

Deep Reinforcement Learning

Alireza Azimi

October 14, 2026

So what's so deep about deep reinforcement learning?

Reinforcement learning, in plain terms, is about learning how to act through experience. An **agent** takes actions in an **environment**, receives rewards, and learns to make decisions that maximize its cumulative reward.

The agent's rule for choosing actions is called a **policy**. A deterministic policy maps a state to an action:

$$\pi : \mathcal{S} \rightarrow \mathcal{A},$$

where $\mathcal{S}$ is the set of possible states and $\mathcal{A}$ is the set of possible actions.

A policy is just a mathematical function. It can be very simple—even a little “zombie-like.” For example:

$$\pi(s) = \text{move left.}$$

This policy always picks the same action, regardless of the situation. A more useful policy might consider its surroundings:

$$\pi(s) = \begin{cases} \text{stop moving,} & \text{if near a cliff,} \\ \text{move forward,} & \text{otherwise.} \end{cases}$$

Now we're getting somewhere. This resembles a program that makes decisions using explicit conditions.

We can get very far with this approach, but writing every rule by hand becomes difficult as the task grows. How do we recognize a cliff? Where should we go when we're safe? What if we encounter lava instead? Every new situation may require another rule.

The policy can only respond using the logic we supplied. How could it improve its decisions through experience instead? That question gets at the heart of reinforcement learning.

What is the agent trying to maximize?

For simplicity, consider a task that ends: reaching a destination, finishing a game, or completing a trial. Each run is called an **episode**. An episode can end in success, failure, or when a time limit is reached.

At time t , the agent observes its environment and takes action A_t . The environment then produces a reward R_{t+1} and a new observation.

If the episode ends at time T , the **return** from time t is:

$$G_t = \sum_{k=t}^{T-1} R_{k+1}.$$

Return is the sum of rewards along one particular episode. Since the environment—or the policy itself—may involve randomness, the same policy can produce different returns on different runs.

Our objective is therefore to find a policy that maximizes **expected return**:

$$\pi^* \in \arg \max_{\pi} \mathbb{E}_{\pi}[G_0].$$

The expectation averages over the possible episodes produced by following π , including randomness in the starting state, actions, and environment.

Often, we also use a **discount factor** γ :

$$G_t = \sum_{k=t}^{T-1} \gamma^{k-t} R_{k+1}, \quad 0 \leq \gamma \leq 1.$$

Setting $\gamma = 1$ gives the undiscounted episodic return above. Smaller values give less weight to rewards farther in the future.

Notice that the objective concerns cumulative reward, not just the next reward. A useful action might have an immediate cost but lead to a much better outcome later.

From policies to parameters

Our policy needs information on which to base its decisions. Let's call the agent's observation at time t O_t .

A **state** describes the environment's current situation; an **observation** is the information available to the agent. These can differ. A camera image, for example, may not reveal everything the agent needs to know. In that case, a policy may also need memory of previous observations.

For now, suppose the current observation is sufficient. We can write a parameterized deterministic policy as:

$$A_t = \pi_\theta(O_t),$$

where θ represents its adjustable parameters.

Policies can also be **stochastic**: instead of choosing one action directly, they assign probabilities to actions:

$$A_t \sim \pi_\theta(\cdot \mid O_t).$$

Remember $y = mx + b$ from school? Here, x is the input, while both m and b are parameters. Changing either changes how the function maps inputs to outputs.

The same idea applies to a parameterized policy. We adjust θ to change the agent's behavior. Within our chosen policy family, the objective becomes:

$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_0], \quad \theta^* \in \arg \max_{\theta} J(\theta).$$

This does not guarantee that our policy family contains the best possible behavior—or that our algorithm will find it. It gives us a concrete objective to work toward.

How does learning happen?

Writing down the objective does not tell us how to solve it.

An RL algorithm uses experience—observations, actions, and rewards—to improve behavior. During online training, the agent gathers this experience by interacting with the environment. Offline RL instead learns from previously collected data.

Two difficulties make this interesting. The agent must discover useful actions through **exploration**, and it must work out which earlier decisions contributed to later rewards. The latter is called **credit assignment**.

Humans still design the task, reward signal, and learning setup. What changes is that we no longer need to specify the correct action for every situation.

Where does the “deep” come in?

A linear policy might take the form:

$$\pi_{\theta}(o) = Wo + b, \quad \theta = (W, b).$$

That can be useful, but it has limited expressive power. For more complicated relationships between observations and actions, we can use a neural network.

A neural network combines layers of transformations, typically with nonlinear activation functions between them. A deep network has multiple layers that learn intermediate representations of the input. For example, a network processing images might learn visual features that help the agent choose actions.

Simply making a function complicated or differentiable does not make it a neural network. The layered structure is what matters.

Deep reinforcement learning uses deep neural networks as part of an RL system. The network may represent the policy directly, estimate how much future reward an action could lead to, or learn a model of the environment.

For example, a deep Q-network estimates action values and uses those estimates to choose actions. It does not need a separate neural network that directly outputs a policy. [See Spinning Up’s overview of RL algorithms.](#)

So the search for better behavior is the reinforcement learning part. Representing the functions involved with deep neural networks is the deep learning part.

That is what makes deep RL “deep.”