

7. Optimal Control and Policy Based Reinforcement Learning

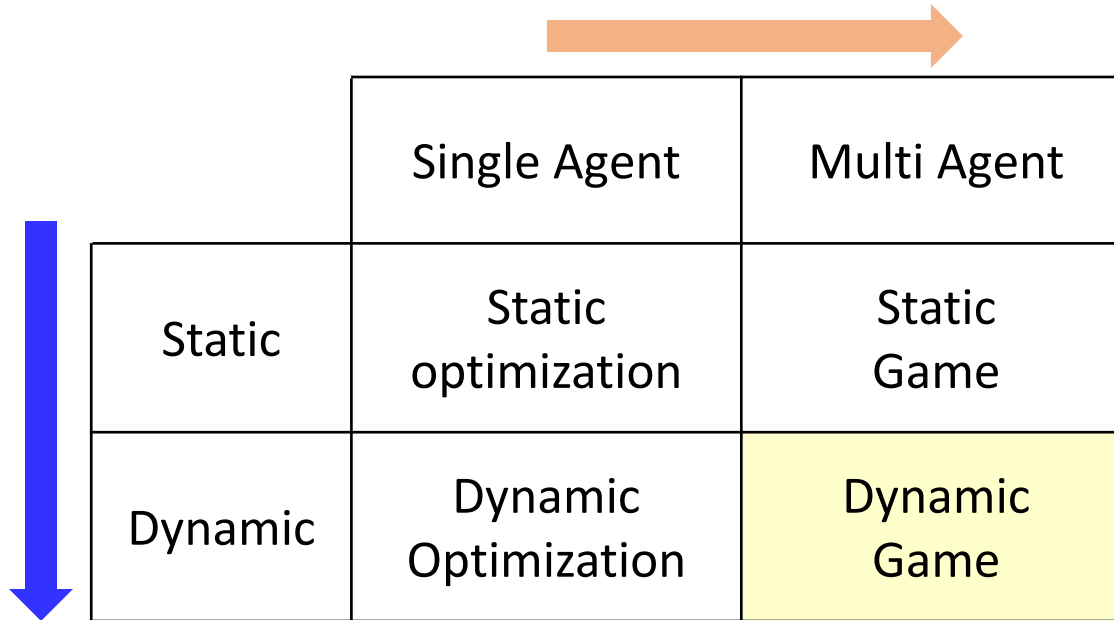
Optimal Control

- **Discrete-Time Optimal Control**
 - ✓ Dynamic Programming
 - ✓ Affine Quadratic Problem
 - ✓ Infinite Horizon Linear Quadratic Problem
- **Continuous-Time Optimal Control**
 - ✓ Dynamic Programming (Hamilton-Jacobi-Bellman Equation)
 - ✓ Minimum Principle
 - ✓ Affine Quadratic Problem
 - ✓ Infinite Horizon Linear Quadratic Problem

Basic Principle to Analyze Dynamic Games

Equilibrium concept:

-Nash; Zero-sum; Stackelberg; Correlated



	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	Dynamic Game

Dynamic optimization as a static optimization concept:

- Minimum principle (necessary condition)
- Dynamic programming principle (sufficient condition)
- Need to specify information structure

$$\begin{aligned} L^{1*} &\triangleq L^1(u^{1*}; u^{2*}; \dots; u^{N*}) \leq L^1(u^1; u^{2*}; \dots; u^{N*}), \\ L^{2*} &\triangleq L^2(u^{1*}; u^{2*}; \dots; u^{N*}) \leq L^2(u^{1*}; u^2; \dots; u^{N*}), \\ &\dots \\ L^{N*} &\triangleq L^N(u^{1*}; u^{2*}; \dots; u^{N*}) \leq L^N(u^{1*}; u^{2*}; \dots; u^{N*}) \end{aligned}$$

(Think in normal form game setting)

Overview

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	Dynamic Game

Action space

Time space	Model based	Finite	Infinite
	Discrete	Discrete time MDP $P(s_{t+1} s_t, a_t)$	Discrete-time dynamic system $x_{t+1} = f(x_t, u_t)$
	Continuous	Continuous time MDP $P(s_{t+h} s_t, a_t)$	Continuous-time dynamic system $\dot{x}_t = f(x_t, u_t)$

Overview

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	Dynamic Game

Action space

Time space	Model free	Finite	Infinite
	Discrete	Value-based Reinforcement Learning	Policy-based Reinforcement Learning
	Continuous		

Overview

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	Dynamic Game

Action space		
Time space	Model based	
	Finite	Infinite
	Discrete	DT Infinite dynamic game (Stochastic Game)
	Continuous	CT-time Infinite dynamic game (differential game)

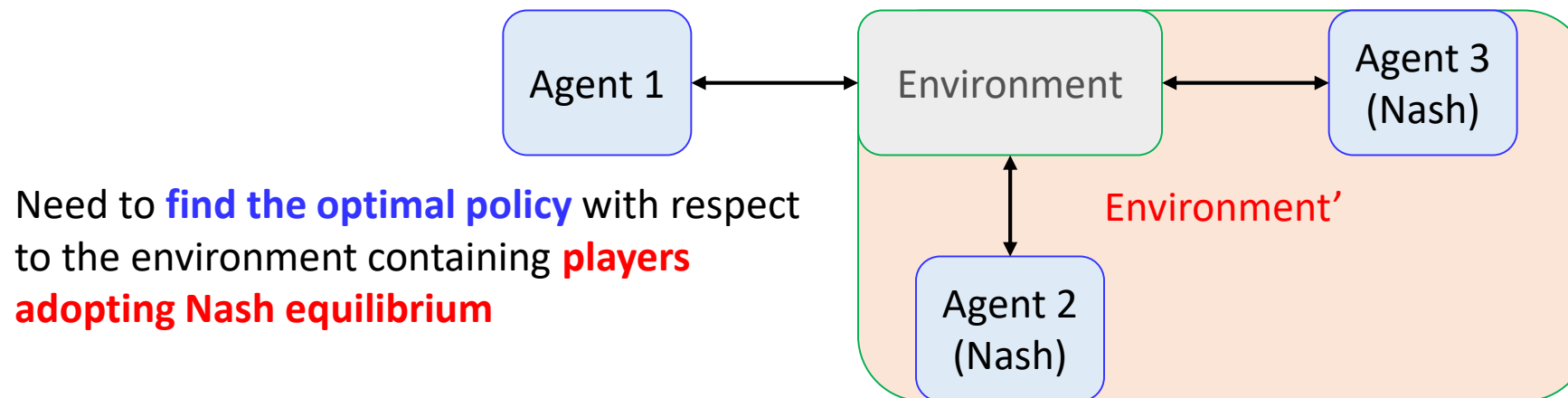
Overview

	Single Agent	Multi Agent
Static	Static optimization	Static Game
Dynamic	Dynamic Optimization	Dynamic Game

Action space			
Time space	Model free	Finite	Infinite
	Discrete	Multi-Agent Value-based RL	Multi-Agent Policy-based RL
	Continuous		

Why We Learn Optimal Control Theory?

- **Optimal control problems** constitute a special class of infinite dynamic games with one player and one criterion
 - The mathematical tools available for such problems are useful in dynamic game theory
- If the players adopt the **non-cooperative Nash equilibrium solution concept**, in which case each player is faced with a single criterion optimization problem (i.e., optimal control problem) with the strategies of the remaining players taken to be **fixed** at their equilibrium values.
 - Hence, in order to verify whether a given set of strategies is in Nash equilibrium, we inevitably have to utilize the tools of **optimal control theory**.



Discrete-Time Optimal Control Problems

Want to find **a control sequence** $u = \{u_k, k \in \mathbf{K}\}$ which minimizes

$$L(u) = \sum_{k=0}^{K-1} g_k(x_k, u_k) + g_K(x_K) \quad (1)$$

where the state variable x_k satisfies discrete-time system constraints:

$$x_{k+1} = f_k(x_k, u_k), \quad u_k \in U_k, \quad k \in \mathbf{K} \quad (2)$$

on time steps $k \in \mathbf{K} = \{0, 1, \dots, K-1\}$

- $u = \{u_k, k \in \mathbf{K}\}, u_k = \gamma_k(x_k)$
 - ✓ $\gamma_k(\cdot)$ is a permissible control strategy at stage $k \in \mathbf{K}$

Dynamic Programming for Discrete-Time Optimal Control Problems

- In order to determine the minimizing control strategy, we define the expression for the minimum cost from **any starting point x** at any **initial time k** , which is so called value function

$$V(k, x) = \min_{\gamma_k, \dots, \gamma_K} \left[\sum_{i=k}^K g_i(x_{i+1}, u_i, x_i) \right] \quad (3)$$

with $u_i = \gamma_i(x_i) \in U_i$ and $x_k = x$

- A direct application of the **principle of optimality** now readily leads to the recursive relation

$$V(k, x) = \min_{u_k} \left[g_k(\overset{x_{k+1}}{f_k(x, u_k)}, u_k, x) + V(k+1, \overset{x_{k+1}}{f_k(x, u_k)}) \right] \quad (4)$$

- If the optimal control problem admits a solution $u^* = \{\gamma_k^*, k \in \mathbf{K}\}$, then the solution $V(1, x)$ of Eq.(4) should be equal to $L(u^*)$
- Each u_k^* should be determined as an argument of the RHS of Eq.(4) (**single-shot optimization**)

An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

Infinite horizon linear-quadratic problem (Discrete Time Optimal Control)

- discrete-time system $x_{t+1} = Ax_t + Bu_t$, $x_0 = x^{init}$
- problem: choose u_0, u_1, \dots to minimize

$$J = \sum_{\tau=0}^{\infty} (x_{\tau}^T Q x_{\tau} + u_{\tau}^T R u_{\tau})$$

with given constant state and input weight matrices

$$Q = Q^T \geq 0, \quad R = R^T > 0$$

- ✓ this is an infinite dimensional problem

Infinite horizon linear-quadratic problem (Discrete Time Optimal Control)

- **Problem:** it's possible that $J = \infty$ for all input sequences u_0, \dots

$$x_{t+1} = 2x_t + 0u_t, \quad x^{init} = 1$$

- Let's assume (A, B) is controllable
- then for any x^{init} there's an input sequence

$$u_0, \dots, u_{n-1}, 0, 0, \dots$$

that steers x to zero at $t = n$, and keeps it there

- ✓ For this u , $J < \infty$
- ✓ And therefore, $\min_u J < \infty$ for any x^{init}

Infinite horizon linear-quadratic problem (Discrete Time Optimal Control)

- To apply dynamic programming approach define value function $V : \mathbf{R}^n \rightarrow \mathbf{R}$

$$V(z) = \min_{u_0, \dots} \sum_{\tau=0}^{\infty} (x_{\tau}^T Q x_{\tau} + u_{\tau}^T R u_{\tau})$$

subject to $x_0 = z, \quad x_{\tau+1} = Ax_{\tau} + Bu_{\tau}$

- $V(z)$ is the minimum LQR cost-to-go, starting from state z
- doesn't depend on time-to-go, which is always ∞ ; infinite horizon problem is *shift invariant*

Infinite horizon linear-quadratic problem (Discrete Time Optimal Control)

- **Fact:** V is quadratic, i.e., $V(z) = z^T P z$, where $P = P^T \geq 0$
- Applying Bellman minimum principle (or HJB equation for discrete system)

$$V(z) = \min_w (z^T Q z + w^T R w + V(Az + Bw))$$

or

$$z^T P z = \min_w (z^T Q z + w^T R w + (Az + Bw)^T P (Az + Bw))$$

- minimizing $w^* = -(R + B^T P B)^{-1} B^T P A z$
- so HJB equation is

$$\begin{aligned} z^T P z &= z^T Q z + w^{*T} R w^* + (Az + Bw^*)^T P (Az + Bw^*) \\ &= z^T (Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A) z \end{aligned}$$

- ✓ This must hold for all z , so we can conclude that P satisfies the **ARE**

$$P = Q + A^T P A - A^T P B (R + B^T P B)^{-1} B^T P A$$

- The optimal input is constant state feedback $u_t = K x_t$

$$K = -(R + B^T P B)^{-1} B^T P A$$

ARE: Algebraic Riccati Equation

Infinite horizon linear-quadratic problem (Discrete Time Optimal Control)

- **Fact:** the ARE has only one positive semidefinite solution P
 - ✓ ARE plus $P = P^T \geq 0$ uniquely characterizes value function
- Consequence: The Riccati recursion

$$P_{k+1} = Q + A^T P_k A - A^T P_k B (R + B^T P_k B)^{-1} B^T P_k A, \quad P_1 = Q$$

converges to the unique PSD solution of the ARE (when (A,B) controllable)

- Infinite-horizon LQR optimal control is same as steady-state finite horizon optimal control

Continuous-Time Optimal Control Problems

Consider the following optimal control problem defined by

$$L(u) = \int_0^T g(t, x(t), u(t)) dt + q(T, x(T)) \quad (5)$$

where the state variable $x(t)$ satisfies the differential equation:

$$\dot{x}(t) = f(t, x(t), u(t)), \quad x(0) = x_0, \quad t \geq 0 \quad (6)$$

- $u(t) = \gamma(t, x(t)) \in U$,
✓ $\gamma \in \Gamma$ is the class of all admissible feedback strategies

Dynamic Programming for Continuous-Time Optimal Control Problems

- The minimum cost-to-go from any initial state x and any initial time t is described by the so-called ***value function*** defined by

$$V(t, x) = \min_{u(s), t \leq s \leq T} \left[\int_t^T g(s, x(s), u(s)) ds + q(T, x(T)) \right] \quad (7)$$

Satisfying the boundary condition

$$V(T, x) = q(T, x(T)) \quad (8)$$

Hamilton-Jacobi-Bellman (HJB) equation

- The dynamic programming approach, when applied to optimal control problems defined in continuous time, leads to a partial differential equation (PDE) which is known as the **Hamilton-Jacobi-Bellman (HJB) equation**.
- A direct application of **the principle of optimality** on Eq. (7), under the assumption of continuous differentiability of V , leads to the HJB equation

$$-\frac{\partial V(t, x)}{\partial t} = \min_u \left[\frac{\partial V(t, x)}{\partial x} f(t, x, u) + g(t, x, u) \right] \quad (9)$$

with $V(T, x) = q(T, x(T))$ boundary condition

- In general, it is not easy to compute $V(t, x)$ and the continuous differentiability assumption is rather restrictive.
- If $V(t, x)$ exists, the HJB equation provides a means of obtaining the optimal control strategy **(sufficient condition)**

Proof of Hamilton-Jacobi-Bellman (HJB) equation

Proof:

According to Bellman's optimality principle, the following identity holds for small δ

$$V(t, x(t)) = \min_{u \in U} [g(t, x, u)\delta + V(t + \delta, x(t + \delta))]$$

where $V(t + \delta, x(t + \delta)) = V(t + \delta, x(t) + f(t, x, u) \cdot \delta)$

$$= V(t, x(t)) + \frac{\partial V(t, x)}{\partial t} \delta + \frac{\partial V(t, x)}{\partial x} f(t, x, u) \cdot \delta + o(\delta)$$

Substituting into (3) gives

$$V(t, x(t)) = \min_{u \in U} \left[g(t, x, u) \cdot \delta + V(t, x(t)) + \frac{\partial V(t, x)}{\partial x} f(t, x, u) \cdot \delta + \frac{\partial V(t, x)}{\partial t} \delta + o(\delta) \right]$$

Canceling $V(t, x(t))$ both side and divide by δ gives:

$$-\frac{\partial V(t, x)}{\partial t} = \min_u \left[\frac{\partial V(t, x)}{\partial x} f(t, x, u) + g(t, x, u) \right]$$

Obtaining the optimal control strategy from HJB equation

Theorem

If a continuously differentiable function $V(t, x)$ can be found that satisfies the HJB equation (9), then it generate the optimal strategy through the static (pointwise) minimization problem defined by the RHS of (9)

$$-\frac{\partial V(t, x)}{\partial t} = \min_{u \in U} \left[\frac{\partial V(t, x)}{\partial x} f(t, x, u) + g(t, x, u) \right] \quad (9)$$



$$u^*(t) = \operatorname{argmin}_{u \in U} \left[\frac{\partial V(t, x)}{\partial x} f(t, x, u) + g(t, x, u) \right] \quad (10)$$

Obtaining the optimal control strategy from HJB equation

Proof:

- If we are given two strategies, $\gamma^* \in \Gamma$ (the optimal one) and $\gamma \in \Gamma$ (an arbitrary one), with the corresponding terminating trajectories x^* and x , and terminal times T^* and T , respectively, then Eq. (9) reads

$$g(t, x, u) + \frac{\partial V(t, x)}{\partial x} f(t, x, u) + \frac{\partial V(t, x)}{\partial t} \geq 0 \quad (11)$$

$$g(t, x^*, u^*) + \frac{\partial V(t, x^*)}{\partial x} f(t, x^*, u^*) + \frac{\partial V(t, x^*)}{\partial t} \equiv 0 \quad (12)$$

where γ^* and γ have been replaced by the corresponding controls u^* and u , respectively.

- Integrating Eq.(11) from 0 to T and Eq.(12) from 0 to T^* leads to

$$\begin{aligned} \int_0^T g(t, x, u) + V(T, x(T)) - V(0, x_0) &\geq 0 \\ \int_0^{T^*} g(t, x^*, u^*) + V(T^*, x^*(T^*)) - V(0, x_0) &= 0 \end{aligned}$$

- Elimination of $V(0, x_0)$ yields

$$\int_0^T g(t, x, u) + q(T, x(T)) \geq \int_0^{T^*} g(t, x^*, u^*) + q(T^*, x^*(T^*)) \geq 0$$

➤ u^* is the optimal control (sequence of actions), and therefore γ^* is the optimal strategy

Infinite horizon linear-quadratic problem (Continuous Time Optimal Control)

- continuous-time system $\dot{x}(t) = Ax(t) + Bu(t)$
- problem: choose $u(t)$ to minimize

$$J(u) = \int_0^T (x(t)^T Q x(t) + u(t)^T R u(t)) dt + x(T)^T Q_f x(T)$$

- ✓ $g = x^T Q x + u^T R u$
- ✓ $f = Ax + Bu$
- ✓ Let's assume $V(t, x) = x^T P_t x$

- HJB equation is employed

$$-\frac{\partial V(t, x)}{\partial t} = \min_{u \in U} \left[\frac{\partial V(t, x)}{\partial x} f(t, x, u) + g(t, x, u) \right]$$
$$\Rightarrow -x \dot{P}_t x = \min_u \{ 2P_t x (Ax + Bu) + x^T Q x + u^T R u \}$$

- minimizing over u yields $u^* = -R^{-1} B^T P_t$, and substitute to HJB;

$$-\dot{P}_t = A^T P_t + P_t A - P_t B R^{-1} B^T P_t + Q$$

which is called as **Riccati differential equation** in optimal control.

The Minimum Principle

- Dynamic optimization using Dynamic Programming approach → HJB equation
 - Handy to use
 - Easy to extend to stochastic feedback control
- Dynamic optimization using the Lagrangian Method → Minimum Principle
 - Intuitively show how dynamic optimization is solved using static optimization technique
 - Will be used to state the equilibrium condition (necessary conditions) for various dynamic game

From Static Optimization to Dynamic Optimization

- We derive the first-order necessary conditions for the basic dynamic optimization problem to find a control function $u(\cdot)$ that minimizes the cost functional

$$J(u) = \int_0^T g(t, x(t), u(t)) dt + h(x(T)) \quad \text{where } h(x(T)) = q(T, x(T))$$

where the state variable $x(t)$ satisfies the differential equation:

$$\dot{x}(t) = f(t, x(t), u(t)), \quad x(0) = x_0$$

- ✓ $x(t) \in \mathbf{R}^n, u(t) \in \mathbf{R}^m$
- ✓ It is assumed that the dynamic process starts from $t = 0$ and ends at the fixed terminal time $T > 0$.
- ✓ $f(t, x, u)$ and $g(t, x, u)$ are continuous functions on \mathbf{R}^{n+m+1}
- ✓ The set of admissible control functions $u(t) \in U$ consists of the set of functions that are continuous on $[0, T]$.

The Euler-Lagrange Equation

- Inspired by the theory of static optimization we introduce for each t in the interval $[0, T]$ the quantity $\lambda(t)[f(t, x(t), u(t)) - \dot{x}(t)]$ that satisfies

$$\int_0^T \lambda(t)[f(t, x(t), u(t)) - \dot{x}(t)]dt = 0$$

- ✓ the Lagrange multiplier $\lambda(t)$ (or costate variable) is an arbitrarily chosen row vector
- Define the new cost function \bar{J} which coincides with the original cost function J if the dynamic constraint, $\dot{x}(t) = f(t, x(t), u(t))$, is satisfied as

$$\begin{aligned}\bar{J} &:= \int_0^T \{g(t, x, u) + \lambda(t)f(t, x, u) - \lambda(t)\dot{x}(t)\}dt + h(x(T)) \\ &= \int_0^T \{H(t, x, u, \lambda) - \lambda(t)\dot{x}(t)\}dt + h(x(T))\end{aligned}$$

- ✓ Where the Hamiltonian function $H(t, x, u, \lambda) = g(t, x, u) + \lambda(t)f(t, x, u)$
- Conducting integration by parts, \bar{J} can be rewritten as

$$\bar{J} := \underbrace{\int_0^T \{H(t, x, u, \lambda) + \dot{\lambda}(t)x(t)\}dt}_{\bar{J}_1} + \underbrace{h(x(T)) - \lambda(T)x(T)}_{\bar{J}_2} + \underbrace{\lambda(0)x_0}_{\bar{J}_3}$$

The Euler-Lagrange Equation

- Assume that $u^*(t) \in U$ is an optimal control path generating the minimum value of \bar{J} and $x^*(t)$ is the corresponding optimal state trajectory
- If we perturb this optimal $u^*(t)$ path with a continuous perturbing curve $p(t)$, we can generate 'neighboring' control paths

$$u(t) = u^*(t) + \epsilon p(t)$$

- ✓ ϵ is small scalar
- ✓ $u(t)$ induces a corresponding 'neighboring' state trajectory $x(t, \epsilon, p)$ with $t \in [0, T]$
- The cost function associated with the perturbation becomes
$$\bar{J}(\epsilon) := \int_0^T \{H(t, x(t, \epsilon, p), u(t), \lambda(t)) + \dot{\lambda}(t)x(t, \epsilon, p)\} dt + h(x(T, \epsilon, p), \lambda(T)) - \lambda(0)x_0$$
- By assumption $\bar{J}(\epsilon)$ has a minimum at $\epsilon = 0 \rightarrow \frac{d\bar{J}(\epsilon)}{d\epsilon} = 0$ at $\epsilon = 0$

The Euler-Lagrange Equation

$$\bar{J}(\epsilon) := \int_0^T \{H(t, x(t, \epsilon, p), u(t), \lambda(t)) + \dot{\lambda}(t)x(t, \epsilon, p)\} dt + h(x(T, \epsilon, p)) - \lambda(T)x(T, \epsilon, p) + \lambda(0)x_0$$

- Evaluating the derivative of $\bar{J}(\epsilon)$ yields

$$\frac{d\bar{J}(\epsilon)}{d\epsilon} = \int_0^T \left\{ \frac{\partial H}{\partial x} \frac{dx(t, \epsilon, p)}{d\epsilon} + \frac{\partial H}{\partial u} p(t) + \dot{\lambda}(t) \frac{dx(t, \epsilon, p)}{d\epsilon} \right\} dt + \frac{\partial h(x(T))}{\partial x} \frac{dx(T, \epsilon, p)}{d\epsilon} - \lambda(t) \frac{dx(T, \epsilon, p)}{d\epsilon}$$

$$\frac{d\bar{J}(\epsilon)}{d\epsilon} = \int_0^T \left\{ \left(\frac{\partial H}{\partial x} + \dot{\lambda}(t) \right) \frac{dx(t, \epsilon, p)}{d\epsilon} + \frac{\partial H}{\partial u} p(t) \right\} dt + \left(\frac{\partial h(x(T))}{\partial x} - \lambda(T) \right) \frac{dx(T, \epsilon, p)}{d\epsilon}$$

✓ $\frac{d\bar{J}(\epsilon)}{d\epsilon} = 0$ at $\epsilon = 0$ if we choose $\lambda(t)$

$$\frac{\partial H(t, x^*, u^*, \lambda)}{\partial x} + \dot{\lambda}(t) = 0 \text{ with } \frac{\partial h(x(T))}{\partial x} - \lambda(T)$$

✓ $\frac{d\bar{J}(\epsilon)}{d\epsilon} = 0$ at $\epsilon = 0$ if and only if

$$\int_0^T \frac{\partial H(t, x^*, u^*, \lambda)}{\partial u} p(t) dt = 0$$

- This should hold for any continuous function $p(t)$ on $[0, T]$. Choosing $p(t) = \frac{\partial H^T(t, x^*, u^*, \lambda)}{\partial u}$ shows that another necessary condition

$$\frac{\partial H(t, x^*, u^*, \lambda)}{\partial u} = 0$$

The Euler-Lagrange Equation

- We derive the first-order necessary conditions for the basic dynamic optimization problem to find a control function $u(\cdot)$ that minimizes the cost functional

$$J(u) = \int_0^T g(t, x(t), u(t)) dt + h(x(T)) \quad \text{where } h(x(T)) = q(T, x(T))$$

where the state variable $x(t)$ satisfies the differential equation:

$$\dot{x}(t) = f(t, x(t), u(t)), \quad x(0) = x_0$$

Theorem

If $u^*(t) \in U$ is a control that yields a local minimum for the cost function above and $x^*(t)$ and $\lambda^*(t)$ are the corresponding state and costate, then it is necessary that

$$\begin{aligned} \dot{x}(t) &= f(t, x^*, u^*) \left(= \frac{\partial H(t, x^*, u^*, \lambda)}{\partial \lambda} \right), & x^*(0) &= x_0 \\ \dot{\lambda}^*(t) &= \frac{\partial H(t, x^*, u^*, \lambda^*)}{\partial x}; & \lambda^*(T) &= \frac{\partial h(x^*(T))}{\partial x} \\ \frac{\partial H(t, x^*, u^*, \lambda^*)}{\partial u} &= 0 \end{aligned}$$

- The dynamic optimization problem has been reduced to a static optimization problem which should hold at every single instant of time

From the Euler Lagrange to Minimum Principle (Generalization)

- **Euler-Lagrange method** assumes that
 - ✓ $u(t)$ could (in principle) to be chosen arbitrarily in R^m
 - ✓ $u(t)$ is continuous for all $t \in [0, T]$
- However, these assumptions are too restrictive, thus a more general problem setting is required
- **Pontryagin Minimum principle** assumes that
 - ✓ There is a subset $U \subset R^m$ such that for all $t \in [0, T], u(t) \in U$
 - ✓ This is the next generalization of the Euler-Lagrange method

The Minimum Principle

Consider the following optimal control problem defined by

$$L(u) = \int_0^T g(t, x(t), u(t)) dt + q(T, x(T))$$

where the state variable $x(t)$ satisfies the differential equation:

$$\dot{x}(t) = f(t, x(t), u(t)), \quad x(0) = x_0, \quad t \geq 0$$

Theorem (The minimum principle)

In the continuous time dynamic system defined by equation (1) and (2), optimal control $u^*(t)$ and corresponding trajectory $x^*(t)$ satisfy following equations:

$$H(t, \lambda, x, u) := g(t, x, u) + \lambda(t)f(t, x, u)$$

$$\dot{x}^*(t) = f(t, x^*, u^*) \left(= \frac{\partial H(t, x^*, u^*, \lambda)}{\partial \lambda} \right), x^*(0) = x_0;$$

$$\dot{\lambda}^*(t) = -\frac{\partial H(t, \lambda, x^*, u^*)}{\partial x}; \lambda^*(T) = \frac{\partial h(x^*(T))}{\partial x}$$

$$u^*(t) = \operatorname{argmin}_{u \in U} H(t, \lambda^*, x^*, u)$$

Proof of The Minimum Principle

Proof)

We start with Hamilton-Jacobi-Bellman equation,

$$\nabla_t V(t, x) + \min_{u \in U} [\nabla_x V(t, x) f(t, x, u) + g(t, x, u)] = 0$$

Let $u^* = \operatorname{argmin}_{u \in U} [\nabla_x V(t, x) \cdot f(t, x, u) + g(t, x, u)]$ then

$$0 = \nabla_x V(t, x) f(t, x, u^*) + g(t, x, u^*) + \nabla_t V(t, x)$$

Derivatives w.r.t. x and t yields

$$0 = \nabla_{xx}^2 V(t, x) f(t, x, u^*) + \nabla_x V(t, x) \nabla_x f(t, x, u^*) + \nabla_x g(t, x, u^*) + \nabla_{xt}^2 V(t, x) \cdots (*)$$

$$0 = \nabla_{xt}^2 V(t, x) \cdot f(t, x, u^*) + \nabla_{tt}^2 V(t, x) \cdots (**)$$

(*) and (**) holds for all (t, x) .

Let us specialize them along an optimal state and trajectory $x^*(t), u^*(t)$, then we have

Proof of The Minimum Principle

Proof)

$$0 = \nabla_{xx}^2 V(t, x) f(t, x, u^*) + \nabla_x V(t, x) \nabla_x f(t, x, u^*) + \nabla_x g(t, x, u^*) + \nabla_{xt}^2 V(t, x) \cdots (*)$$

$$0 = \nabla_{xt}^2 V(t, x) \cdot f(t, x, u^*) + \nabla_{tt}^2 V(t, x) \cdots (**)$$

the term $\nabla_{xx}^2 V(t, x^*) f(t, x^*, u^*) + \nabla_{xt}^2 V(t, x^*)$ in (*) is equal the following total derivative w.r.t. t

$$\frac{d}{dt} (\nabla_x V(t, x^*)), \quad (\text{let } \nabla_x V(t, x^*) := p(t))$$

(*) becomes $\dot{p}(t) = -\nabla_x f(x^*, u^*) p(t) - \nabla_x g(x^*, u^*)$

Similarly, the term $\nabla_{xt}^2 V(t, x^*) f(t, x^*, u^*) + \nabla_{tt}^2 V(t, x^*)$ in (**) is equal the total derivative

$$\frac{d}{dt} (\nabla_t V(t, x^*)), \quad (\text{let } \nabla_t V(t, x^*) := p_0(t))$$

and (**) becomes $\dot{p}_0(t) = 0 \rightarrow p_0(t) = \text{constant}$

We have boundary condition

$$p(T) = \nabla_x V(T(x^*), x^*) = \nabla_x q(T(x^*), x^*)$$

Proof of The Minimum Principle

Proof)

So, the original HJB $u^* = \operatorname{argmin}_u [\nabla_x V(t, x) f(t, x, u) + g(t, x, u)]$ become

$$u^* = \operatorname{argmin}_{u \in U} [p(t) \cdot f(t, x, u) + g(t, x, u)]$$

Define $H(x, u, p) := p(t)f(t, x, u) + g(t, x, u)$, then system and ad joint equations can be written in terms of Hamiltonian

$$\dot{x}^*(t) = \nabla_p H(x^*, u^*, p), \quad \dot{p}(t) = -\nabla_x H(x^*, u^*, p)$$

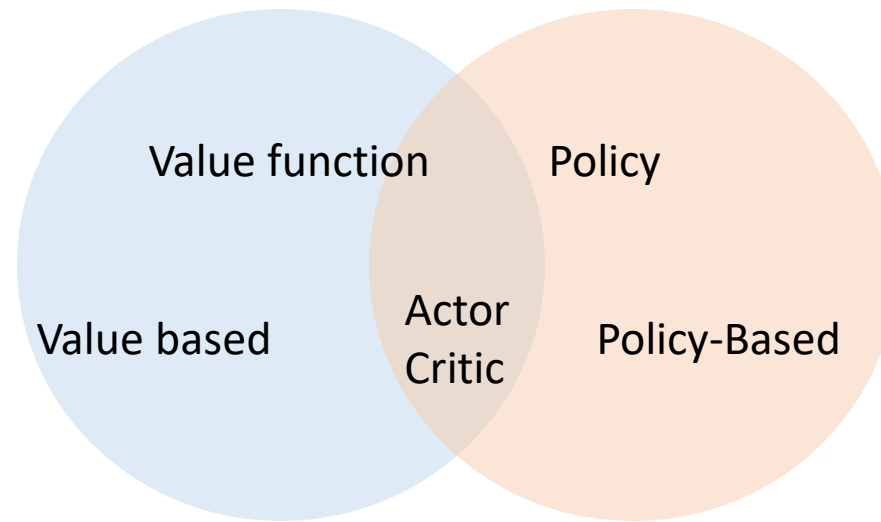
■

Comments (Minimum principle v.s. HJB)

- When satisfied along a trajectory, Pontryagin's minimum principle is a ***necessary condition, not sufficient contrition***, for an optimum.
- The Hamilton–Jacobi–Bellman equation provides a ***necessary and sufficient*** condition for an optimum, but this condition must be satisfied over the whole of the state space.
- While the Hamilton-Jacobi-Bellman equation admits a straightforward extension to stochastic optimal control problems, the minimum principle does not
 - DP not only solves the original optimization problem, but tells us even ***at arbitrary times and arbitrary states*** what the best action is
 - Having the optimal policy available as a function of times and state (“**feedback**”) is often viewed as even better than having it available only as a function of time (“**open-loop form**”)
- The Minimum Principle requires solving an ODE with split boundary conditions. It is not trivial to solve, but easier than solving a PDE in the HJB.

Policy Gradient Based Reinforcement Learning

Overview



Motivation

- One of the primary goals of the field of artificial intelligence is to solve complex tasks from unprocessed, high-dimensional, sensory input.
- Recent progresses in RL have shown the possibilities
 - ✓ Deep Q Network for Atari games and
 - ✓ AlphaGo
- Although DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces

Policy Gradient

- Policy Gradient methods learn the policy directly with a parameterized function respect to θ , $\pi_\theta(a|s)$.
- The reward function can be defined as (in continuous case):

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)$$

- ✓ where $d^\pi(s)$ is stationary distribution of Markov chain for π_θ (on-policy state distribution under π)

$$d^\pi(s) = \lim_{t \rightarrow \infty} P(s_t = s | s_0, \pi_\theta)$$

- ✓ $d^\pi = d^{\pi_\theta}$; $V^\pi = V^{\pi_\theta}$; $Q^\pi = Q^{\pi_\theta}$ for simplicity

- We can find the best θ^* using *gradient ascent* such that

$$\theta^* = \operatorname{argmax}_{\theta} J(\theta)$$

- It is natural to expect policy-based methods are more useful in the **continuous space**
 - ✓ There is an infinite number of actions and (or) states to estimate the values
 - ✓ Difficult to compute $\operatorname{argmax}_{a \in \mathcal{A}} Q^\pi(s, a)$ requiring full scan of the action space

How to compute the gradient $\nabla J(\theta)$?

- Gradient can be computed by perturbing θ by a small amount ϵ in the k -th dimension.

$$\frac{\partial J(\theta)}{\partial \theta_k} \approx \frac{J(\theta + \epsilon u_k) - J(\theta)}{\epsilon}$$

- ✓ It works even when $J(\theta)$ is not differentiable, but very slow
- Is there more efficient method?

Policy Gradient Theorem

- The reward function is

$$J(\theta) = \sum_{s \in \mathcal{S}} d^\pi(s) V^\pi(s) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a)$$

- Computing the gradient $\nabla J(\theta)$ is tricky because it depends on both the action selection and the stationary distribution of states following the target selection behavior
- The **policy gradient theorem** makes the computation of $\nabla J(\theta)$ simple

$$\begin{aligned} \nabla_\theta J(\theta) &= \nabla_\theta \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) Q^\pi(s, a) \\ &\propto \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla_\theta \pi_\theta(a|s) Q^\pi(s, a) && \text{gradient does not depends on } Q^\pi \text{ (will be proved)} \\ &= \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} Q^\pi(s, a) \\ &= \sum_s d^\pi(s) \sum_{a \in \mathcal{A}} \pi_\theta(a|s) \nabla \ln \pi_\theta(a|s) Q^\pi(s, a) && \because (\ln x)' = 1/x \\ &= \mathbb{E}_\pi[\nabla \ln \pi_\theta(a|s) Q^\pi(s, a)] \end{aligned}$$

- ✓ where \mathbb{E}_π refers to $\mathbb{E}_{s \sim d^\pi, a \sim \pi_\theta}$ when both state and action distributions follow the policy π_θ (**on-policy**)

Policy Gradient Theorem Proof

$$\nabla_{\theta} V^{\pi}(s) = \nabla_{\theta} \left(\sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q^{\pi}(s, a) \right)$$

$$= \sum_{a \in \mathcal{A}} \left(\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \nabla_{\theta} Q^{\pi}(s, a) \right)$$

∴ derivative product rule

$$= \sum_{a \in \mathcal{A}} \left(\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \nabla_{\theta} \sum_{s', r} P(s', r|s, a) (r + V^{\pi}(s')) \right)$$

∴ Extend Q^{π} with further state value

$$= \sum_{a \in \mathcal{A}} \left(\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \sum_{s', r} P(s', r|s, a) \nabla_{\theta} V^{\pi}(s') \right)$$

∴ $P(s', r|s, a)$ is not a function of θ

$$= \sum_{a \in \mathcal{A}} \left(\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \sum_{s'} P(s'|s, a) \nabla_{\theta} V^{\pi}(s') \right)$$

∴ $P(s' |s, a) = \sum_r P(s', r|s, a)$

$$\nabla_{\theta} V^{\pi}(s) = \sum_{a \in \mathcal{A}} \left(\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \sum_{s'} P(s'|s, a) \nabla_{\theta} V^{\pi}(s') \right)$$

✓ a nice recursive form and the future state value function $V^{\pi}(s')$ can be repeated unrolled as will be shown next slide

Policy Gradient Theorem Proof

$$\nabla_{\theta} V^{\pi}(s) = \sum_{a \in \mathcal{A}} \left(\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \sum_{s'} P(s'|s, a) \nabla_{\theta} V^{\pi}(s') \right)$$

$$= \phi(s) + \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) \sum_{s'} P(s'|s, a) \nabla_{\theta} V^{\pi}(s')$$

$$= \phi(s) + \sum_{s'} \sum_a \pi_{\theta}(a|s) P(s'|s, a) \nabla_{\theta} V^{\pi}(s')$$

$$= \phi(s) + \sum_{s'} \rho^{\pi}(s \rightarrow s', 1) \nabla_{\theta} V^{\pi}(s')$$

$$= \phi(s) + \sum_{s'} \rho^{\pi}(s \rightarrow s', 1) \left[\phi(s') + \sum_{s''} \rho^{\pi}(s' \rightarrow s'', 1) \nabla_{\theta} V^{\pi}(s'') \right]$$

$$= \phi(s) + \sum_{s'} \rho^{\pi}(s \rightarrow s', 1) \phi(s') + \sum_{s''} \rho^{\pi}(s \rightarrow s'', 2) \nabla_{\theta} V^{\pi}(s'')$$

$$= \phi(s) + \sum_{s'} \rho^{\pi}(s \rightarrow s', 1) \phi(s') + \sum_{s''} \rho^{\pi}(s \rightarrow s'', 2) \nabla_{\theta} V^{\pi}(s'') + \sum_{s'''} \rho^{\pi}(s \rightarrow s''', 3) \nabla_{\theta} V^{\pi}(s''')$$

$$= \dots; \text{repeatedly unrolling the part of } \nabla_{\theta} V^{\pi}(\cdot)$$

$$= \sum_{x \in \mathcal{S}} \sum_k^{\infty} \rho^{\pi}(s \rightarrow x, k) \phi(x)$$

$$\phi(s) = \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a)$$

Policy Gradient Theorem Proof

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \nabla_{\theta} V^{\pi}(s_0) \\&= \sum_s \sum_k^{\infty} \rho^{\pi}(s_0 \rightarrow s, k) \phi(s) \\&= \sum_s \eta(s) \phi(s) \\&= \left(\sum_s \eta(s) \right) \sum_s \frac{\eta(s)}{(\sum_s \eta(s))} \phi(s) \\&\propto \sum_s \frac{\eta(s)}{(\sum_s \eta(s))} \phi(s) \\&= \sum_s d^{\pi}(s) \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) \\&= \sum_s d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} Q^{\pi}(s, a) \\&= \sum_s d^{\pi}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) \nabla_{\theta} \ln \pi_{\theta}(a|s) Q^{\pi}(s, a) \\&= \mathbb{E}_{\pi} [\nabla_{\theta} \ln \pi_{\theta}(a|s) Q^{\pi}(s, a)]\end{aligned}$$

$$\because \eta(s) = \sum_{k=0}^{\infty} \rho^{\pi}(s_0 \rightarrow s, k)$$

\because Normalize $\eta(s), s \in \mathcal{S}$ to be a probability dist.

$\because \sum_s \eta(s)$ is a constant

$$\because \phi(s) = \sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a)$$

\mathbb{E}_{π} refers to $\mathbb{E}_{s \sim d^{\pi}, a \sim \pi_{\theta}}$ (**on-policy**)

Generalized Advantage Estimation (GAE) (Schulman et al., 2016)

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

- The policy gradient theorem lays the theoretical foundation for various policy gradient algorithms.
- This vanilla policy gradient update has no bias but high variance. Many following algorithms were proposed to reduce the variance while keeping the bias unchanged.
 - ✓ Reinforce
 - ✓ Actor-Critic
 - ✓ Off-Policy Policy Gradient
 - ✓ Asynchronous Advantage Actor Critic (Mnih et al., 2016)
 - ✓ Deterministic Policy Gradient (DPG)
 - ✓ Deep Deterministic Policy Gradient (DDPG) (Lillicrap, et al., 2015)
 - ✓ Trust region policy optimization (TRPO) (Schulman, et al., 2015)
 - ✓ Proximal Policy Optimization (PPO) (Schulman, et al., 2017)
 - ✓ Soft Actor Critic (SAC) (Haarnoja et al., 2018)
 - ✓ Multi-Agent Deep Deterministic Policy Gradient (MADDPG) (Lowe et al., 2017)

REINFORCE Algorithm

- REINFORCE (Monte-Carlo policy gradient) relies on an estimated return by Monte-Carlo methods using episode samples to update the policy parameter θ
- REINFORCE works because the expectation of the sample gradient is equal to the actual gradient:

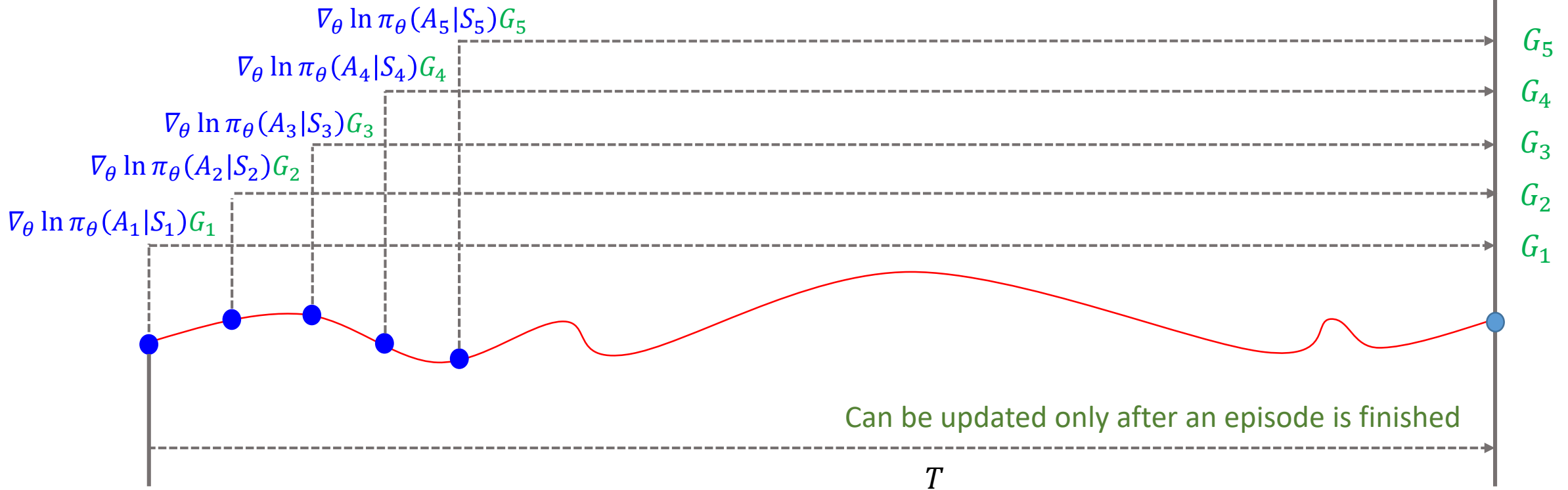
$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi}[\nabla_{\theta} \ln \pi_{\theta}(a|s) Q^{\pi}(s, a)] \\ &= \mathbb{E}_{\pi}[\nabla_{\theta} \ln \pi_{\theta}(A_t|S_t) G_t] \quad \because Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t|S_t, A_t]\end{aligned}$$

- ✓ Measure G_t from real sample trajectories and use that to update policy gradient
- ✓ It relies on a full trajectory and that's why it is a Monte-Carlo method

1. Initialize the policy parameter θ at random.
2. Generate one trajectory on policy π_{θ} : $S_1, A_1, R_2, S_2, A_2, \dots, S_T$.
3. For $t=1, 2, \dots, T$:
 1. Estimate the the return G_t ;
 2. Update policy parameters: $\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_{\theta} \ln \pi_{\theta}(A_t|S_t)$

- A widely used variation of REINFORCE is to **subtract a baseline value** from the return G_t to reduce the variance of gradient estimation while keeping the bias unchanged

REINFORCE Algorithm



- Update per every action
- Update per every episode
- Update per episode & batch

$$\theta \leftarrow \theta + \alpha \gamma^t \nabla_{\theta} \ln \pi_{\theta}(A_t|S_t) G_t$$

$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^T \gamma^t \nabla_{\theta} \ln \pi_{\theta}(A_t|S_t) G_t \right]$$

$$\theta \leftarrow \theta + \alpha \left[\frac{1}{N} \sum_{i=1}^N \sum_{t=1}^T \gamma^t \nabla_{\theta} \ln \pi_{\theta}(A_t^{(i)}|S_t^{(i)}) G_t^{(i)} \right]$$

Actor-Critic

- It makes a lot of sense to learn the value function in addition to the policy, since knowing the value function can assist the policy update, such as by reducing gradient variance in vanilla policy gradients
- Two main components in policy gradient are the policy model and the value function.
 - Critic updates the value function parameters w and depending on the algorithm it could be action value $Q_w(s, a)$ or state-value $V_w(s)$
 - Actor updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic

1. Initialize s, θ, w at random; sample $a \sim \pi_\theta(a|s)$.
2. For $t = 1 \dots T$:
 1. Sample reward $r_t \sim R(s, a)$ and next state $s' \sim P(s'|s, a)$;
 2. Then sample the next action $a' \sim \pi_\theta(a'|s')$;
 3. Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta Q_w(s, a) \nabla_\theta \ln \pi_\theta(a|s)$;
 4. Compute the correction (TD error) for action-value at time t :
$$\delta_t = r_t + \gamma Q_w(s', a') - Q_w(s, a)$$
and use it to update the parameters of action-value function:
$$w \leftarrow w + \alpha_w \delta_t \nabla_w Q_w(s, a)$$
 5. Update $a \leftarrow a'$ and $s \leftarrow s'$.

(on-policy learning)

Off-Policy Policy Gradient (Degris, White & Sutton, 2012).

- Both REINFORCE and the vanilla version of actor-critic method are on-policy:
 - ✓ training samples are collected according to the target policy — the very same policy that we try to optimize for.
- Off policy methods, however, result in several additional advantages:
 - ✓ The off-policy approach does not require full trajectories and can reuse any past episodes (experience replay) for much better sample efficiency.
 - ✓ The sample collection follows a *behavior policy* different from the target policy, bringing *better exploration*

Off-Policy Policy Gradient

$$J(\theta) = \sum_{s \in \mathcal{S}} d^{\beta}(s) \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q^{\pi}(s, a) = \mathbb{E}_{s \sim d^{\beta}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q^{\pi}(s, a) \right]$$

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \nabla_{\theta} \mathbb{E}_{s \sim d^{\beta}} \left[\sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) Q^{\pi}(s, a) \right] \\ &= \mathbb{E}_{s \sim d^{\beta}} \left[\sum_{a \in \mathcal{A}} (\nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) + \pi_{\theta}(a|s) \nabla_{\theta} Q^{\pi}(s, a)) \right] \end{aligned}$$

Derivative product rule

$$\approx \mathbb{E}_{s \sim d^{\beta}} \left[\sum_{a \in \mathcal{A}} \nabla_{\theta} \pi_{\theta}(a|s) Q^{\pi}(s, a) \right]$$

Ignore $\pi_{\theta}(a|s) \nabla_{\theta} Q^{\pi}(s, a)$

$$= \mathbb{E}_{s \sim d^{\beta}} \left[\sum_{a \in \mathcal{A}} \beta(a|s) \frac{\pi_{\theta}(a|s)}{\beta(a|s)} \frac{\nabla_{\theta} \pi_{\theta}(a|s)}{\pi_{\theta}(a|s)} Q^{\pi}(s, a) \right]$$

$$= \mathbb{E}_{\beta} \left[\frac{\pi_{\theta}(a|s)}{\beta(a|s)} \nabla_{\theta} \ln \pi_{\theta}(a|s) Q^{\pi}(s, a) \right]$$

Importance weight

- if we use an approximated gradient with the gradient of Q ignored, we still guarantee the policy improvement and eventually achieve the true local minimum (Degris, White & Sutton, 2012).
- In summary, when applying policy gradient in the off-policy setting, we can simply adjust it with a weighted sum and the weight is the ratio of the target policy to the behavior policy,

Asynchronous Advantage Actor Critic (Mnih et al., 2016)

- Deep RL algorithms based on experience replay have achieved unprecedented success in challenging domains such as Atari 2600. However, experience replay has several drawbacks:
 - It uses **more memory** and commutation per real interaction
 - It requires off-policy learning algorithms that can update from data generated by an **older policy**
- Instead of experience replay, A3C asynchronously executes **multiple agents in parallel**, on multiple instances of the environment
 - Parallelism decorrelates the agents' data input a more stationary processes, since at any given time step the parallel agents will be experiencing a variety of different states.
 - Parallelism stabilizes learning, playing a role of experience replay buffer
- Make the observation that multiple actor-learners running in parallel are likely to be exploring different parts of the environment
 - One can explicitly use different exploration policies in each actor-learners to maximize the diversity
 - Make samples less correlated

Asynchronous Advantage Actor Critic (Mnih et al., 2016)

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

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$

// Assume thread-specific parameter vectors θ' and θ'_v

Initialize thread step counter $t \leftarrow 1$

repeat

Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.

Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$

$t_{start} = t$

Get state s_t

repeat

Perform a_t according to policy $\pi(a_t|s_t; \theta')$

Receive reward r_t and new state s_{t+1}

$t \leftarrow t + 1$

$T \leftarrow T + 1$

until terminal s_t **or** $t - t_{start} == t_{max}$

$R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t // \text{ Bootstrap from last state} \end{cases}$

for $i \in \{t - 1, \dots, t_{start}\}$ **do**

$R \leftarrow r_i + \gamma R$


Accumulate gradients wrt θ' : $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i; \theta')(R - V(s_i; \theta'_v))$

Accumulate gradients wrt θ'_v : $d\theta_v \leftarrow d\theta_v + \partial (R - V(s_i; \theta'_v))^2 / \partial \theta'_v$

end for

Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.

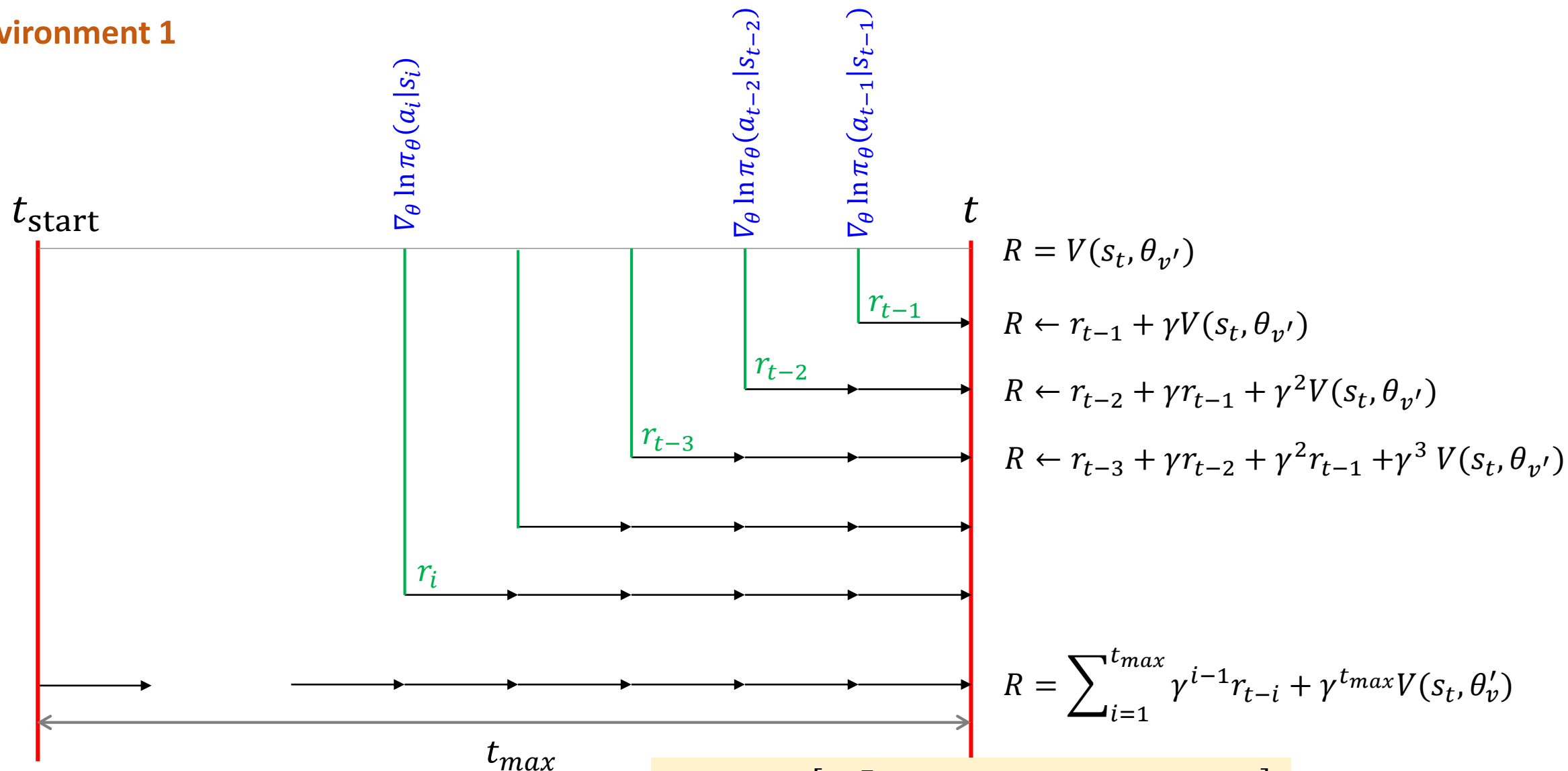
until $T > T_{max}$


$$R = \sum_{i=1}^{t_{max}} \gamma^{i-1} r_{t-i} + \gamma^{t_{max}} V(s_t, \theta'_v)$$

- The gradient accumulation step can be considered as a parallelized reformation of mini-batch-based stochastic gradient update

Asynchronous Advantage Actor Critic (Mnih et al., 2016)

Environment 1



$$\theta \leftarrow \theta + \alpha \left[\sum_{t=1}^T \gamma^t \nabla_{\theta} \ln \pi_{\theta}(A_t | S_t) (R_t - V(s_t)) \right]$$

Asynchronous Advantage Actor Critic (Mnih et al., 2016)

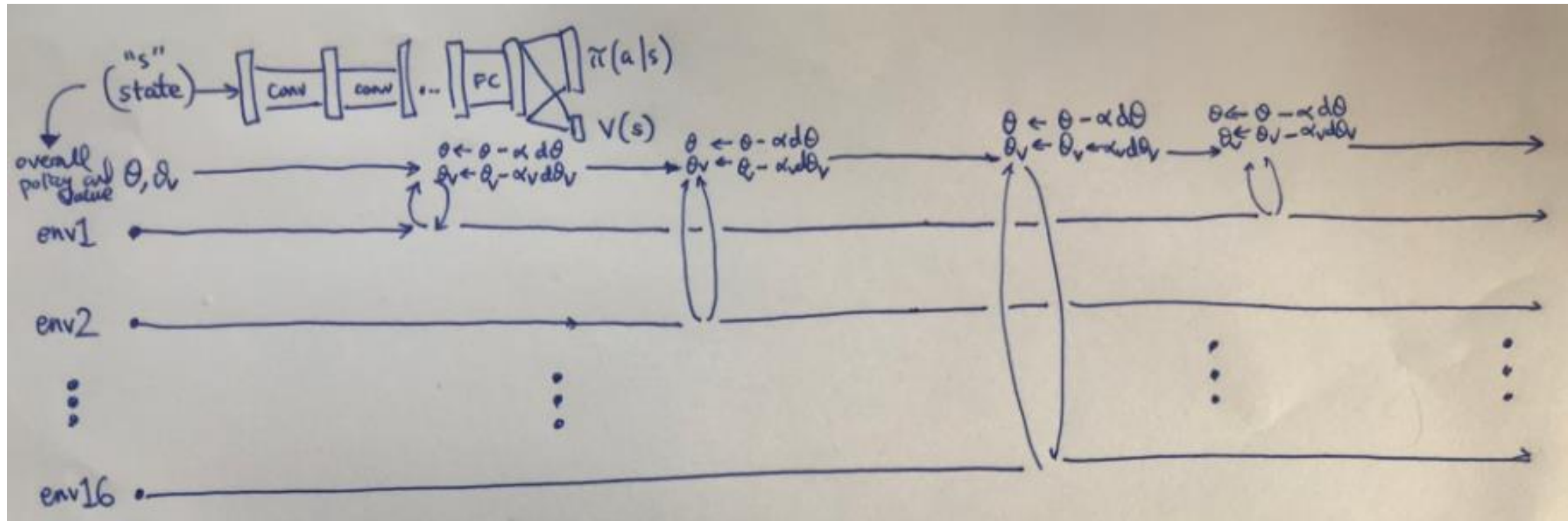


Figure from : <https://danieltakeshi.github.io/2018/06/28/a2c-a3c/>

Deterministic Policy Gradient (DPG) (Silver et al., 2014)

- So far, the policy function $\pi(\cdot | s)$ is always modeled as a probability distribution over actions a given the current state s and thus it is stochastic.
- Deterministic policy gradient (DPG) instead models the policy as a deterministic decision: $a = \mu(s)$.
- How can you calculate the gradient of the policy function when it outputs a single action?
- Let's define few notations to facilitate the discussion:
 - $\rho_0(s)$: The initial distribution over states
 - $\rho^\mu(s \rightarrow s', k)$: Starting from state s , the visitation probability density at state s' after moving k steps

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

Deterministic Policy Gradient (DPG) (Silver et al., 2014)

- The objective function to optimize for is listed as follows:

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

- Deterministic policy gradient theorem:

$$\begin{aligned} \nabla_\theta J(\theta) &= \int_S \rho^\mu(s) \nabla_a Q^\mu(s, a) \nabla_\theta \mu_\theta(s) \Big|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s \sim \rho^\mu} \left[\nabla_a Q^\mu(s, a) \nabla_\theta \mu_\theta(s) \Big|_{a=\mu_\theta(s)} \right] \end{aligned}$$

- We can consider **the deterministic policy as a special case of the stochastic one**, when the probability distribution contains only one extreme non-zero value over one action.
- We expect the stochastic policy to require more samples as it integrates the data over the whole state and action space.

Deterministic Policy Gradient (DPG) (Silver et al., 2014)

- The deterministic policy gradient theorem can be plugged into common policy gradient frameworks.

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

- on-policy actor-critic algorithm with deterministic policy $a = \mu_\theta(s)$

$$\begin{aligned}\delta_t &= R_t + \gamma Q_w(s_{t+1}, a_{t+1}) - Q_w(s_t, a_t) \\ w_{t+1} &= w_t + \alpha_w \delta_t \nabla_w Q_w(s_t, a_t) \\ \theta_{t+1} &= \theta_t + \alpha_\theta \nabla_a Q_w(s_t, a_t) \nabla_\theta \mu_\theta(s) \Big|_{a_t = \mu_\theta(s_t)}\end{aligned}$$

- However, unless there is sufficient noise in the environment, it is very hard to guarantee enough exploration due to the determinacy of the policy.
 - We can either add noise into the policy (ironically this makes it nondeterministic!) or
 - learn it off-policy way by following a different stochastic behavior policy to collect samples.

Deterministic Policy Gradient (DPG) (Silver et al., 2014)

- Let's consider the off-policy approach.
- The training trajectories are generated by a stochastic policy $\beta(a|s)$
 - ✓ the state distribution follows the corresponding discounted state density ρ^β
- The objective function and the gradient of it can be computed as:

$$\begin{aligned} J(\theta) &= \int_S \rho^\beta(s) Q^\mu(s, \mu_\theta(s)) ds \\ \nabla_\theta J(\theta) &= \int_S \rho^\beta(s) \nabla_a Q^\mu(s, a) \nabla_\theta \mu_\theta(s) \Big|_{a=\mu_\theta(s)} ds \\ &= \mathbb{E}_{s \sim \rho^\beta} \left[\nabla_a Q^\mu(s, a) \nabla_\theta \mu_\theta(s) \Big|_{a=\mu_\theta(s)} \right] \end{aligned}$$

- Because the policy is deterministic, we only need $Q^\mu(s, \mu_\theta(s))$ rather than $\sum_a \pi(a|s) Q^\pi(s, a)$ as the estimated reward of a given state s .
 - ✓ Because the deterministic policy gradient removes the integral over actions, we can **avoid importance sampling even when using off-policy approach**.
 - ✓ We expect the deterministic policy **requires less samples** than stochastic policy as it integrates the data over only state space.

Deep Deterministic Policy Gradient (DDPG) (Lillicrap, et al., 2015)

- Deep Deterministic Policy Gradient (DDPG), is a model-free off-policy actor-critic algorithm, combining DPG with DQN.
 - Recall that DQN (Deep Q-Network) stabilizes the learning of Q-function by experience replay and the frozen target network.
- The original DQN works in discrete action space, and DDPG extends it to continuous space with the actor-critic framework while learning a deterministic policy.
- In order to do better exploration, an exploration policy μ' is constructed by adding noise \mathcal{N} :

$$\mu'(s) = \mu_{\theta}(s) + \mathcal{N}$$

- In addition, DDPG does soft updates (“conservative policy iteration”) on the parameters of both actor and critic, with $\tau \ll 1$

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

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

Deep Deterministic Policy Gradient (DDPG) (Lillicrap, et al., 2015)

Algorithm 1 DDPG algorithm

Randomly initialize critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ .

Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$

Initialize replay buffer R

for episode = 1, M **do**

 Initialize a random process \mathcal{N} for action exploration

 Receive initial observation state s_1

for t = 1, T **do**

 Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise

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

 Store transition (s_t, a_t, r_t, s_{t+1}) in R

 Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R

 Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'})$

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

 Update the actor policy using the sampled policy gradient:

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

 Update the target networks:

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

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

end for
end for

for every
transition

Gradually changing target networks

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

Approximate it using minibatch samples

Multi-agent DDPG (MADDPG) (Lowe et al., 2017)

- Multi-agent DDPG (MADDPG) extends DDPG to an environment where multiple agents are coordinating to complete tasks with only local information.
- In the viewpoint of one agent, the environment is non-stationary as policies of other agents are quickly upgraded and remain unknown.
 - MADDPG is an actor-critic model redesigned particularly for handling such a non-stationary environment and interactions between agents.
- The problem can be formalized in the multi-agent version of MDP, also known as Markov games

\mathcal{N} : set of agents

\mathcal{S} : set of joint states

\mathcal{A}_i : set of possible action for agent i ; $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_N$ set of joint action

\mathcal{O}_i : set of observation for agent i ; $\mathcal{O} = \mathcal{O}_1 \times \cdots \times \mathcal{O}_N$: set of joint observation

$\mathcal{T}: \mathcal{S} \times \mathcal{A}_1 \times \cdots \times \mathcal{A}_N \mapsto \mathcal{S}$ transition function

$\pi_{\theta_i}: \mathcal{O}_i \times \mathcal{A}_i \mapsto [0,1]$ stochastic policy of agent i

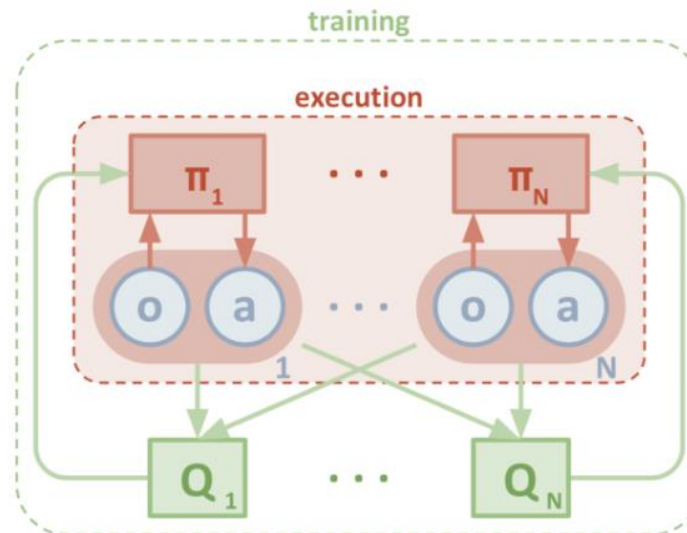
$\mu_{\theta_i}: \mathcal{O}_i \mapsto \mathcal{A}_i$ deterministic policy of agent i

Multi-agent DDPG (MADDPG) (Lowe et al., 2017)

- The critic in MADDPG learns a centralized action-value function for agent i

$$Q_i^{\vec{\mu}}(\vec{o}, a_1, \dots, a_N)$$

- ✓ $\vec{\mu} = \mu_1, \dots, \mu_N$:joint policies
- ✓ $\vec{o} = o_1, \dots, o_N$:joint observations
- ✓ $\vec{a} = a_1, \dots, a_N$:joint actions
- Each $Q_i^{\vec{\mu}}$ is **learned separately** for $i = 1, \dots, N$ and therefore **multiple agents can have arbitrary reward structures**, including conflicting rewards in a competitive setting.
- Meanwhile multiple actors, one for each agent, are exploring and upgrading the policy parameters θ_i on their own.



Multi-agent DDPG (MADDPG) (Lowe et al., 2017)

- **Actor update:**

$$\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{(\vec{o}, a) \sim D} \left[\nabla_{a_i} Q_i^{\vec{\mu}}(\vec{o}, a_1, \dots, a_N) \nabla_{\theta_i} \mu_{\theta_i}(o_i) \Big|_{a_i = \mu_{\theta_i}(o_i)} \right]$$

- ✓ D is the memory buffer for experience replay, containing multiple episode samples $(\vec{o}, a_1, \dots, a_N, r_1, \dots, r_N, \vec{o}')$: given current observation \vec{o} , agents take joint actions a_1, \dots, a_N and get rewards r_1, \dots, r_N , leading to the new observation \vec{o}'

- **Critic update:**

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(\vec{o}, a_1, \dots, a_N, r_1, \dots, r_N, \vec{o}') \sim D} \left[\left(Q_i^{\vec{\mu}}(\vec{o}, a_1, \dots, a_N) - y \right)^2 \right]$$

$$\text{where } y = r_i + \gamma Q_i^{\vec{\mu}'}(\vec{o}', a'_1, \dots, a'_N) \Big|_{a'_i = \mu'_{\theta_i}(o_i)}$$

- ✓ $\vec{\mu}'$ are the target policies with delayed softly-updated parameters.
- ✓ For each agent need to have $\vec{\mu} = \mu_1, \dots, \mu_N$ to compute the joint actions at the next observation \vec{o}'
- ✓ Each agent learns other agents policy its own during the learning process.
- ✓ Using the approximated policies, MADDPG still can learn efficiently although the inferred policies might not be accurate.

Multi-agent DDPG (MADDPG) (Lowe et al., 2017)

- To mitigate the high variance triggered by the interaction between competing or collaborating agents in the environment, MADDPG proposed one more element - policy ensembles:
 - ✓ Train K policies for one agent;
 - ✓ Pick a random policy for episode rollouts;
 - ✓ Take an ensemble of these K policies to do gradient update.
- In summary, MADDPG added three additional ingredients on top of DDPG to make it adapt to the multi-agent environment:
 - ✓ Centralized critic + decentralized actors;
 - ✓ Actors are able to use estimated policies of other agents for learning;
 - ✓ Policy ensembling is good for reducing variance.

Trust region policy optimization (TRPO) (Schulman, et al., 2015)

- To improve training stability, we should avoid parameter updates that change the policy too much at one step.
- Trust region policy optimization (TRPO) carries out this idea by enforcing a KL divergence constraint on the size of policy update at each iteration.
- **If off policy**, the objective function measures the total advantage over the state visitation distribution and actions, while the rollout is following a different behavior policy $\beta(a|s)$:

$$\begin{aligned} J(\theta) &= \sum_{s \in \mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a \in \mathcal{A}} \pi_{\theta}(a|s) \hat{A}_{\theta_{\text{old}}}(s, a) \\ &= \sum_{s \in \mathcal{S}} \rho^{\pi_{\theta_{\text{old}}}} \sum_{a \in \mathcal{A}} \beta(a|s) \frac{\pi_{\theta}(a|s)}{\beta(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a) \\ &= \mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}, a \sim \beta} \left[\frac{\pi_{\theta}(a|s)}{\beta(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a) \right] \end{aligned}$$

- ✓ θ_{old} is the policy parameters before the update and thus known to us. $\rho^{\pi_{\theta_{\text{old}}}}$ is defined similarly.
- ✓ $\beta(a|s)$ is the behavior policy for collecting trajectories

Trust region policy optimization (TRPO) (Schulman, et al., 2015)

- **If on policy**, the behavior policy is $\pi_{\theta}(a|s)$:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}, a \sim \beta} \left[\frac{\pi_{\theta}(a|s)}{\beta(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a) \right] \\ &= \mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}, a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a) \right] \end{aligned}$$

- TRPO aims to maximize the objective function $J(\theta)$ subject to, trust region constraint which enforces the distance between old and new policies measured by KL-divergence to be small enough, within a parameter δ

$$\mathbb{E}_{s \sim \rho^{\pi_{\theta_{\text{old}}}}} [D_{KL}(\pi_{\theta_{\text{old}}}(\cdot | s) \parallel \pi_{\theta}(\cdot | s))] \leq \delta$$

- In this way, the old and new policies would not diverge too much when this hard constraint is met.
 - While still, TRPO can guarantee a monotonic improvement over policy iteration (why?)

Proximal Policy Optimization (PPO) (Schulman, et al., 2017)

- Given that TRPO is relatively complicated and we still want to implement a similar constraint, proximal policy optimization (PPO) simplifies it by using a clipped surrogate objective while retaining similar performance.
- Denote the probability ratio between old and new policies as:

$$r(\theta) = \frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$$

- Then, the objective function of TRPO (on policy) becomes:

$$J^{\text{TRPO}}(\theta) = \mathbb{E} \left[\frac{\pi_{\theta}(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}_{\theta_{\text{old}}}(s, a) \right] = \mathbb{E} [r(\theta) \hat{A}_{\theta_{\text{old}}}(s, a)]$$

- Without a limitation on the distance between θ_{old} and θ , to maximize $J^{\text{TRPO}}(\theta)$ would lead to instability with extremely large parameter updates and big policy ratios.
- PPO imposes the constraint by forcing $r(\theta)$ to be within a small interval around 1, precisely $[1 - \epsilon, 1 + \epsilon]$

$$J^{\text{CLIP}}(\theta) = \mathbb{E} \left[\min \left(r(\theta) \hat{A}_{\theta_{\text{old}}}(s, a), \text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_{\theta_{\text{old}}}(s, a) \right) \right]$$

- ✓ The function $\text{clip}(r(\theta), 1 - \epsilon, 1 + \epsilon)$ clips the ratio $r(\theta)$ within $[1 - \epsilon, 1 + \epsilon]$.

Proximal Policy Optimization (PPO) (Schulman, et al., 2017)

- When applying PPO on the network architecture with shared parameters for both policy (actor) and value (critic) functions, the objective function is augmented composed of
 - ✓ the clipped reward
 - ✓ an error term on the value estimation
 - ✓ an entropy term to encourage sufficient exploration.

$$J^{\text{CLIP}'}(\theta) = \mathbb{E} \left[J^{\text{CLIP}}(\theta) - c_1 (V_\theta(s) - V_{\text{target}})^2 + c_2 H(s, \pi_\theta(\cdot)) \right]$$

c_1 and c_2 are two hyperparameters constants.

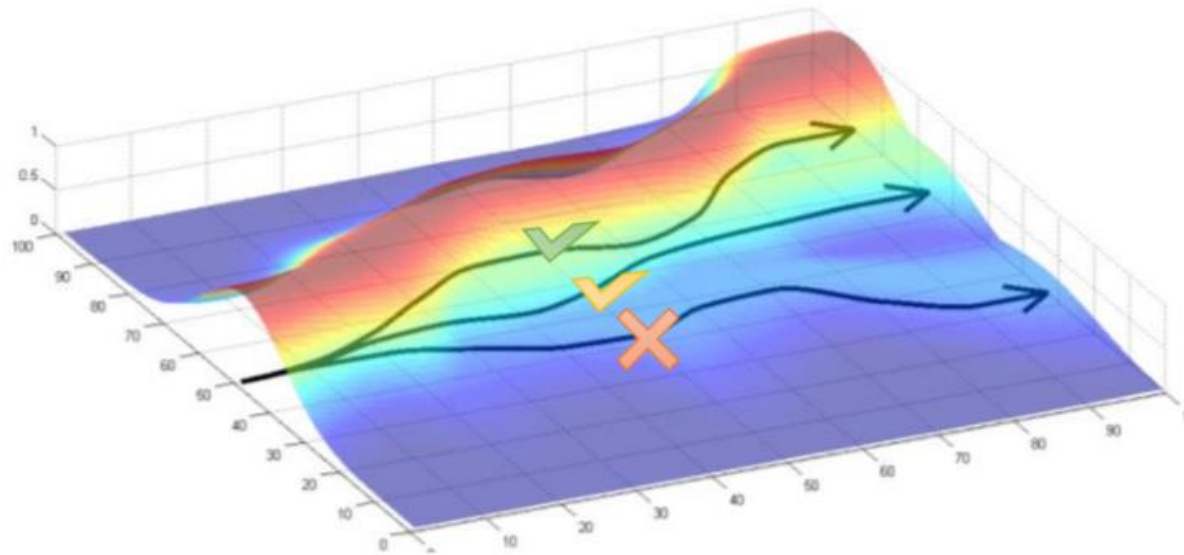
actor-critic with experience replay (ACER) (Wang, et al., 2017)

- ACER is an off-policy actor-critic model with experience replay,
 - ✓ greatly increasing the sample efficiency and decreasing the data correlation.
- ACER is A3C's off-policy counterpart.
- The major obstacle to making A3C off policy is how to control the stability of the off-policy estimator. ACER proposes three designs to overcome it:
 - ✓ Use Retrace Q-value estimation;
 - ✓ Truncate the importance weights with bias correction;
 - ✓ Apply efficient TRPO.

Will be updated later

Policy Gradient Algorithm From Different Angle

- We derived Policy Gradient Theorem using the concept of state value function to employ recursive formula on the basis of dynamic programming (Bellman equation)
- Not let's derive similar result without using DP approach but using ***episodic*** approach where agent engages in multiple trajectories in its environment



Policy Gradient Algorithm From Different Angle

- Let's define a trajectory τ of length T as

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, s_T)$$

- ✓ s_0 comes from the starting distribution of states
- ✓ $a_i \sim \pi_\theta(a_i | s_i)$ with π_θ parameterized **policy**
- ✓ $s_i \sim P(s_i | s_{i-1}, a_{i-1})$ with P dynamic model describing how **environment** changes
- The probability of trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \dots, s_{T-1}, a_{T-1}, s_T)$ can be expressed as

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

- We can compute

$$\begin{aligned} \nabla_\theta \log p(\tau) &= \nabla_\theta \log \left(\mu(s_0) \prod_{t=0}^{T-1} \pi_\theta(a_t | s_t) P(s_{t+1} | s_t, a_t) \right) \\ &= \nabla_\theta \left[\cancel{\log \mu(s_0)} + \sum_{t=0}^{T-1} (\log \pi_\theta(a_t | s_t) + \log \cancel{P(s_{t+1} | s_t, a_t)}) \right] \\ &= \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \end{aligned}$$

Policy Gradient Algorithm From Different Angle

- We want to maximize

$$\mathbb{E}_{\pi_{\theta}} \left[\sum_{t=0}^{T-1} r_t \right] = \mathbb{E}_{\tau \sim p_{\theta}} [R(\tau)]$$

\because we know τ is influenced by π_{θ}

- The gradient of objective is computed as

$$\begin{aligned} \nabla_{\theta} \mathbb{E}_{\tau \sim p_{\theta}} [R(\tau)] &= \nabla_{\theta} \int p_{\theta}(\tau) R(\tau) d\tau \\ &= \int \frac{p_{\theta}(\tau)}{p_{\theta}(\tau)} \nabla_{\theta} p_{\theta}(\tau) R(\tau) d\tau \\ &= \int p_{\theta}(\tau) \nabla_{\theta} \ln p_{\theta}(\tau) R(\tau) d\tau \\ &= \mathbb{E}_{\tau \sim p_{\theta}} [\nabla_{\theta} \ln p_{\theta}(\tau) R(\tau)] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) R(\tau) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t=0}^{T-1} r_t \right) \right] \end{aligned}$$

$$\because \nabla_{\theta} \log p(\tau) = \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

Policy Gradient Algorithm From Different Angle

- We have

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t=0}^{T-1} r_t \right) \right]$$

- ✓ The expectation is with respect to the policy function, so we can think $\tau \sim \pi_{\theta}$
- We need trajectories to get an empirical expectation, which estimates the actual expectation. The naïve way is to run the agent on batch of episodes

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathbb{E}_{\tau \in \mathcal{T}} [R(\tau)]$$

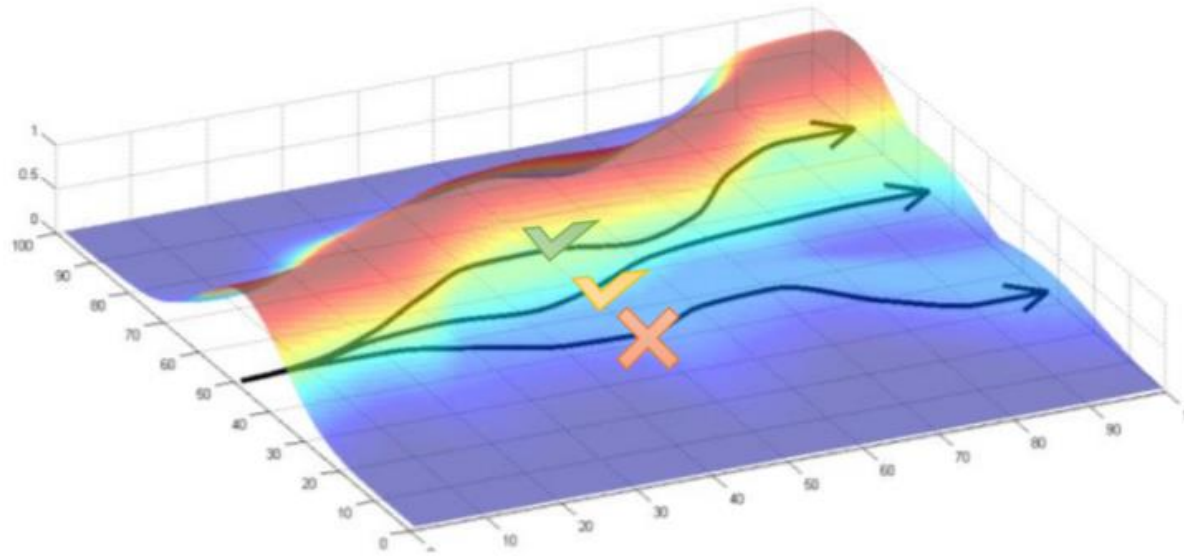
- ✓ where \mathcal{T} is a set of trajectories sampled

$$\nabla_{\theta} \mathbb{E}_{\tau \in \mathcal{T}} [R(\tau)] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta} (a_t^{(i)} | s_t^{(i)}) \left(\sum_{t=0}^{T-1} r_t^{(i)} \right)$$

- ✓ $\tau^{(i)} = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, s_1^{(i)}, a_1^{(i)}, r_1^{(i)}, \dots, s_{T-1}^{(i)}, a_{T-1}^{(i)}, r_{T-1}^{(i)}, s_T^{(i)})$ is the i -th trajectory.
- ✓ This approach will be too slow and unreliable due to high variance on the gradient estimate
- ✓ We need various **variance reduction** techniques

Interpreting Policy Gradient

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\underbrace{\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right)}_{\text{probability of trajectory}} \underbrace{\left(\sum_{t=0}^{T-1} r_t \right)}_{\text{Reward of trajectory}} \right] \approx \underbrace{\frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \right) \left(\sum_{t=0}^{T-1} r_t^{(i)} \right)}_{\nabla_{\theta} J_{ML}(\theta)}$$



- Increase the likelihood for the trajectory with large accumulated reward
- Decrease the likelihood for the trajectories with small accumulated reward

Variance Reduction (Imposing Causality)

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right) \left(\sum_{t=0}^{T-1} r_t \right) \right] \approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \right) \left(\sum_{t=0}^{T-1} r_t^{(i)} \right)$$

- Apply causality: policy at time t' cannot affect reward at time t when $t < t'$

$$\begin{aligned} \nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] &\approx \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \right) \left(\sum_{t=0}^{T-1} r_t^{(i)} \right) \\ &= \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \sum_{t'=t}^{T-1} r_{t'}^{(i)} \\ &= \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t^{(i)} | s_t^{(i)}) \underbrace{\hat{Q}_t^{(i)}}_{\text{"reward to go"}} \end{aligned}$$

$$\begin{aligned} &(f_0 + f_1 + \dots + f_t + \dots + f_T)(f_0 + f_1 + \dots + f_t + \dots + f_T) \\ &= f_0(f_0 + f_1 + \dots + f_t + \dots + f_T) + \\ &\quad f_1(\underbrace{f_0}_{\text{grey}} + f_1 + \dots + f_t + \dots + f_T) + \\ &\quad \vdots \\ &\quad f_t(\underbrace{f_0 + f_1 + \dots}_{\text{grey}} + f_t + \dots + f_T) + \\ &\quad \vdots \\ &\quad f_T(\underbrace{f_0 + f_1 + \dots + f_t + \dots}_{\text{grey}} + f_T) + \end{aligned}$$

Variance Reduction (Subtracting Baseline)

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta} \left(a_t^{(i)} | s_t^{(i)} \right) \sum_{t'=t}^{T-1} r_{t'}^{(i)} \right] \approx \frac{1}{N} \sum_{i=1}^N \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta} \left(a_t^{(i)} | s_t^{(i)} \right) \sum_{t'=t}^{T-1} r_{t'}^{(i)}$$

- Subtract baseline term to reduce variance

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right]$$

- We need to verify the things:
 - 1) Inserting baseline does not make the gradient estimate biased
 - 2) Baseline reduce actually variance

➤ Staying unbiased and reducing variance is always good!

Variance Reduction (Subtracting Baseline)

(1) Inserting baseline does not make the gradient estimate biased

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \right) - \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) b(s_t) \right]\end{aligned}$$

- All we need to show is that for any single time t , the gradient of $\log \pi_{\theta}(a_t | s_t)$ multiplied with $b(s_t)$ is zero:

$$\begin{aligned}\mathbb{E}_{\tau \sim \pi_{\theta}}[\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) b(s_t)] &= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[\mathbb{E}_{s_{t+1:T}, a_{t:T-1}} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) b(s_t)] \right] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[b(s_t) \cdot \mathbb{E}_{s_{t+1:T}, a_{t:T-1}} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] \right] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t-1}} \left[b(s_t) \cdot \mathbb{E}_{a_t} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] \right] \\ &= \mathbb{E}_{s_{0:t}, a_{0:t-1}} [b(s_t) \cdot 0] = 0\end{aligned}$$

$$\because \mathbb{E}_{a_t} [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t)] = \int \frac{\nabla_{\theta} \pi_{\theta}(a_t | s_t)}{\pi_{\theta}(a_t | s_t)} \pi_{\theta}(a_t | s_t) da_t = \nabla_{\theta} \int \pi_{\theta}(a_t | s_t) da_t = \nabla_{\theta} \cdot 1 = 0$$

Variance Reduction (Subtracting Baseline)

(1) Inserting Baseline reduce actually variance

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \right) - \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) b(s_t) \right]\end{aligned}$$

- Why subtracting baseline reduces variance?

$$\begin{aligned}\text{Var} \left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t(\tau) - b(s_t)) \right) &\approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (R_t(\tau) - b(s_t)) \right)^2 \right] & \text{Var} \left(\sum_{i=1}^n X_i \right) \approx \sum_{i=1}^n \text{Var}(X_i) \\ &\approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[(\nabla_{\theta} \log \pi_{\theta}(a_t | s_t))^2 \right] \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left((R_t(\tau) - b(s_t))^2 \right) \right] & E[AB] = E[A]E[B]\end{aligned}$$

$\left(R_t(\tau) = \sum_{t'=t}^{T-1} r_{t'} \right)$

- We can optimally choose $b(s_t)$ to minimize variance as

$$\min_{b(s_t)} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left((R_t(\tau) - b(s_t))^2 \right) \right]$$

- ✓ Which gives the least square solution $b(s_t) = \mathbb{E}_{\tau \sim \pi_{\theta}}[R_t(\tau)]$

➤ have to re-fit the baseline estimate each time to make it as close to the expected return

Variance Reduction (Advantage Function)

$$\begin{aligned}\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \left(\sum_{t'=t}^{T-1} r_{t'} - b(s_t) \right) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) (Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) A^{\pi}(s_t, a_t) \right] \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \Psi_t \right]\end{aligned}$$

- The problem of policy gradient is reduced to finding good estimates Ψ_t , a topic of **Generalized Advantage Estimation** (Schulman et al., 2016)

where Ψ_t may be one of the following:

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

Variance Reduction (Off Policy learning)

$$\theta^* = \operatorname{argmax}_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)]$$

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log \pi_{\theta}(\tau) R(\tau)]$$

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

$$\mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \nabla_{\theta} \mathbb{E}_{\tau \sim \bar{\pi}} \left[\frac{\pi_{\theta}(\tau)}{\bar{\pi}(\tau)} R(\tau) \right]$$

$$\frac{\pi_{\theta}(\tau)}{\bar{\pi}_{\theta}(\tau)} = \frac{\cancel{p(s_0)} \prod_{t=0}^{T-1} \pi_{\theta}(a_t | s_t) \cancel{P(s_{t+1} | s_t, a_t)}}{\cancel{p(s_0)} \prod_{t=0}^{T-1} \bar{\pi}(a_t | s_t) \cancel{P(s_{t+1} | s_t, a_t)}} = \frac{\prod_{t=0}^{T-1} \pi_{\theta}(a_t | s_t)}{\prod_{t=0}^{T-1} \bar{\pi}(a_t | s_t)}$$

importance sampling

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

Variance Reduction (Off Policy learning)

$$J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] = \nabla_\theta \mathbb{E}_{\tau \sim \bar{\pi}} \left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} R(\tau) \right]$$

$$\nabla_{\theta'} J(\theta') = \mathbb{E}_{\tau \sim \pi_\theta} \left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)} \nabla_{\theta'} \log \pi_{\theta'}(\tau) R(\tau) \right]$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} \left[\left(\prod_{t=1}^T \frac{\pi_{\theta'}(a_t | s_t)}{\pi_\theta(a_t | s_t)} \right) \left(\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \right) \left(\sum_{t=1}^T r_t \right) \right]$$

$$= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_\theta(a_{t'} | s_{t'})} \right) \left(\sum_{t'=t}^T r_{t'} \left(\prod_{t''=1}^{t'} \frac{\pi_{\theta'}(a_{t''} | s_{t''})}{\pi_\theta(a_{t''} | s_{t''})} \right) \right) \right]$$

Ignore

Causality

$$\approx \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^T \nabla_{\theta'} \log \pi_{\theta'}(a_t | s_t) \left(\prod_{t'=1}^t \frac{\pi_{\theta'}(a_{t'} | s_{t'})}{\pi_\theta(a_{t'} | s_{t'})} \right) \left(\sum_{t'=t}^T r_{t'} \right) \right]$$