theta}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s, A_t = a]$$
$$= \mathbb{E}_{\pi_\theta}[\sum_{k=t}^{H} \gamma^{k-t} R_{t+1} | S_t = s, A_t = a]$$

- The Q-value of an action (s,a) is the expectation of the reward to go.

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{k=t}^{H} \gamma^{k-t} r(s_k, a_k, s_{k+1}) \right)$$
$$\approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)$$

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)]$$

- We can sample the above, give a whole episode.

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)]$$

- We can sample the above, give a whole episode.
- Typically, people pull out the sum, and split up this into separate gradients, e.g.,

$$\begin{aligned} \Delta\theta_t &= \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t) \\ &= \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \end{aligned}$$

such that $\mathbb{E}_{\pi_\theta}[\sum_t \Delta\theta_t] = \nabla_\theta J(\theta)$.

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)]$$

- We can sample the above, give a whole episode.
- Typically, people pull out the sum, and split up this into separate gradients, e.g.,

$$\begin{aligned} \Delta\theta_t &= \nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t) \\ &= \nabla_\theta \log \pi_\theta(a_t|s_t) G_t \end{aligned}$$

  such that $\mathbb{E}_{\pi_\theta}[\sum_t \Delta\theta_t] = \nabla_\theta J(\theta)$.
- Thus, we have:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho_\theta, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t) Q^{\pi_\theta}(s_t, a_t)] \tag{18}$$

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)]$$

- The actual return $R(\tau)$ is replaced by its expectation $Q^{\pi_\theta}(s,a)$.

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)]$$

- The actual return $R(\tau)$ is replaced by its expectation $Q^{\pi_\theta}(s, a)$.
- The policy gradient is now an expectation over **single transitions** instead of complete trajectories, allowing **bootstrapping** as in TD methods.

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s,a)]$$

- The actual return $R(\tau)$ is replaced by its expectation $Q^{\pi_\theta}(s,a)$.
- The policy gradient is now an expectation over **single transitions** instead of complete trajectories, allowing **bootstrapping** as in TD methods.
- However, $Q^{\pi_\theta}(s,a)$ is unknown.

Policy Gradient Theorem

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)]$$

- The actual return $R(\tau)$ is replaced by its expectation $Q^{\pi_\theta}(s, a)$.
- The policy gradient is now an expectation over **single transitions** instead of complete trajectories, allowing **bootstrapping** as in TD methods.
- However, $Q^{\pi_\theta}(s, a)$ is unknown.
- It is possible to estimate the Q-values with a function approximator $Q_\phi(s, a)$ with parameters ϕ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)]$$

Policy Gradient Theorem

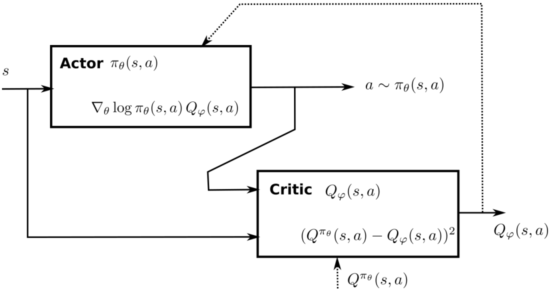
$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)]$$

- The actual return $R(\tau)$ is replaced by its expectation $Q^{\pi_\theta}(s, a)$.
- The policy gradient is now an expectation over **single transitions** instead of complete trajectories, allowing **bootstrapping** as in TD methods.
- However, $Q^{\pi_\theta}(s, a)$ is unknown.
- It is possible to estimate the Q-values with a function approximator $Q_\phi(s, a)$ with parameters ϕ:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\theta, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)]$$

- The resulting algorithm belongs to the **actor-critic** class.

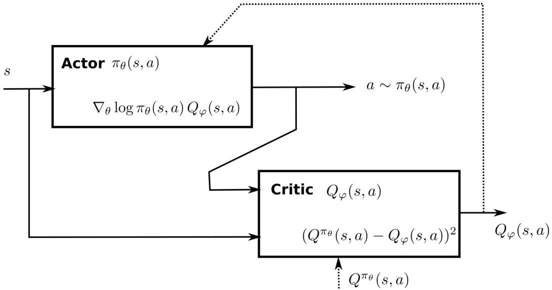
Policy Gradient Theorem



- **Actor** $\pi_\theta(a|s)$ approximates the policy by maximizing $J(\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\phi(s,a)]$$

Policy Gradient Theorem



- **Actor** $\pi_\theta(a|s)$ approximates the policy by maximizing $J(\theta)$:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) Q_\phi(s,a)]$$

- **Critic** $Q_\phi(s,a)$ estimates the policy by minimizing the MSE with the true Q-value:

$$(Q^{\pi_\theta}(s,a) - Q_\phi(s,a))^2$$

Advantage Actor-Critic Methods

- In REINFORCE, the policy gradient is estimated by:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau_i)$$

- The Policy Gradient Theorem then gives the formulation:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) Q_\phi(s_t, a_t)$$

- To reduce variance, we can employ a baseline b:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)(Q^{\pi_\theta}(s_t, a_t) - b)$$

- We make the baseline state-dependent by $b = V^{\pi_\theta}(s)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t)(Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t))$$

Advantage Actor-Critic Methods

- The factor multiplying the log-likelihood of the policy is:

$$A^{\pi_\theta}(s,a) = Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \tag{19}$$

  which is the advantage of the action a in state s. We need to approximate both function $Q^{\pi_\theta}(s,a)$ and $V^{\pi_\theta}(s)$.
- Advantage actor-critic methods approximate the advantage of an action:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\pi, a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) A_\phi(s,a)] \tag{20}$$

- $A_\phi(s,a)$ is called the advantage estimate, and should be equal to the real advantage in expectation:

$$A^{\pi_\theta}(s,a) = \mathbb{E}_{s\sim\rho_\theta, a\sim\pi_\theta}[A_\phi(s,a)]$$

Advantage Actor-Critic Methods

Different methods could be used to compute the advantage estimate $A_\phi(s,a)$:

- **MC advantage estimate**: finite episodes, slow updates

$$A_\phi(s,a) = R(s,a) - V_\phi(s) \tag{21}$$

- **TD advantage estimate**: unstable

$$A_\phi(s,a) = r(s,a,s') + \gamma V_\phi(s') - V_\phi(s) \tag{22}$$

- **n-step advantage estimate**: a trade-off btw MC and TD

$$A_\phi(s,a) = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t) \tag{23}$$

which is at the core of A2C and A3C.

Advantage Actor-Critic (A2C)

- n-step advantage estimate uses the n next immediate rewards and approximates the rest with the value of the state visited n steps later:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{n-1}\gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right)\right] \tag{24}$$

Advantage Actor-Critic (A2C)

- n-step advantage estimate uses the n next immediate rewards and approximates the rest with the value of the state visited n steps later:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{n-1}\gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right)\right] \quad (24)$$

- TD can be seen as a 1-step algorithm.

Advantage Actor-Critic (A2C)

- n-step advantage estimate uses the n next immediate rewards and approximates the rest with the value of the state visited n steps later:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{n-1}\gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right)\right] \quad (24)$$

- TD can be seen as a 1-step algorithm.
- For sparse rewards, n-step allows to update the n last actions which lead to a win/loss, instead of only the last one in TD. Also, there is no need for finite episodes as in MC.

Advantage Actor-Critic (A2C)

- n-step advantage estimate uses the n next immediate rewards and approximates the rest with the value of the state visited n steps later:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right)\right] \quad (24)$$

- TD can be seen as a 1-step algorithm.
- For sparse rewards, n-step allows to update the n last actions which lead to a win/loss, instead of only the last one in TD. Also, there is no need for finite episodes as in MC.
- n-step estimation ensures a trade-off between:

Advantage Actor-Critic (A2C)

- n-step advantage estimate uses the n next immediate rewards and approximates the rest with the value of the state visited n steps later:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right)\right] \quad (24)$$

- TD can be seen as a 1-step algorithm.
- For sparse rewards, n-step allows to update the n last actions which lead to a win/loss, instead of only the last one in TD. Also, there is no need for finite episodes as in MC.
- n-step estimation ensures a trade-off between:
  - **bias**: wrong updates based on estimated values as in TD.

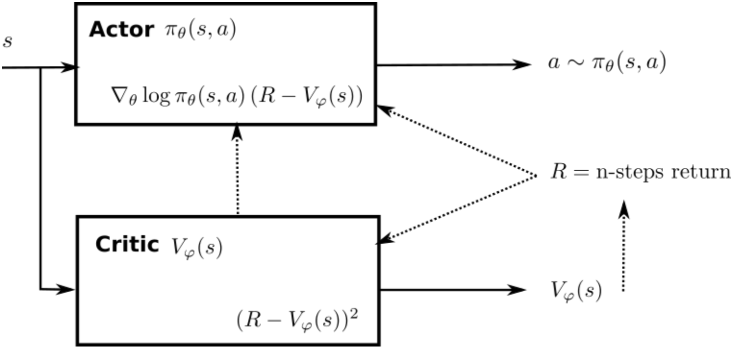
Advantage Actor-Critic (A2C)

- n-step advantage estimate uses the n next immediate rewards and approximates the rest with the value of the state visited n steps later:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{k=0}^{n-1}\gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1}) - V_\phi(s_t)\right)\right] \quad (24)$$

- TD can be seen as a 1-step algorithm.
- For sparse rewards, n-step allows to update the n last actions which lead to a win/loss, instead of only the last one in TD. Also, there is no need for finite episodes as in MC.
- n-step estimation ensures a trade-off between:
  - **bias**: wrong updates based on estimated values as in TD.
  - **variance**: variability of the obtained returns as in MC.

Advantage Actor-Critic (A2C)



- The actor outputs the policy π_θ for a state s, i.e., a vector of probabilities for each action.
- The critic outputs the value $V_\phi(s)$ of a state s.

Advantage Actor-Critic (A2C)

1. Sample a batch of transition (s, a, r, s') using the current policy π_θ.
2. For each state encountered, compute

$$R_t = \sum_{k=0}^{n-1} \gamma^k r_{t+k+1} + \gamma^n V_\phi(s_{t+n+1})$$

3. Update the actor using

$$\nabla_\theta J(\theta) = \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V\phi(s_t))$$

4. Update the critic to minimize the TD error

$$\mathcal{L}(\phi) = \sum_t (R_t - V_\phi(s_t))^2$$

5. Repeat.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
    1. Initialize empty episode minibatch.
    2. For $k \in [0, n]$: Sample episode

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     2.1 Select an action a_k using the actor π_θ

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     2.1 Select an action a_k using the actor π_θ
     2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
     2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.
  3. if s_n is not terminal: $R = V_\phi(s_n)$ with the critic, else $R = 0$.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.
  3. if s_n is not terminal: $R = V_\phi(s_n)$ with the critic, else $R = 0$.
  4. Reset gradient $d\theta$ and $d\phi$ to 0.

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
    1. Initialize empty episode minibatch.
    2. For $k \in [0, n]$: Sample episode
        2.1 Select an action a_k using the actor π_θ
        2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
        2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.
    3. if s_n is not terminal: $R = V_\phi(s_n)$ with the critic, else $R = 0$.
    4. Reset gradient $d\theta$ and $d\phi$ to 0.
    5. For $k \in [n-1, 0]$: Backwards iteration over the episode

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.
  3. if s_n is not terminal: $R = V_\phi(s_n)$ with the critic, else $R = 0$.
  4. Reset gradient $d\theta$ and $d\phi$ to 0.
  5. For $k \in [n-1, 0]$: Backwards iteration over the episode
     - 5.1 Update the discounted sum of rewards $R = r_k + \gamma R$

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.
  3. if s_n is not terminal: $R = V_\phi(s_n)$ with the critic, else $R = 0$.
  4. Reset gradient $d\theta$ and $d\phi$ to 0.
  5. For $k \in [n-1, 0]$: Backwards iteration over the episode
     - 5.1 Update the discounted sum of rewards $R = r_k + \gamma R$
     - 5.2 Accumulate the policy gradient using the critic:

$$d\theta \leftarrow d\theta + \nabla_\theta \log \pi_\theta(a_k|s_k)(R - V_\phi(s_k))$$

Advantage Actor-Critic (A2C) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Observe the initial state s_0.
- For $t \in [0, T_{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample episode
     2.1 Select an action a_k using the actor π_θ
     2.2 Perform a_k and observe the next state s_{k+1} and the reward r_{k+1}
     2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch.
  3. if s_n is not terminal: $R = V_\phi(s_n)$ with the critic, else $R = 0$.
  4. Reset gradient $d\theta$ and $d\phi$ to 0.
  5. For $k \in [n-1, 0]$: Backwards iteration over the episode
     5.1 Update the discounted sum of rewards $R = r_k + \gamma R$
     5.2 Accumulate the policy gradient using the critic:

$$d\theta \leftarrow d\theta + \nabla_\theta \log \pi_\theta(a_k|s_k)(R - V_\phi(s_k))$$

     5.3 Accumulate the critic gradient:

$$d\phi \leftarrow d\phi + \nabla_\phi (R - V_\phi(s_k))^2$$

Advantage Actor-Critic (A2C) - Pseudocode

- For $t \in [0, T_{total}]$ (continued):
- ...
  6. Update the actor with the accumulated gradients
  $$\theta \leftarrow \theta + \eta d\theta$$
  7. Update the critic with the accumulated gradients
  $$\phi \leftarrow \phi - \eta d\phi$$

N.B.

- Not all states are updated with the same horizon n.

Advantage Actor-Critic (A2C) - Pseudocode

- For $t \in [0, T_{total}]$ (continued):
- ...

6. Update the actor with the accumulated gradients

$$\theta \leftarrow \theta + \eta d\theta$$

7. Update the critic with the accumulated gradients

$$\phi \leftarrow \phi - \eta d\phi$$

N.B.

- Not all states are updated with the same horizon n.
- The last action in the sample episode will only use the last reward and the value of the final state (TD learning).

Advantage Actor-Critic (A2C) - Pseudocode

- For $t \in [0, T_{total}]$ (continued):
- ...
  6. Update the actor with the accumulated gradients

$$\theta \leftarrow \theta + \eta d\theta$$

  7. Update the critic with the accumulated gradients

$$\phi \leftarrow \phi - \eta d\phi$$

N.B.

- Not all states are updated with the same horizon n.
- The last action in the sample episode will only use the last reward and the value of the final state (TD learning).
- The first action will use the n accumulated rewards.

Advantage Actor-Critic (A2C) - Pseudocode

- For $t \in [0, T_{total}]$ (continued):
- ...
  6. Update the actor with the accumulated gradients

  $$\theta \leftarrow \theta + \eta d\theta$$

  7. Update the critic with the accumulated gradients

  $$\phi \leftarrow \phi - \eta d\phi$$

N.B.

- Not all states are updated with the same horizon n.
- The last action in the sample episode will only use the last reward and the value of the final state (TD learning).
- The first action will use the n accumulated rewards.
- A2C performs online learning: a couple of transitions are explored using the current policy, which is immediately updated.

Advantage Actor-Critic (A2C)

- As for value-based networks (e.g., DQN), the underlying neural network will be affected by the correlated inputs and outputs: a single batch contains similar states and action (e.g., consecutive frames of a video game).

Advantage Actor-Critic (A2C)

- As for value-based networks (e.g., DQN), the underlying neural network will be affected by the correlated inputs and outputs: a single batch contains similar states and action (e.g., consecutive frames of a video game).
- A2C and A3C do not use *experience replay memory* as DQN,

Advantage Actor-Critic (A2C)

- As for value-based networks (e.g., DQN), the underlying neural network will be affected by the correlated inputs and outputs: a single batch contains similar states and action (e.g., consecutive frames of a video game).
- A2C and A3C do not use *experience replay memory* as DQN,
- but they use **multiple parallel actors and learners**.

Advantage Actor-Critic (A2C)

- As for value-based networks (e.g., DQN), the underlying neural network will be affected by the correlated inputs and outputs: a single batch contains similar states and action (e.g., consecutive frames of a video game).
- A2C and A3C do not use *experience replay memory* as DQN,
- but they use **multiple parallel actors and learners**.
- The actor and critic are stored in a global network.

Advantage Actor-Critic (A2C)

- As for value-based networks (e.g., DQN), the underlying neural network will be affected by the correlated inputs and outputs: a single batch contains similar states and action (e.g., consecutive frames of a video game).
- A2C and A3C do not use *experience replay memory* as DQN,
- but they use **multiple parallel actors and learners**.
- The actor and critic are stored in a global network.
- Multiple instances of the environments are created in parallel threads (workers or actor-learners).

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     1.2 Sample an episode of n steps

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     1.2 Sample an episode of n steps
     1.3 Return the accumulated gradients $d\theta_i$ and $d\phi_i$

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     1.2 Sample an episode of n steps
     1.3 Return the accumulated gradients $d\theta_i$ and $d\phi_i$
  2. Wait for all workers to terminate.

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     1.2 Sample an episode of n steps
     1.3 Return the accumulated gradients $d\theta_i$ and $d\phi_i$
  2. Wait for all workers to terminate.
  3. Merge all accumulated gradients into $d\theta$ and $d\phi$.

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     - 1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     - 1.2 Sample an episode of n steps
     - 1.3 Return the accumulated gradients $d\theta_i$ and $d\phi_i$
  2. Wait for all workers to terminate.
  3. Merge all accumulated gradients into $d\theta$ and $d\phi$.
  4. Update the global actor and critic networks.

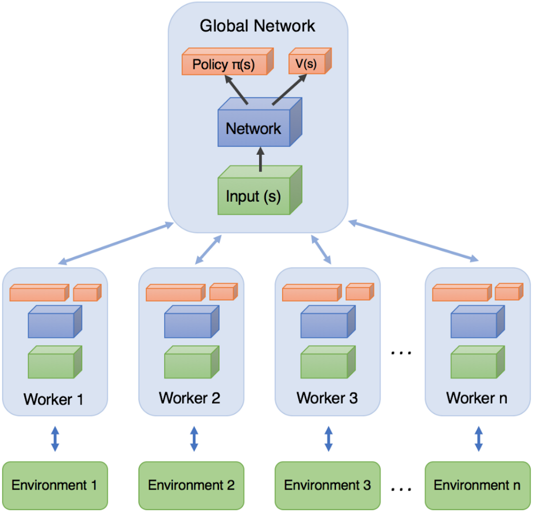
Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     1.2 Sample an episode of n steps
     1.3 Return the accumulated gradients $d\theta_i$ and $d\phi_i$
  2. Wait for all workers to terminate.
  3. Merge all accumulated gradients into $d\theta$ and $d\phi$.
  4. Update the global actor and critic networks.

Advantage Actor-Critic (A2C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- Repeat:
  1. For each worker i in parallel:
     1.1 Get a copy of the global actor π_θ and crtic V_ϕ
     1.2 Sample an episode of n steps
     1.3 Return the accumulated gradients $d\theta_i$ and $d\phi_i$
  2. Wait for all workers to terminate.
  3. Merge all accumulated gradients into $d\theta$ and $d\phi$.
  4. Update the global actor and critic networks.
- Each worker explores different regions of the environment so that the final batch for training the global networks is less correlated:
  - Set different initial states in each worker
  - Use different exploration rate
  - ...
- This method is easy for simulated environments (e.g., video games), but difficult for real-world systems like robots.

(Asynchronous) Advantage Actor-Critic (A2C - A3C)



Asynchronous Advantage Actor-Critic (A3C)

- A3C extends A2C by removing the need of synchronization between the workers at the end of each episode before applying the gradients.
- In A2C, gradient merging and parameter updates are sequential operations, so no significant speedup if the number of workers is increased.
- In A3C, each worker reads and writes the network parameters whenever it wants.
- The obtained parameters would be a mixed of different networks!!!
- However, if the learning rate is small enough, there is anyway not a big difference between two successive versions of the network parameters.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.
    2. Sample an episode of n steps.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.
    2. Sample an episode of n steps.
    3. Compute the accumulated gradients $d\theta_i$ and $d\phi_i$.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.
    2. Sample an episode of n steps.
    3. Compute the accumulated gradients $d\theta_i$ and $d\phi_i$.
    4. Update the global actor and critic networks asynchronously.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.
    2. Sample an episode of n steps.
    3. Compute the accumulated gradients $d\theta_i$ and $d\phi_i$.
    4. Update the global actor and critic networks asynchronously.

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.
    2. Sample an episode of n steps.
    3. Compute the accumulated gradients $d\theta_i$ and $d\phi_i$.
    4. Update the global actor and critic networks asynchronously.
- In the A3C paper (Mnih et al., 2016), Atari games can be solved using 16 CPU cores instead of a powerful GPU as in DQN, and achieved a better performance in less training time (1 day instead of 8).

Asynchronous Advantage Actor-Critic (A3C)

- Initialize the actor π_θ and the critic V_ϕ in the global network.
- For each worker i in parallel:
  - Repeat:
    1. Get a copy of the global actor π_θ and critic V_ϕ.
    2. Sample an episode of n steps.
    3. Compute the accumulated gradients $d\theta_i$ and $d\phi_i$.
    4. Update the global actor and critic networks asynchronously.
- In the A3C paper (Mnih et al., 2016), Atari games can be solved using 16 CPU cores instead of a powerful GPU as in DQN, and achieved a better performance in less training time (1 day instead of 8).
- The more workers, the faster the computations, the better the performance (as the policy updates are less correlated).

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.
- To enforce exploration, A3C adds an entropy regularization term to the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))] \quad (25)$$

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.
- To enforce exploration, A3C adds an entropy regularization term to the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))] \quad (25)$$

- The entropy of the policy for a state s_t can be computed:

$$H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.
- To enforce exploration, A3C adds an entropy regularization term to the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))] \quad (25)$$

- The entropy of the policy for a state s_t can be computed:

$$H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

- $H(\pi_\theta(s_t))$ measures the **randomness** of a policy:

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.
- To enforce exploration, A3C adds an entropy regularization term to the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))] \quad (25)$$

- The entropy of the policy for a state s_t can be computed:

$$H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

- $H(\pi_\theta(s_t))$ measures the **randomness** of a policy:
  - Fully deterministic policy: entropy is zero.

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.
- To enforce exploration, A3C adds an entropy regularization term to the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))] \quad (25)$$

- The entropy of the policy for a state s_t can be computed:

$$H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

- $H(\pi_\theta(s_t))$ measures the **randomness** of a policy:
  - Fully deterministic policy: entropy is zero.
  - Completely random policy: entropy is maximal.

A3C - Entropy Regularization

- In Actor-Critic methods, exploration relies on the learned policies are stochastic: (on-policy) $\pi(a|s)$ describes the probability of taking action a in state s.
- To enforce exploration, A3C adds an entropy regularization term to the policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a_t|s_t)(R_t - V_\phi(s_t)) + \beta \nabla_\theta H(\pi_\theta(s_t))] \quad (25)$$

- The entropy of the policy for a state s_t can be computed:

$$H(\pi_\theta(s_t)) = -\sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t)$$

- $H(\pi_\theta(s_t))$ measures the **randomness** of a policy:
  - Fully deterministic policy: entropy is zero.
  - Completely random policy: entropy is maximal.
- β controls the level of regularization.

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

where

1. $\psi_t = R_t$ is the REINFORCE algorithm (MC sampling).

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

where

1. $\psi_t = R_t$ is the REINFORCE algorithm (MC sampling).
2. $\psi_t = R_t - b$ is the REINFORCE with baseline algorithm.

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

where

1. $\psi_t = R_t$ is the REINFORCE algorithm (MC sampling).
2. $\psi_t = R_t - b$ is the REINFORCE with baseline algorithm.
3. $\psi_t = Q^{\pi_\theta}(s_t, a_t)$ is the policy gradient theorem.

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

where

1. $\psi_t = R_t$ is the REINFORCE algorithm (MC sampling).
2. $\psi_t = R_t - b$ is the REINFORCE with baseline algorithm.
3. $\psi_t = Q^{\pi_\theta}(s_t, a_t)$ is the policy gradient theorem.
4. $\psi_t = A^{\pi_\theta}(s_t, a_t)$ is the advantage actor critic.

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

where

1. $\psi_t = R_t$ is the REINFORCE algorithm (MC sampling).
2. $\psi_t = R_t - b$ is the REINFORCE with baseline algorithm.
3. $\psi_t = Q^{\pi_\theta}(s_t, a_t)$ is the policy gradient theorem.
4. $\psi_t = A^{\pi_\theta}(s_t, a_t)$ is the advantage actor critic.
5. $\psi_t = r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ is the TD actor critic.

Policy Gradient methods

Different versions of policy gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t)\, \psi_t] \tag{26}$$

where

1. $\psi_t = R_t$ is the REINFORCE algorithm (MC sampling).
2. $\psi_t = R_t - b$ is the REINFORCE with baseline algorithm.
3. $\psi_t = Q^{\pi_\theta}(s_t, a_t)$ is the policy gradient theorem.
4. $\psi_t = A^{\pi_\theta}(s_t, a_t)$ is the advantage actor critic.
5. $\psi_t = r_{t+1} + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$ is the TD actor critic.
6. $\psi_t = \sum_{k=0}^{n-1} r_{t+k+1} + \gamma^n V^\pi(s_{t+n+1}) - V^\pi(s_t)$ is the n-step algorithm (A2C).

Policy Gradient methods

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) \, \psi_t]$$

- The more ψ_t relies on real rewards (R_t), the more the gradient will be correct on average (small bias), but the more it will vary.

Policy Gradient methods

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) \, \psi_t]$$

- The more ψ_t relies on real rewards (R_t), the more the gradient will be correct on average (small bias), but the more it will vary.
- $\rightarrow$ the **sample complexity** is increased: we need to average more sample to correctly estimate the gradient.

Policy Gradient methods

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) \, \psi_t]$$

- The more ψ_t relies on real rewards (R_t), the more the gradient will be correct on average (small bias), but the more it will vary.
- $\rightarrow$ the **sample complexity** is increased: we need to average more sample to correctly estimate the gradient.
- The more ψ_t relies on the estimations (the TD error), the more stable the gradient (small variance), but the more incorrect it is (high bias).

Policy Gradient methods

$$\nabla_\theta J(\theta) = \mathbb{E}_{s_t \sim \rho^\pi, a_t \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s_t, a_t) \, \psi_t]$$

- The more ψ_t relies on real rewards (R_t), the more the gradient will be correct on average (small bias), but the more it will vary.
- $\rightarrow$ the **sample complexity** is increased: we need to average more sample to correctly estimate the gradient.
- The more ψ_t relies on the estimations (the TD error), the more stable the gradient (small variance), but the more incorrect it is (high bias).
- $\rightarrow$ This can lead to sub-optimal policies.

Off-policy Actor-Critic

- Actor-critic methods are generally **on-policy**: the actions used to explore the environment must be generated by the actor. Otherwise, the feedback provided by the critic (the advantage) will introduce a huge bias (i.e., an error) in the policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, Q^{\pi_\theta}(s, a)]$$

Off-policy Actor-Critic

- Actor-critic methods are generally **on-policy**: the actions used to explore the environment must be generated by the actor. Otherwise, the feedback provided by the critic (the advantage) will introduce a huge bias (i.e., an error) in the policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, Q^{\pi_\theta}(s, a)]$$

- The state distribution ρ_π defines the states that can be visited using the actor policy π_θ.

Off-policy Actor-Critic

- Actor-critic methods are generally **on-policy**: the actions used to explore the environment must be generated by the actor. Otherwise, the feedback provided by the critic (the advantage) will introduce a huge bias (i.e., an error) in the policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, Q^{\pi_\theta}(s, a)]$$

- The state distribution ρ_π defines the states that can be visited using the actor policy π_θ.
- If, during MC sampling of the policy gradient, the states s do not come from this distribution, the approximated policy gradient will be wrong (high bias) and the resulting policy will be sub-optimal.

Off-policy Actor-Critic

- Actor-critic methods are generally **on-policy**: the actions used to explore the environment must be generated by the actor. Otherwise, the feedback provided by the critic (the advantage) will introduce a huge bias (i.e., an error) in the policy gradient.

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho^\pi, a \sim \pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) \, Q^{\pi_\theta}(s, a)]$$

- The state distribution ρ_π defines the states that can be visited using the actor policy π_θ.
- If, during MC sampling of the policy gradient, the states s do not come from this distribution, the approximated policy gradient will be wrong (high bias) and the resulting policy will be sub-optimal.
- **Sample complexity**: If the actor is initialized in a flat region of the reward space (where there is not a lot of rewards), gradient updates only change slightly the policy and it may take a lot of iterations until interesting policies are discovered.

Off-policy Actor-Critic

Off-policy algorithms use a behavior policy $b(a|s)$ to **explore** the environment and **train** the target policy to reproduce the results.

- If the behavior policy does not explore the optimal actions, the target policy can't find it by itself, except by chance.

Off-policy Actor-Critic

Off-policy algorithms use a behavior policy $b(a|s)$ to **explore** the environment and **train** the target policy to reproduce the results.

- If the behavior policy does not explore the optimal actions, the target policy can't find it by itself, except by chance.
- If the behavior policy is good enough, this can reduce the amount of exploration.

Off-policy Actor-Critic

Off-policy algorithms use a behavior policy $b(a|s)$ to **explore** the environment and **train** the target policy to reproduce the results.

- If the behavior policy does not explore the optimal actions, the target policy can't find it by itself, except by chance.
- If the behavior policy is good enough, this can reduce the amount of exploration.
- Q-learning is off-policy TD learning:

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

Off-policy Actor-Critic

Off-policy algorithms use a behavior policy $b(a|s)$ to **explore** the environment and **train** the target policy to reproduce the results.

- If the behavior policy does not explore the optimal actions, the target policy can't find it by itself, except by chance.
- If the behavior policy is good enough, this can reduce the amount of exploration.
- Q-learning is off-policy TD learning:

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

- SARSA is on-policy TD learning:

$$\delta = r(s, a, s') + \gamma \, Q^\pi(s', \pi(s')) - Q^\pi(s, a)$$

Off-policy Actor-Critic

Off-policy algorithms use a behavior policy $b(a|s)$ to **explore** the environment and **train** the target policy to reproduce the results.

- If the behavior policy does not explore the optimal actions, the target policy can't find it by itself, except by chance.
- If the behavior policy is good enough, this can reduce the amount of exploration.
- Q-learning is off-policy TD learning:

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

- SARSA is on-policy TD learning:

$$\delta = r(s, a, s') + \gamma \, Q^\pi(s', \pi(s')) - Q^\pi(s, a)$$

- SARSA uses the next action sampled from $\pi(a'|s')$ to update the current transition. This next action must be performed. The policy must be ϵ-soft (e.g., ϵ-greedy).

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

- There are mostly two ways to create behavior policy:

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

- There are mostly two ways to create behavior policy:
  - Use expert knowledge / human demonstrations.

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

- There are mostly two ways to create behavior policy:
  - Use expert knowledge / human demonstrations.
  - Derive it from the target policy.

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

- There are mostly two ways to create behavior policy:
  - Use expert knowledge / human demonstrations.
  - Derive it from the target policy.
    - In Q-learning, the target policy can be **deterministic**, i.e., always select the greedy action (with the maximum Q-value).

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

- There are mostly two ways to create behavior policy:
  - Use expert knowledge / human demonstrations.
  - Derive it from the target policy.
    - In Q-learning, the target policy can be **deterministic**, i.e., always select the greedy action (with the maximum Q-value).
    - The behavior policy can be derived from the target policy by making it ϵ-soft, for example, ϵ-greedy.

Off-policy Actor-Critic

- In Q-learning, the behavior policy $b(a|s)$ must be able to explore actions which are selected by the target policy:

$$\pi(a|s) > 0 \rightarrow b(a|s) > 0$$

- There are mostly two ways to create behavior policy:
  - Use expert knowledge / human demonstrations.
  - Derive it from the target policy.
    - In Q-learning, the target policy can be **deterministic**, i.e., always select the greedy action (with the maximum Q-value).
    - The behavior policy can be derived from the target policy by making it ϵ-soft, for example, ϵ-greedy.
- Off-policy learning allows experience replay memory (ERM). The transitions used for training the target policy were generated by an older version of it. A3C is on-policy: multiple parallel learners are used to solve the correlation problems of inputs and outputs.

Off-policy methods - Importance sampling

- Off-policy methods learn a target policy $\pi(a|s)$ while exploring with a behavior policy $b(a|s)$.

Off-policy methods - Importance sampling

- Off-policy methods learn a target policy $\pi(a|s)$ while exploring with a behavior policy $b(a|s)$.
- In policy gradient methods, we want to maximize the expected return of trajectories:

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \int_\tau \rho_\theta(\tau)\, R(\tau)\, d\tau \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau_i)$$

where ρ_θ is the distribution of trajectories τ generated by the target policy π_θ.

Off-policy methods - Importance sampling

- Off-policy methods learn a target policy $\pi(a|s)$ while exploring with a behavior policy $b(a|s)$.
- In policy gradient methods, we want to maximize the expected return of trajectories:

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_\theta}[R(\tau)] = \int_\tau \rho_\theta(\tau)\, R(\tau)\, d\tau \approx \frac{1}{N} \sum_{i=1}^{N} R(\tau_i)$$

  where ρ_θ is the distribution of trajectories τ generated by the target policy π_θ.
- If we use a behavior policy to generate the trajectories, what we are actually estimating is:

$$\hat{J}(\theta) = \mathbb{E}_{\tau \sim \rho_b}[R(\tau)] = \int_\tau \rho_b(\tau)\, R(\tau)\, d\tau$$

  where ρ_b is the distribution of the trajectories generated by the behavior policy. Thus, in general, $\hat{J}(\theta)$ can be different from $J(\theta)$.

Off-policy methods - Importance sampling

We can rewrite the objective function as:

$$
\begin{aligned}
J(\theta) &= \mathbb{E}_{\tau\sim\rho_\theta}[R(\tau)] \\
&= \int_\tau \rho_\theta(\tau)\, R(\tau)\, d\tau \\
&= \int_\tau \frac{\rho_b(\tau)}{\rho_b(\tau)}\, \rho_\theta(\tau)\, R(\tau)\, d\tau \\
&= \int_\tau \rho_b(\tau) \frac{\rho_\theta(\tau)}{\rho_b(\tau)}\, R(\tau)\, d\tau \\
&= \mathbb{E}_{\tau\sim\rho_b}\left[\frac{\rho_\theta(\tau)}{\rho_b(\tau)}\, R(\tau)\right]
\end{aligned}
$$

Importance sampling

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_b} \left[\frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau) \right] \tag{27}$$

- $\frac{\rho_\theta(\tau)}{\rho_b(\tau)}$ is the importance sampling weight for the trajectory τ.

Importance sampling

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_b} \left[\frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau) \right] \tag{27}$$

- $\frac{\rho_\theta(\tau)}{\rho_b(\tau)}$ is the importance sampling weight for the trajectory τ.
- If τ generated by b is associated with a lot of rewards $R(\tau)$ with high probability $\rho_b(\tau)$ then the actor should learn to reproduce that trajectory with high probability $\rho_\theta(\tau)$ as well to maximize $J(\theta)$.

Importance sampling

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_b} \left[\frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau) \right] \tag{27}$$

- $\frac{\rho_\theta(\tau)}{\rho_b(\tau)}$ is the importance sampling weight for the trajectory τ.
- If τ generated by b is associated with a lot of rewards $R(\tau)$ with high probability $\rho_b(\tau)$ then the actor should learn to reproduce that trajectory with high probability $\rho_\theta(\tau)$ as well to maximize $J(\theta)$.
- If the associated reward is low $(R(\tau) \approx 0$, the target policy can forget about it (by setting $\rho_\theta(\tau) \approx 0)$.

Importance sampling

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_b}\left[\frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau)\right]$$

- Using the definition of the likelihood of a trajectory:

$$\begin{aligned}
\frac{\rho_\theta(\tau)}{\rho_b(\tau)} &= \frac{p_0(s_0)\ \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)}{p_0(s_0)\ \prod_{t=0}^{T} b(a_t|s_t) p(s_{t+1}|s_t, a_t)} \\
&= \frac{\prod_{t=0}^{T} \pi_\theta(a_t|s_t)}{\prod_{t=0}^{T} b(a_t|s_t)} = \prod_{t=0}^{T} \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)}
\end{aligned}$$

Importance sampling

$$J(\theta) = \mathbb{E}_{\tau \sim \rho_b}\left[\frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau)\right]$$

- Using the definition of the likelihood of a trajectory:

$$\begin{aligned}\frac{\rho_\theta(\tau)}{\rho_b(\tau)} &= \frac{p_0(s_0)\ \prod_{t=0}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)}{p_0(s_0)\ \prod_{t=0}^{T} b(a_t|s_t) p(s_{t+1}|s_t, a_t)} \\ &= \frac{\prod_{t=0}^{T} \pi_\theta(a_t|s_t)}{\prod_{t=0}^{T} b(a_t|s_t)} = \prod_{t=0}^{T} \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)}\end{aligned}$$

- $J(\theta)$ can then be estimated by MC sampling:

$$J(\theta) \approx \frac{1}{m} \sum_{i=1}^{m} \frac{\rho_\theta(\tau_i)}{\rho_b(\tau_i)} R(\tau_i) \tag{28}$$

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:
   - The reward r_{t+1}

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:
   - The reward r_{t+1}
   - The probability $b(a_t|s_t)$ that the behavior policy generates this transition.

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:
   - The reward r_{t+1}
   - The probability $b(a_t|s_t)$ that the behavior policy generates this transition.
   - The probability $\pi_\theta(a_t|s_t)$ that the target policy generates this transition.

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:
   - The reward r_{t+1}
   - The probability $b(a_t|s_t)$ that the behavior policy generates this transition.
   - The probability $\pi_\theta(a_t|s_t)$ that the target policy generates this transition.
2. Estimate the objective function with:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{t=0}^{H} \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)} \right) \left(\sum_{t=0}^{H} \gamma^t \, r_{t+1} \right)$$

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:
   - The reward r_{t+1}
   - The probability $b(a_t|s_t)$ that the behavior policy generates this transition.
   - The probability $\pi_\theta(a_t|s_t)$ that the target policy generates this transition.
2. Estimate the objective function with:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{t=0}^{H} \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)} \right) \left(\sum_{t=0}^{H} \gamma^t \, r_{t+1} \right)$$

3. Update the target policy to maximize $J(\theta)$.

Importance sampling

1. Generate N trajectories τ_i using the behavior policy. For each transition (s_t, a_t, s_{t+1}), store:
   - The reward r_{t+1}
   - The probability $b(a_t|s_t)$ that the behavior policy generates this transition.
   - The probability $\pi_\theta(a_t|s_t)$ that the target policy generates this transition.
2. Estimate the objective function with:

$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\prod_{t=0}^{H} \frac{\pi_\theta(a_t|s_t)}{b(a_t|s_t)} \right) \left(\sum_{t=0}^{H} \gamma^t \, r_{t+1} \right)$$

3. Update the target policy to maximize $J(\theta)$.
4. Repeat.

Policy gradient with importance sampling

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_b}\left[\nabla_\theta \log \rho_\theta(\tau) \frac{\rho_\theta(\tau)}{\rho_b(\tau)} R(\tau)\right] \tag{29}$$

- The return after being in a state s_t only depends on future states.
- The importance sampling weight only depends on the past weights.

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho_b}\left[\sum_{t=0}^{H} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\prod_{t'=0}^{t} \frac{\pi_\theta(a_{t'}|s_{t'})}{b(a_{t'}|s_{t'})}\right) \left(\sum_{t'=t}^{H} \gamma^{t'-t} r(s_{t'}, a_{t'})\right)\right] \tag{30}$$

Self-Imitation Learning (SIL)

- SIL method extends on-policy actor-critic algorithms (e.g., A2C) with a replay buffer to feed past good experiences to the neural network to speed up learning.

Self-Imitation Learning (SIL)

- SIL method extends on-policy actor-critic algorithms (e.g., A2C) with a replay buffer to feed past good experiences to the neural network to speed up learning.
- Use prioritized experience replay to select only transitions whose actual return is higher than their current value.

Self-Imitation Learning (SIL)

- SIL method extends on-policy actor-critic algorithms (e.g., A2C) with a replay buffer to feed past good experiences to the neural network to speed up learning.
- Use prioritized experience replay to select only transitions whose actual return is higher than their current value.
- Two *additional* loss functions for the actor and the critic:

$$\mathcal{L}^{\text{SIL}}_{\text{actor}}(\theta) = \mathbb{E}_{s,a \in \mathcal{D}}[\log \pi_\theta(a|s) \, (R(s,a) - V_\varphi(s))^+]$$
$$\mathcal{L}^{\text{SIL}}_{\text{critic}}(\varphi) = \mathbb{E}_{s,a \in \mathcal{D}}[((R(s,a) - V_\varphi(s))^+)^2]$$

where $(x)^+ = \max(0, x)$ is the positive function.

Self-Imitation Learning (SIL)

- SIL method extends on-policy actor-critic algorithms (e.g., A2C) with a replay buffer to feed past good experiences to the neural network to speed up learning.
- Use prioritized experience replay to select only transitions whose actual return is higher than their current value.
- Two *additional* loss functions for the actor and the critic:

$$\mathcal{L}^{\text{SIL}}_{\text{actor}}(\theta) = \mathbb{E}_{s,a \in \mathcal{D}}[\log \pi_\theta(a|s) \, (R(s,a) - V_\varphi(s))^+]$$
$$\mathcal{L}^{\text{SIL}}_{\text{critic}}(\varphi) = \mathbb{E}_{s,a \in \mathcal{D}}[((R(s,a) - V_\varphi(s))^+)^2]$$

  where $(x)^+ = \max(0, x)$ is the positive function.
- Transitions sampled from the replay buffer will participate to the off-policy learning only if their return is higher than the (currently known) expected value of the state $V_\phi(s)$.

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_{\text{total}}]$:
  1. Initialize empty episode minibatch.

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_{\text{total}}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_\text{total}]$:
    1. Initialize empty episode minibatch.
    2. For $k \in [0, n]$: Sample
        - 2.1 Select an action a_k using the actor π_θ

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_\text{total}]$:
    1. Initialize empty episode minibatch.
    2. For $k \in [0, n]$: Sample
        - 2.1 Select an action a_k using the actor π_θ
        - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_\text{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_\text{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch
  3. If s_n is not terminal: set $R_n = V_\phi(s_n)$, else $R_n = 0$.

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_{\text{total}}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch
  3. If s_n is not terminal: set $R_n = V_\phi(s_n)$, else $R_n = 0$.
  4. For $k \in [n-1, 0]$: #Backward iteration over the episode

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_{\text{total}}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch
  3. If s_n is not terminal: set $R_n = V_\phi(s_n)$, else $R_n = 0$.
  4. For $k \in [n-1, 0]$: #Backward iteration over the episode
     - 4.1 Update the discounted sum of rewards $R_k = r_k + \gamma R_{k+1}$

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_{\text{total}}]$:
    1. Initialize empty episode minibatch.
    2. For $k \in [0, n]$: Sample
        - 2.1 Select an action a_k using the actor π_θ
        - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}
        - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch
    3. If s_n is not terminal: set $R_n = V_\phi(s_n)$, else $R_n = 0$.
    4. For $k \in [n-1, 0]$: #Backward iteration over the episode
        - 4.1 Update the discounted sum of rewards $R_k = r_k + \gamma R_{k+1}$
        - 4.2 Store in the replay buffer $\mathcal{D}$.

Self-Imitation Learning (SIL) - Pseudocode

- Initialize the actor π_θ and the critic V_ϕ with random weights.
- Initialize the prioritized experience replay buffer $\mathcal{D}$.
- Observe the initial state s_0.
- For $t \in [0, T_\text{total}]$:
  1. Initialize empty episode minibatch.
  2. For $k \in [0, n]$: Sample
     - 2.1 Select an action a_k using the actor π_θ
     - 2.2 Perform the action a_k and observe s_{k+1} and r_{k+1}
     - 2.3 Store (s_k, a_k, r_{k+1}) in the episode minibatch
  3. If s_n is not terminal: set $R_n = V_\phi(s_n)$, else $R_n = 0$.
  4. For $k \in [n-1, 0]$: #Backward iteration over the episode
     - 4.1 Update the discounted sum of rewards $R_k = r_k + \gamma R_{k+1}$
     - 4.2 Store in the replay buffer $\mathcal{D}$.
  5. Update the actor and the critic on-policy with the episode:

$$\theta \leftarrow \theta + \eta \sum_k \nabla_\theta \log \pi_\theta(a_k|s_k)\,(R_k - V_\varphi(s_k))$$

$$\varphi \leftarrow \varphi + \eta \sum_k \nabla_\varphi (R - V_\varphi(s_k))^2$$

Self-Imitation Learning (SIL) - Pseudocode

- ...
  6. For $m \in [0, M]$:
     6.1 Sample a minibatch of K transitions (s_k, a_k, R_k) from the replay buffer $\mathcal{D}$ prioritized with high $(R_k - V_\phi(s_k))$.
     6.2 Update the actor and the critic off-policy with self-imitation:

$$\theta \leftarrow \theta + \eta \sum_k \nabla_\theta \log \pi_\theta(a_k|s_k) \, (R_k - V_\varphi(s_k))^+$$

$$\varphi \leftarrow \varphi + \eta \sum_k \nabla_\varphi ((R_k - V_\varphi(s_k))^+)^2$$

Self-Imitation Learning (SIL) - Pseudocode

- ...
  6. For $m \in [0, M]$:
     6.1 Sample a minibatch of K transitions (s_k, a_k, R_k) from the replay buffer $\mathcal{D}$ prioritized with high $(R_k - V_\phi(s_k))$.
     6.2 Update the actor and the critic off-policy with self-imitation:

$$\theta \leftarrow \theta + \eta \sum_k \nabla_\theta \log \pi_\theta(a_k|s_k) \left(R_k - V_\varphi(s_k)\right)^+$$

$$\varphi \leftarrow \varphi + \eta \sum_k \nabla_\varphi ((R_k - V_\varphi(s_k))^+)^2$$

- A2C+SIL is shown to have a better performance than SoTA methods (A3C, TRPO, Reactor, PPO) on Atari games and continuous control problems (MuJoCo).

Deterministic Policy Gradient

- So far, in policy gradient methods, we assume a stochastic policy $\pi_\theta(a_t|s_t)$ assigning probabilities to each discrete action (or some distribution for continuous action).

Deterministic Policy Gradient

- So far, in policy gradient methods, we assume a stochastic policy $\pi_\theta(a_t|s_t)$ assigning probabilities to each discrete action (or some distribution for continuous action).
- Stochastic policies ensure exploration of the state-action space: actions have a non-zero probability of being selected.

Deterministic Policy Gradient

- So far, in policy gradient methods, we assume a stochastic policy $\pi_\theta(a_t|s_t)$ assigning probabilities to each discrete action (or some distribution for continuous action).
- Stochastic policies ensure exploration of the state-action space: actions have a non-zero probability of being selected.
- Drawbacks:

Deterministic Policy Gradient

- So far, in policy gradient methods, we assume a stochastic policy $\pi_\theta(a_t|s_t)$ assigning probabilities to each discrete action (or some distribution for continuous action).
- Stochastic policies ensure exploration of the state-action space: actions have a non-zero probability of being selected.
- Drawbacks:
    - The policy gradient theorem only works on-policy. This prevents the use of an experience replay memory as in in DQN to stabilize learning. Importance sampling can help, but is unstable for long trajectories.

Deterministic Policy Gradient

- So far, in policy gradient methods, we assume a stochastic policy $\pi_\theta(a_t|s_t)$ assigning probabilities to each discrete action (or some distribution for continuous action).
- Stochastic policies ensure exploration of the state-action space: actions have a non-zero probability of being selected.
- Drawbacks:
    - The policy gradient theorem only works on-policy. This prevents the use of an experience replay memory as in in DQN to stabilize learning. Importance sampling can help, but is unstable for long trajectories.
    - Because of the stochasticity of the policy, the returns may vary a lot between two episodes generated by the same optimal policy $\rightarrow$ a lot of variance in the policy gradient $\rightarrow$ worse sample complexity than value-based methods $\rightarrow$ more samples to get rid of this variance.

Deterministic Policy Gradient

- Value-based methods like DQN produce a **deterministic policy**.

Deterministic Policy Gradient

- Value-based methods like DQN produce a **deterministic policy**.
- After learning, the action to be executed is the greedy action:

$$a_t^* = \arg\max_a Q_\theta(s_t, a)$$

Deterministic Policy Gradient

- Value-based methods like DQN produce a **deterministic policy**.
- After learning, the action to be executed is the greedy action:

$$a_t^* = \arg\max_a Q_\theta(s_t, a)$$

- Exploration is enforced by forcing the behavior policy (the one used to generate samples) to be stochastic (ϵ-greedy), but the learned policy is itself deterministic.

Deterministic Policy Gradient

- Value-based methods like DQN produce a **deterministic policy**.
- After learning, the action to be executed is the greedy action:

$$a_t^* = \arg\max_a Q_\theta(s_t, a)$$

- Exploration is enforced by forcing the behavior policy (the one used to generate samples) to be stochastic (ϵ-greedy), but the learned policy is itself deterministic.
- This is off-policy learning: allowing to use a different policy than the learned one to explore.

Deterministic Policy Gradient

- Value-based methods like DQN produce a **deterministic policy**.
- After learning, the action to be executed is the greedy action:

$$a_t^* = \arg\max_a Q_\theta(s_t, a)$$

- Exploration is enforced by forcing the behavior policy (the one used to generate samples) to be stochastic (ϵ-greedy), but the learned policy is itself deterministic.
- This is off-policy learning: allowing to use a different policy than the learned one to explore.
- When using an experience replay memory, the behavior policy is simply an older version of the learning policy (samples stored in the ERM were generated by an older version of the actor).

Deterministic Policy Gradient

- We want to learn a *parameterized* deterministic policy $\mu_\theta(s)$.

Deterministic Policy Gradient

- We want to learn a *parameterized* deterministic policy $\mu_\theta(s)$.
- The goal is to maximize the expectation over all states reachable by the policy of the **reward to go** (return) after each action:

$$J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[R(s, \mu_\theta(s))] \tag{31}$$

Deterministic Policy Gradient

- We want to learn a *parameterized* deterministic policy $\mu_\theta(s)$.
- The goal is to maximize the expectation over all states reachable by the policy of the **reward to go** (return) after each action:
$$J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[R(s, \mu_\theta(s))] \tag{31}$$
- As in the stochastic case, the distribution of states reachable is impossible to estimate, so we have to perform approximation.

Deterministic Policy Gradient

- We want to learn a *parameterized* deterministic policy $\mu_\theta(s)$.
- The goal is to maximize the expectation over all states reachable by the policy of the **reward to go** (return) after each action:
$$J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[R(s, \mu_\theta(s))] \tag{31}$$
- As in the stochastic case, the distribution of states reachable is impossible to estimate, so we have to perform approximation.
- Q-value of an action is the expectation of the reward to go after that action $Q^\pi(s, a) = \mathbb{E}_\pi[R(s, a)]$.

Deterministic Policy Gradient

- We want to learn a *parameterized* deterministic policy $\mu_\theta(s)$.
- The goal is to maximize the expectation over all states reachable by the policy of the **reward to go** (return) after each action:

$$J(\theta) = \mathbb{E}_{s\sim\rho_\mu}[R(s, \mu_\theta(s))] \tag{31}$$

- As in the stochastic case, the distribution of states reachable is impossible to estimate, so we have to perform approximation.
- Q-value of an action is the expectation of the reward to go after that action $Q^\pi(s, a) = \mathbb{E}_\pi[R(s, a)]$.
- Maximizing the returns of maximizing the true Q-value of all actions leads to the same optimal policy.

Deterministic Policy Gradient

- We want to learn a *parameterized* deterministic policy $\mu_\theta(s)$.
- The goal is to maximize the expectation over all states reachable by the policy of the **reward to go** (return) after each action:
$$J(\theta) = \mathbb{E}_{s\sim\rho_\mu}[R(s, \mu_\theta(s))] \tag{31}$$
- As in the stochastic case, the distribution of states reachable is impossible to estimate, so we have to perform approximation.
- Q-value of an action is the expectation of the reward to go after that action $Q^\pi(s,a) = \mathbb{E}_\pi[R(s,a)]$.
- Maximizing the returns of maximizing the true Q-value of all actions leads to the same optimal policy.
- Like in Policy Iteration: policy evaluation first finds the true Q-value of all state-action pairs and policy improvement changes the policy by selecting the action with the maximal Q-value $a_t^* = \arg\max_a Q_\theta(s_t, a)$.

Deterministic Policy Gradient

- In continuous control, the gradient of the objective function is the same as the gradient of the Q-value.
- If we have an unbiased estimate $Q^{\mu}(s, a)$ of the value of any action in s, changing the policy $\mu_\theta(s)$ in the direction of $\nabla_\theta Q^{\mu}(s, a)$ leads to an action with a higher Q-value:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta Q^{\mu}(s, a)|_{a=\mu_\theta(s)}] \tag{32}$$

- This is the gradient with respect to the action a of the Q-value is taken at $a = \mu_\theta(s)$. Using the chain rule:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^{\mu}(s, a)|_{a=\mu_\theta(s)}] \tag{33}$$

in which:

$$\frac{\partial Q(s, a)}{\partial \theta} = \frac{\partial Q(s, a)}{\partial a} \times \frac{\partial a}{\partial \theta}$$

Deterministic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

- The first term defines how the action changes when the parameters θ of the actor change.

Deterministic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

- The first term defines how the action changes when the parameters θ of the actor change.
- The second term defines how the Q-value of an action changes when we vary slightly the action (e.g., if the robot moves its joint a bit more to the right, does it get a higher Q-value?).

Deterministic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]$$

- The first term defines how the action changes when the parameters θ of the actor change.
- The second term defines how the Q-value of an action changes when we vary slightly the action (e.g., if the robot moves its joint a bit more to the right, does it get a higher Q-value?).
- This looks like an actor-critic architecture.

Deterministic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$$

- The first term defines how the action changes when the parameters θ of the actor change.
- The second term defines how the Q-value of an action changes when we vary slightly the action (e.g., if the robot moves its joint a bit more to the right, does it get a higher Q-value?).
- This looks like an actor-critic architecture.
- $\nabla_\theta \mu_\theta(s)$ only depends on the actor, while $\nabla_a Q^\mu(s, a)$ is like a critic, telling the actor in which direction to change its policy.

Deterministic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$$

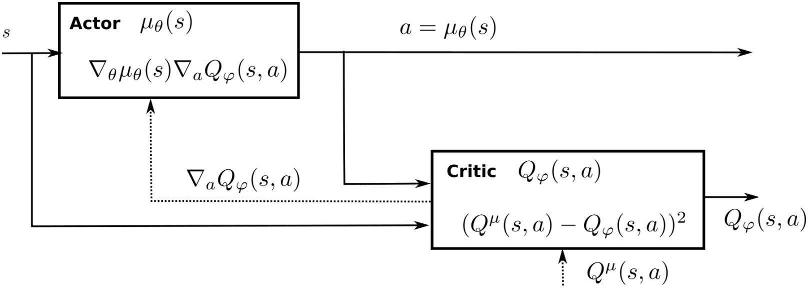
- The first term defines how the action changes when the parameters θ of the actor change.
- The second term defines how the Q-value of an action changes when we vary slightly the action (e.g., if the robot moves its joint a bit more to the right, does it get a higher Q-value?).
- This looks like an actor-critic architecture.
- $\nabla_\theta \mu_\theta(s)$ only depends on the actor, while $\nabla_a Q^\mu(s, a)$ is like a critic, telling the actor in which direction to change its policy.
- How to obtain an unbiased estimate of the Q-value of any action and compute its gradient?

Deterministic Policy Gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim \rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)}]$$

- The first term defines how the action changes when the parameters θ of the actor change.
- The second term defines how the Q-value of an action changes when we vary slightly the action (e.g., if the robot moves its joint a bit more to the right, does it get a higher Q-value?).
- This looks like an actor-critic architecture.
- $\nabla_\theta \mu_\theta(s)$ only depends on the actor, while $\nabla_a Q^\mu(s, a)$ is like a critic, telling the actor in which direction to change its policy.
- How to obtain an unbiased estimate of the Q-value of any action and compute its gradient?
- We can use a function approximator $Q_\phi(s, a)$ and minimize the quadratic error with the true Q-values.

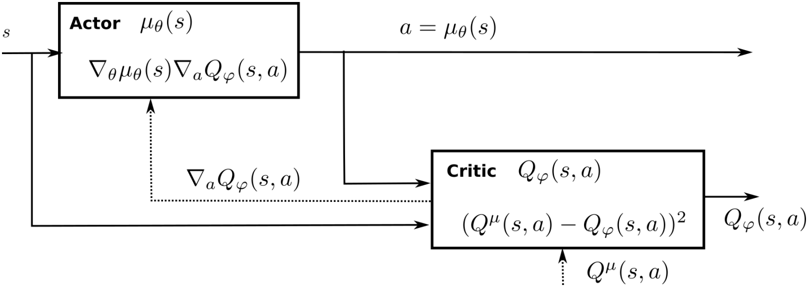
Deterministic Policy Gradient



$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q_\varphi(s,a)|_{a=\mu_\theta(s)}]$$

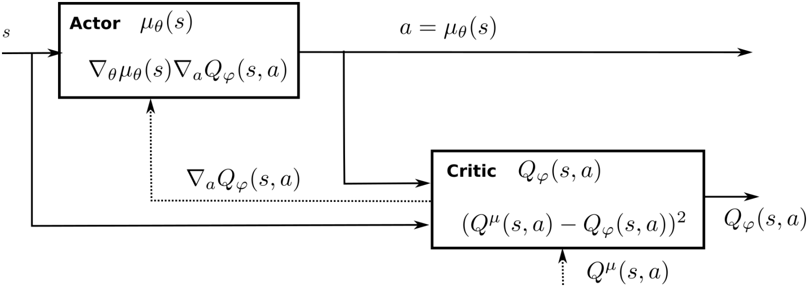
$$J(\varphi) = \mathbb{E}_{s\sim\rho_\mu}[(Q^\mu(s,\mu_\theta(s)) - Q_\varphi(s,\mu_\theta(s)))^2]$$

Deterministic Policy Gradient



$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{s\sim\rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q_\varphi(s,a)|_{a=\mu_\theta(s)}] \\
J(\varphi) &= \mathbb{E}_{s\sim\rho_\mu}[(Q^\mu(s,\mu_\theta(s)) - Q_\varphi(s,\mu_\theta(s)))^2]
\end{aligned}
$$

Deterministic Policy Gradient



$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim\rho_\mu}[\nabla_\theta \mu_\theta(s) \times \nabla_a Q_\varphi(s,a)|_{a=\mu_\theta(s)}]$$
$$J(\varphi) = \mathbb{E}_{s\sim\rho_\mu}[(Q^\mu(s,\mu_\theta(s)) - Q_\varphi(s,\mu_\theta(s)))^2]$$

- However, this architecture worked only with linear function approximators, but not yet with non-linear approximators (e.g., neural networks).

Deep Deterministic Policy Gradient (DDPG)

- DDPG combines ideas from DQN and DPG to solve continuous problems off-policy.
  - deterministic policy gradient for the actor.
  - experience replay memory to store past transitions and learn off-policy.
  - target networks to stabilize learning.
- In DQN, the target networks are updated with the parameters of the trained networks after every interval of a few thousand steps. The target networks change a lot between two updates, but not often.
- In DDPG, the target networks are updated after each update of the trained networks by using a sliding average for both the actor and critic:

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

  with $\tau << 1$. The target networks are always "late" with respect to the trained networks, providing more stability to the learning of Q-values.

Deep Deterministic Policy Gradient (DDPG)

- The critic is learned using Q-learning and target networks:

$$J(\varphi) = \mathbb{E}_{s \sim \rho_\mu}[(r(s, a, s') + \gamma\, Q_{\varphi'}(s', \mu_{\theta'}(s')) - Q_\varphi(s, a))^2] \tag{34}$$

- Exploration issue: Because the policy is deterministic, it can produce the same actions, missing more rewarding ones. DDPG then adds an additive noise to the deterministic action:

$$a_t = \mu_\theta(s_t) + \xi \tag{35}$$

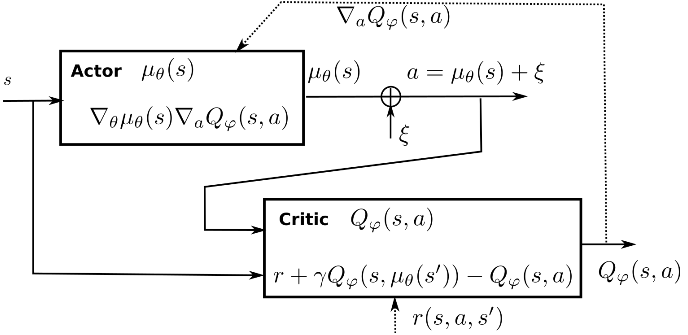
Deep Deterministic Policy Gradient (DDPG)

- The critic is learned using Q-learning and target networks:

$$J(\varphi) = \mathbb{E}_{s \sim \rho_\mu}[(r(s, a, s') + \gamma \, Q_{\varphi'}(s', \mu_{\theta'}(s')) - Q_\varphi(s, a))^2] \tag{34}$$

- Exploration issue: Because the policy is deterministic, it can produce the same actions, missing more rewarding ones. DDPG then adds an additive noise to the deterministic action:

$$a_t = \mu_\theta(s_t) + \xi \tag{35}$$



Deep Deterministic Policy Gradient (DDPG)

- Initialize actor μ_θ and critic Q_ϕ with random weights.
- Create target networks $\mu_{\theta'}$ and $Q_{\phi'}$.
- Initialize experience replay memory D.
- For episode $\in [1, M]$:
  - Initialize random process ξ
  - Observe the initial state s_0.
  - For $t \in [0, T_{max}$:
    1. Select action $a_t = \mu_\theta(s_t) + \xi$
    2. Perform a_t, observe the next state s_{t+1} and the reward r_{t+1}.
    3. Store $(s_t, a_t, r_{t+1}, s_{t+1})$ to experience replay memory D.
    4. Sample a minibatch of N transitions randomly from D.
    5. For each transition (s_k, a_k, r_k, s'_k) in the minibatch, compute the target value using the target networks:

$$y_k = r_k + \gamma \, Q_{\varphi'}(s'_k, \mu_{\theta'}(s'_k))$$

    6. Update the critic by minimizing:

$$\mathcal{L}(\varphi) = \frac{1}{N} \sum_k (y_k - Q_\varphi(s_k, a_k))^2$$

Deep Deterministic Policy Gradient (DDPG)

- ...
  - ...
    7. Update the actor using the sampled policy gradient:

    $$\nabla_\theta J(\theta) = \frac{1}{N} \sum_k \nabla_\theta \mu_\theta(s_k) \times \nabla_a Q_\varphi(s_k, a)|_{a=\mu_\theta(s_k)}$$

    8. Update the target networks:

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