Policy Gradient and Hierarchical Reinforcement Learning

Reinforcement Learning Concepts

October 2025

1 Policy Gradient Methods

Policy Gradient (PG) methods directly parametrize the policy $\pi_\theta(a|s)$ and optimize its parameters θ by maximizing a performance objective (e.g., the expected total return $J(\theta)$).

1.1 Non-Associative Learning and REINFORCE

Non-Associative Learning is the simplest form of policy learning, often referring to the **Monte Carlo Policy Gradient** which learns a state-dependent policy but updates the policy based on the full episode's return.

- **REINFORCE (Monte Carlo Policy Gradient):** The foundational PG algorithm.
- **Objective:** Maximize the expected return $J(\theta) = \mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} R_t]$.
- **Gradient Update:** The update is proportional to the return multiplied by the score function:
$$\Delta\theta \propto \sum_{t=0}^{T} G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$$
- **Properties:** **Unbiased** estimate of the true policy gradient, but suffers from **high variance**.

1.2 Exact Gradient Methods and Gradient Estimation

The **exact gradient** $\nabla J(\theta)$ is impossible to compute without the environment model. PG methods use **Gradient Estimation** by replacing the theoretical expectation with a sample average (the Policy Gradient Theorem):

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(A_t|S_t) Q^\pi(S_t, A_t)\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T} \nabla_\theta \ln \pi_\theta(A_t|S_t) G_t$$

1.3 Approximate Policy Gradient Algorithms

These methods address the high variance of REINFORCE:

- **Baseline Subtraction:** Subtracting a **baseline** $b(S_t)$ from G_t reduces variance without changing the expected value of the gradient. The best baseline is the state value function $V^\pi(S_t)$.

- **Advantage Function:** Using $V^{\pi}(S_t)$ as the baseline yields the **Advantage $A_t = G_t - V^{\pi}(S_t)$**. The gradient is then proportional to $A_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$.
- **Advanced Methods (TRPO, PPO):** Algorithms like Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO) constrain the size of the policy update step to ensure stability and reliable improvement.

2 Actor-Critic Methods

In a simple term, Actor-Critic is a Temporal Difference(TD) version of Policy gradient. It has two networks: Actor and Critic. The actor decided which action should be taken and critic inform the actor how good was the action and how it should adjust. The learning of the actor is based on policy gradient approach. In comparison, critics evaluate the action produced by the actor by computing the value function.

This type of architecture is in Generative Adversarial Network(GAN) where both discriminator and generator participate in a game. The generator generates the fake images and discriminator evaluate how good is the fake image generated with its representation of the real image. Over time Generator can create fake images which cannot be distinguishable for the discriminator. Similarly, Actor and Critic are participating in the game, but both of them are improving over time, unlike GAN.

Actor-critic is similar to a policy gradient algorithm called REINFORCE with baseline. Reinforce is the MONTE-CARLO learning that indicates that total return is sampled from the full trajectory. But in actor-critic, we use bootstrap. Actor-Critic (AC) methods are

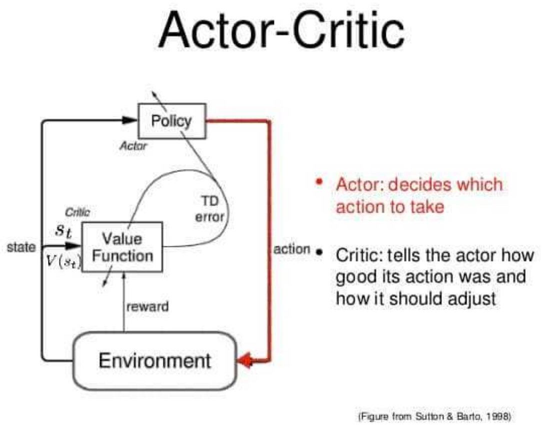
Actor-Critic
Policy
Actor
Critic
TD error
state
s_t
$V(s_t)$
Value Function
reward
action
Environment
Actor: decides which action to take
Critic: tells the actor how good its action was and how it should adjust
(Figure from Sutton & Barto, 1998)

Figure 1: Actor-Critic Architecture

associative in that they combine two interacting components for optimization:

1. **Actor (π_θ):** The policy that selects actions. Updates use the Critic's estimate.
2. **Critic ($\hat{v}_w$ or $\hat{q}_w$):** The value function estimator. Updates are based on TD errors.

Tradeoff: AC methods offer **lower variance** (due to bootstrapping) but introduce **bias** (as they rely on a learned, approximate value function).

3 Hierarchical Reinforcement Learning (HRL)

HRL addresses complex, long-horizon tasks by introducing **temporal abstraction**, allowing the agent to utilize multi-step, high-level actions.

Component	Role	Trainging Signal
Critic	Estimates $V(s)$ or $Q(s, a)$, often using TD Error^2.	Updates its parameters w to minimize TD Error
Actor	Updates the policy π_θ, using the Critic's output to estimate the action advantage.	Updates θ in the direction of the estimated advantage.

3.1 Options Framework

The formalization of temporal abstraction defines an **Option** as a generalization of a primitive action: $(\mathcal{I}, \pi, \beta)$:

- **$\mathcal{I}$ (Initiation Set):** The set of states where the option can be started.
- **π (Intra-Option Policy):** The policy followed (primitive actions) while the option is executing.
- **β (Termination Condition):** The probability of the option terminating in a given state $\beta(s) \in [0, 1]$.

3.2 MAXQ Framework

Developed by Dietterich, MAXQ formalizes HRL through **task decomposition**:

- **Task Graph:** The problem is structured as a hierarchy of composite and primitive tasks (actions).
- **MAXQ Value Function:** Decomposes the value of a composite task M recursively into the value of the subtask $V(j, s)$ and the **Completion Function** $C(j, s, a)$:

$$Q(j, s) = V(j, s) + C(j, s, a)$$

- **Property:** Guarantees convergence to a **recursive optimal policy**, an approximation of the globally optimal policy.

3.3 HAM Framework (Hierarchical Abstract Machine)

The HAM framework is a **formal language** used to specify the hierarchical policy itself.

- **Policy as a Program:** A policy is defined as a program that can include primitive actions, calls to subroutines (other HAMs/subtasks), and control flow constructs (`WHILE`, `IF`).
- **Advantage:** Provides a clear structure and guarantees convergence to an optimal policy *within the constraints of the defined HAM hierarchy.*

3.4 Option Discovery Algorithms

These algorithms automate the identification of useful, reusable skills (Options), avoiding manual definition:

- **State Abstraction / Bottleneck States:** Identifying states that are frequently traversed to reach various goals. These states serve as ideal initiation ($\mathcal{I}$) or termination (β) points for options.

- **Feudal RL:** A structural approach where a high-level Manager sets goals and a low-level Worker executes primitive actions to achieve them.
- **Spectral Clustering/Information Theory:** Using graph analysis or maximizing information transfer to segment the state space and derive options from the resulting regions.