UNIVERSITÀ DEGLI STUDI FIRENZE | DINFO DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE | DISIT DISTRIBUTED SYSTEMS AND INTERNET TECHNOLOGIES LAB

https://www.snap4City.org SNAP4CITY KM4CITY

NeuroSymbolic Artificial Intelligence at Scale

Paolo Nesi, paolo.nesi@unifi.it

Marco Fanfani, marco.fanfani@unifi.it

https://www.disit.org/

Parte: 4 (2025-26)

Neuro-Symbolic Artificial Intelligence

16/06/2026

P4: Deep Reinforced Learning and Symbolic at Scale

- multi agent Deep reinforced learning
- RL and simulation

Corso: **Neuro-Symbolic Artificial Intelligence at Scale**
P4: Deep Reinforced Learning and Symbolic at Scale
multi agent Deep reinforced learning
RL and simulation

[Link: Neuro-Symbolic Artificial Intelligence at Scale | Disit]

TESTO CONSIGLIATO

Reinforcement Learning: An Introduction di Sutton e Barto (https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf)

Luciano Alessandro Ipsaro Palesi
lucianoalessandro.ipsaropalesi@unifi.it
Room 231 – ex 465
https://www.disit.org
https://www.Snap4City.org

Office hours
Monday 10:30 - 13:00,
Via Santa Marta 3 - 50139 Firenze
Room **231**

- **Deep Learning**
- **Explainable Artificial Intelligence (XAI)**
- **Optimization and Simulation of Complex Systems**
- **Smart City, IoT/WoT e Digital Twin**

Google scholar:
https://scholar.google.com/citations?user=ZVSLAdgAAAAJ
Scopus:
https://www.scopus.com/authid/detail.uri?authorId=57226812823
ORCID:
https://orcid.org/0000-0001-8992-2084

Lecture Objectives

By the end of this course, you will be equipped to design, analyze, and implement reinforcement learning systems.

1

The RL Framework

Understand the agent–environment interaction loop and how reward signals drive learning

2

Value Functions

Study Bellman equations and how they underpin every major RL algorithm

3

Fundamental Algorithms

Analyze Dynamic Programming, Monte Carlo Methods, and Temporal Difference Learning

4

Deep RL

Combine deep neural networks with RL to tackle large-scale, real-world problems

Lecture Structure

PART I

Theoretical Foundations

Building rigorous mathematical intuition from the ground up.

- Markov Decision Processes
- Value functions & Bellman equations
- Dynamic Programming
- Monte Carlo Methods

PART II

Deep Reinforcement Learning

Scaling RL to complex, high-dimensional environments.

- Neural value function approximation
- Temporal Difference Learning
- Deep Q-Networks (DQN)
- Modern applications & case studies

Why RL is Neuro-Symbolic

This course sits at the intersection of two historically distinct AI traditions — and RL is one of the clearest examples of their integration.

Symbolic AI

Classical RL rests on explicit mathematical structures: MDPs, Bellman equations, probabilistic models — all interpretable and formally grounded.

Neural AI

Deep neural networks learn rich representations from raw data — enabling RL to operate in image spaces, robotic sensors, and complex environments.

The Synthesis

Deep RL fuses symbolic structure with neural representational power — producing interpretable, scalable, and high-performing decision-making systems.

From Classical RL to Deep RL

The Scalability Problem

Classical tabular methods represent value functions as lookup tables — perfectly precise, but only feasible for small, discrete state spaces.

Real-world tasks — images, robotic sensors, continuous control — have state spaces far too large for tables.

Deep RL Solution: Replace tables with deep neural networks that *approximate* value functions and policies, scaling to virtually any environment.

Reinforcement Learning (RL)

RL is a type of machine learning.

It trains an agent to make decisions by interacting with an environment

- Learning through interaction
- An agent that maximizes reward
- Agent–environment framework
- **Sequential decision-making problems**

The Core Idea of Reinforcement Learning

RL is built on a simple but powerful loop: the agent observes, acts, and learns from the consequences.

Observe State

Choose Action

$$S_t \rightarrow A_t \rightarrow R_{t+1}, S_{t+1}$$

Receive Reward

Environment Transitions

The agent's goal is never to maximize the *immediate* reward — it must maximize the **sum of future rewards**. A chess move may seem weak short-term yet lead to a win many moves later. This temporal dependency is what makes RL uniquely challenging.

RL as Problem, Method, and Research Field

The term **Reinforcement Learning** carries three distinct meanings simultaneously.

policy

$$\pi(s) = a$$

A Problem

Learn a **state → action mapping** (policy) that maximizes cumulative reward.

A Class of Algorithms

Q-learning, SARSA, Monte Carlo Methods, Temporal Difference Learning.

A Research Field

Bridges ML, optimal control, neuroscience, psychology, and decision theory.

Three Fundamental Characteristics of RL

Closed-Loop Problem

The agent's actions directly influence future observations. The system is fully interactive — behavior shapes the data stream.

No Explicit Supervision

The agent never receives "action X is correct." It receives only a **scalar reward signal** reflecting overall performance.

Delayed Consequences

Actions affect not just immediate reward but all future rewards. Identifying which past action caused a future outcome is the **credit assignment problem**.

Which of my actions actually contributed to the final result?

RL vs. Supervised Learning

Supervised Learning

Learns from a labeled dataset **{(x, y)}**. For every input, the correct output is explicitly provided. The goal: learn a mapping **f(x) → y**.

- Fixed, labeled dataset
- Independent data samples
- Clear ground truth signal

Reinforcement Learning

No labeled dataset exists. The agent only receives a reward *after* acting — never the "correct" action. Data is **sequential and interdependent**.

- Must explore to collect experience
- Actions change future data distribution
- Closer to control theory than classic ML

RL vs. Unsupervised Learning

While unsupervised learning discovers hidden structure in data, RL optimizes an **explicit behavioral objective**.

1	2	3
Supervised Learning	**Unsupervised Learning**	**Reinforcement Learning**
Learn input → output mapping from labeled examples	Find hidden patterns — clustering, reduction, generation	Learn decision strategies to maximize cumulative reward

RL may internally leverage supervised or unsupervised techniques, but its core purpose remains **reward optimization over time** — not description.

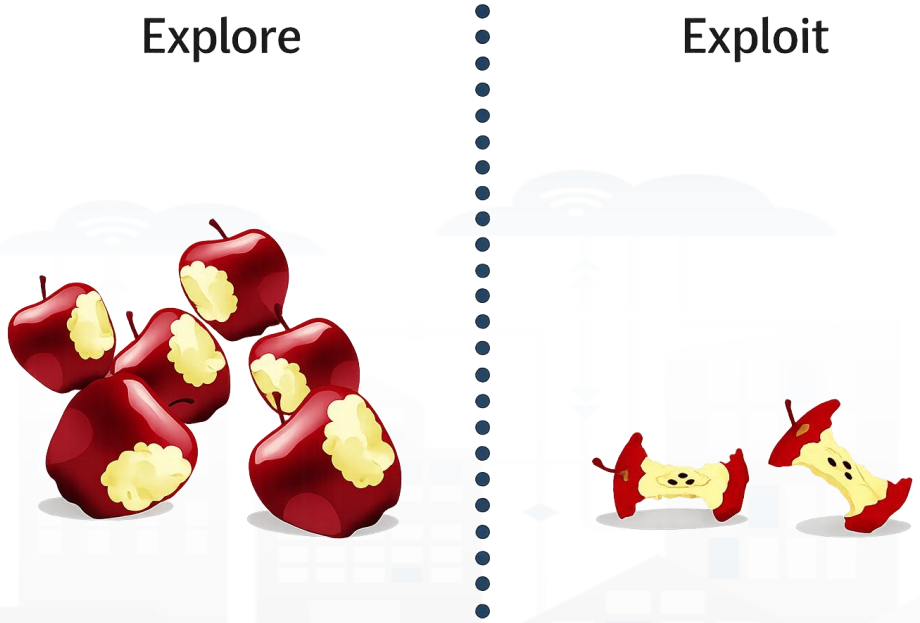
The Exploration vs. Exploitation Dilemma

Exploitation

Use what we already know. If an action yielded high reward before, repeat it — but risk missing something better.

Exploration

Try new actions to gather information — but accept lower short-term reward. Too much exploration wastes opportunities.



The **multi-armed bandit** problem is the classic formalization: given N slot machines with unknown payoffs, how do you allocate plays to maximize total winnings? Balancing this trade-off is one of the central mathematical challenges in RL.

RL as a Complete Agent

Unlike other AI subfields that tackle isolated problems, RL models the **entire decision-making system**.

Perception

Observes the environment state at each time step

Decision

Selects actions under uncertainty and partial information

Reward

Receives feedback and updates its policy accordingly

Adaptation

Continuously improves behavior in stochastic environments

This makes RL a **general framework for sequential decision-making** — directly applicable to robotics, autonomous systems, and resource management.

Interdisciplinary Connections

1

Optimal Control & Dynamic Programming

Bellman's equations from the 1950s form the mathematical backbone of modern RL algorithms.

2

Psychology of Learning

Trial-and-error learning from animal behavior studies directly inspired the RL framework.

3

Neuroscience

Temporal Difference Learning models dopaminergic neuron activity — suggesting the brain implements RL-like mechanisms.

4

Modern AI

AlphaGo, AlphaZero, and Atari agents represent RL's most spectacular real-world achievements.

Examples of RL Problems

All share the same three traits: **environment interaction**, **explicit goals**, and **sequential decisions under uncertainty**.

Chess Player

Plans moves combining long-term strategy with experience-driven intuition.

Industry / SmartCity

Continuously tunes parameters to optimize cost, quality, and production or infrastructure optimization.

Autonomous Robot

Decides whether to keep exploring or return to recharge — balancing goals. BOSTON DYNAMICS

Traffic Light Management

Continuously adjusts traffic signals to minimize congestion, waiting times, and travel delays.

Game Playing

REINFORCEMENT LEARNING IN ACTION

Example: Chess, Go, and Atari Games — where RL agents learn to compete at superhuman levels.

State

Board configuration or game screen

Action

Move selection or game control input

Reward

Win condition, score increase, or game progress

Learning

Discover strategies that maximize long-term success

Used by: DeepMind (AlphaGo, AlphaZero) and game AI research labs pushing the boundaries of machine intelligence.

REINFORCEMENT LEARNING IN ACTION

Robotics

Example: A robot learning to walk or grasp objects through trial-and-error interaction with its environment.

Used by: Boston Dynamics and OpenAI robotics research.
BOSTON DYNAMICS

State
Joint positions and sensor readings

Action
Motor commands to limbs or grippers

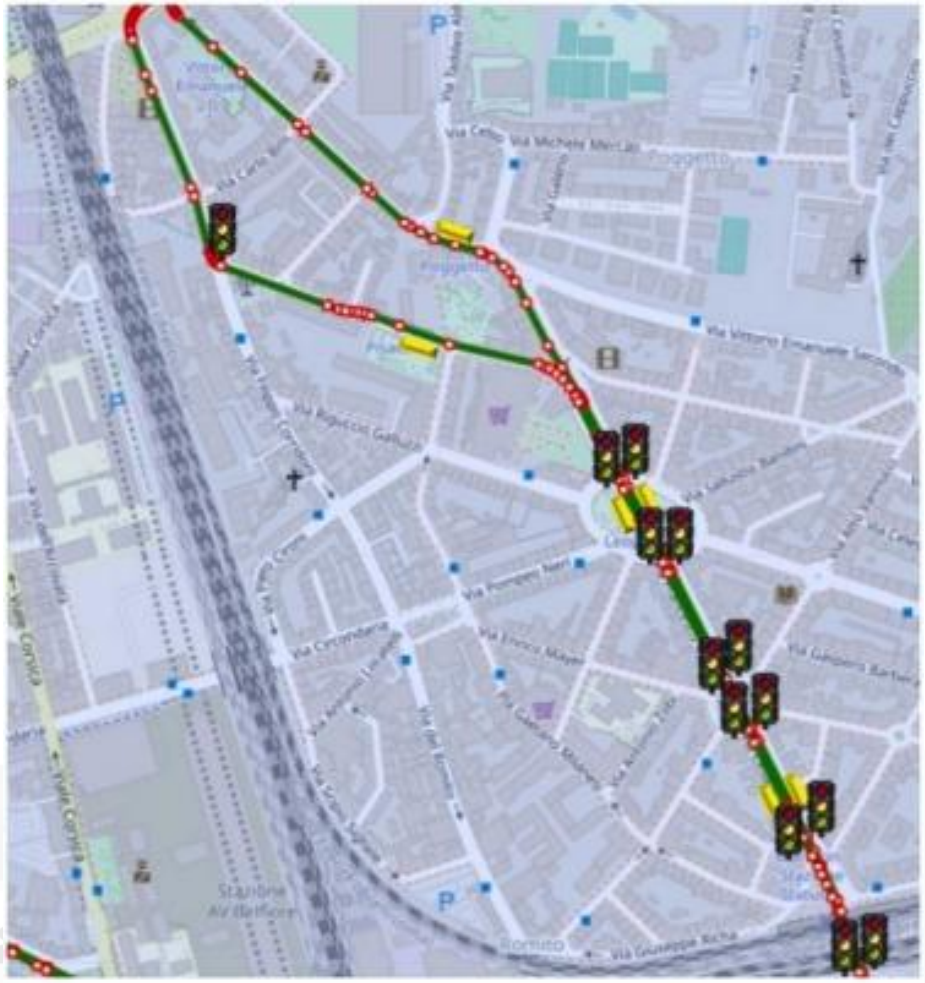
Reward
Stable movement or successful grasp

Learning
Improve motor control through repeated interaction

Traffic Light Control

Example: Smart traffic signal management systems that adapt in real time to reduce congestion across urban road networks.

1. **State**
   Cars waiting at each intersection
2. **Action**
   Change traffic light phase or timing
3. **Reward**
   Reduced wait time and congestion
4. **Learning**
   Optimize city-wide traffic flow over time



Used by: Smart city traffic systems and urban mobility research initiatives worldwide.

Recommendation Systems

Example: Video or product recommendations that learn from your behavior to surface increasingly relevant content.

Used by: YouTube, Netflix, and Spotify to keep billions of users engaged daily.

State

User preferences and interaction history

Action

Recommend a video, song, or product

Reward

Click-through, watch time, or engagement signal

Learning

Maximize long-term user engagement

REINFORCEMENT LEARNING IN ACTION

Training Language Models (RLHF)

Example: Reinforcement Learning from Human Feedback (RLHF) shapes how large language models respond — making them more helpful, accurate, and safe.

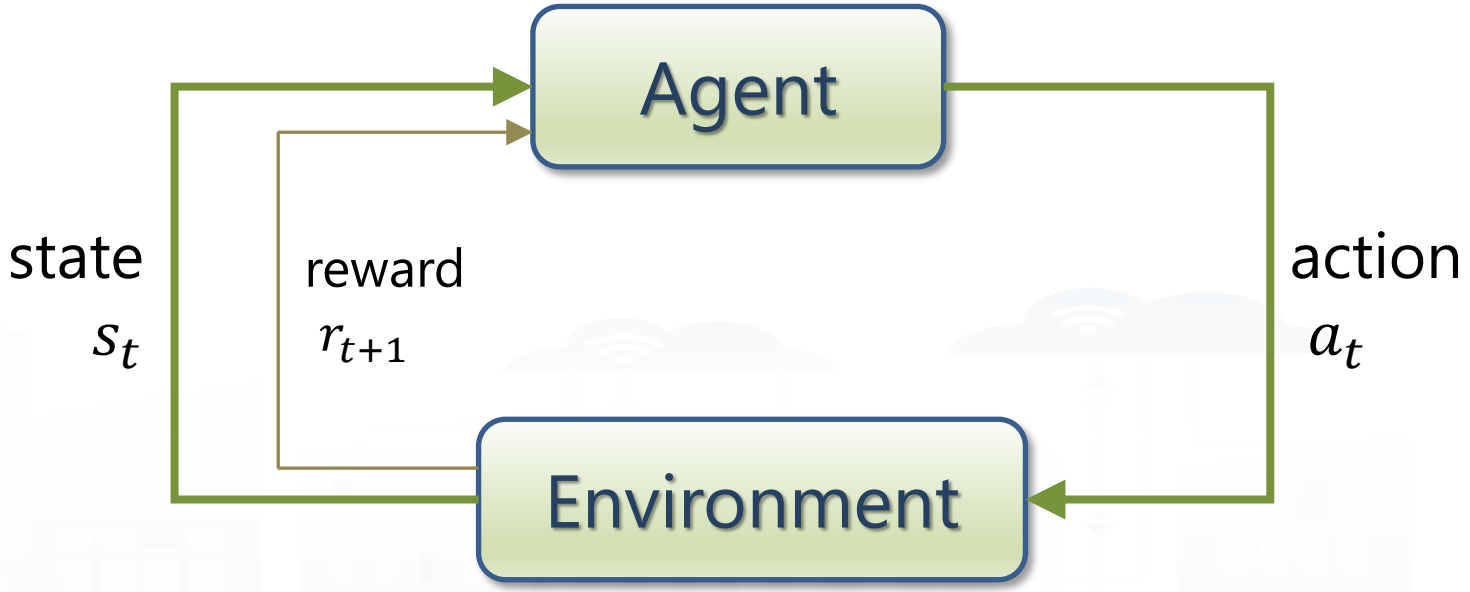
1. **State**
   Generated response or conversation context
2. **Action**
   Adjust model output or policy parameters
3. **Reward**
   Human feedback score rating the response quality
4. **Learning**
   Produce more helpful, accurate, and safe responses

Used by: ChatGPT and virtually all modern large language models as a core alignment technique.

The Agent–Environment Loop

At each timestep **t**, the interaction unfolds in four steps — repeated indefinitely over time.

1. **Observe State S_t**
   The agent perceives the current state of the environment.
2. **Select Action A_t**
   Based on its policy, the agent chooses an action.
3. **Environment Evolves**
   The environment transitions to a new state S_{t+1}
4. **Receive Reward R_{t+1}**
   A scalar signal indicates the quality of the action taken.



The environment can be **stochastic**: the same action in the same state may produce different outcomes at different times. This framework generalizes to robots, games, control systems, and trading algorithms.

$$S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, \dots$$

Elements of RL

State S_t

The current situation as perceived by the agent.

- Chessboard configuration
- Robot joint positions
- Industrial system status

Action A_t

The decision taken by the agent in the current state.

- Moving a chess piece
- Accelerating a vehicle
- Recommending a product

Reward R_{t+1}

A scalar number indicating how useful the action was toward the goal.

- Positive → desired behavior
- Negative → undesired behavior

Policy π

Defines the agent's behavior: which action to choose in each state.

Reward Signal

Defines the objective. The agent seeks to maximize cumulative reward.

Value Function

Estimates how "good" a state is in the long run — beyond immediate reward.

Environment Model

Predicts next state and reward. Used for planning. Optional — not all algorithms require it.

Policy

Policy π

The policy is the function that fully defines how the agent behaves. It maps every state to an action — or a probability distribution over actions.

Stochastic policies are often preferred — they support **exploration**, **robustness**, and **uncertainty handling**. The ultimate goal of RL is to find the optimal policy that maximizes expected cumulative reward.

Deterministic Policy

Each state maps to exactly one action: $\boldsymbol{a} = \boldsymbol{\pi}(\boldsymbol{s})$

Stochastic Policy

Actions are sampled from a distribution: $\boldsymbol{\pi}(\boldsymbol{a}|\boldsymbol{s}) = P(A = a \mid S = s)$

Reward

At each timestep, the environment returns a scalar number R_{t+1}— the reward. This is the **only** formal definition of the agent's objective.

Reward is Immediate

It signals the quality of the action just taken. The agent observes it directly — no estimation needed.

- Eating junk food → high immediate reward
- Long-term health → low value

Why Reward Alone Is Not Enough

A smart agent must look beyond immediate reward and learn to anticipate **future consequences**. Designing the right reward signal is one of the hardest challenges in applied RL.

Value Function

The value of a state is the total expected reward the agent can accumulate in the future, starting from that state and following a given policy.

Formally: $v_\pi(s)$

The value function under policy π estimates the expected return G_t — the sum of discounted future rewards from state s.

$$v_\pi(s) = \mathbb{E}[G_t \mid S_t = s]$$

Understanding this distinction is essential. Optimal decisions are **always** based on value — not on immediate reward.

Reward

Primary signal from the environment. Observed directly at each step. Requires no estimation.

Value

Derived quantity estimated by the agent. Represents expected cumulative reward. Depends on environment dynamics and current policy.

A state may have a **low immediate reward** but a **very high value** — because it leads to highly favorable future situations. This is why almost all RL algorithms include a mechanism to continuously estimate value functions.

Reward

What is good **right now**

Value

What is good **in the long run**

Model-Based vs Model-Free RL

The fourth optional component of RL systems is an **environment model** — a function that predicts the next state and reward given a current state and action.

🗺 Model-Based

The agent builds or uses a model to **simulate future scenarios** and plan ahead before acting.
Example: AlphaGo combines learning with model-based planning.

Model-Free

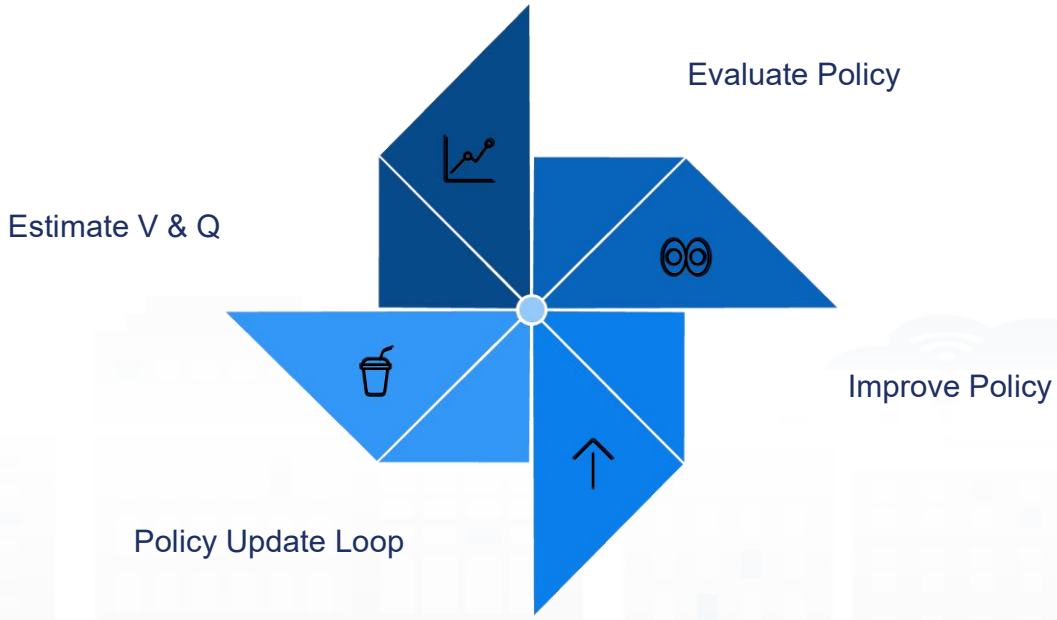
No model is used. The agent learns directly from **trial-and-error experience**. Simpler and widely used. Examples: Q-learning, SARSA, TD learning.

$$p(s', r \mid s, a)$$

Modern RL research increasingly explores **hybrid approaches** that combine the sample efficiency of model-based planning with the robustness of model-free learning.

Value Estimation: The Heart of RL

Reward alone is not enough. Intelligent decision-making requires estimating long-term consequences.



This cycle — **evaluate, then improve** — is called **policy improvement** and is one of the central concepts of the course. Next, we will formalize these ideas mathematically using **Markov Decision Processes (MDPs)**.

Objective: Maximize Cumulative Reward

An RL agent doesn't just chase immediate rewards — it optimizes for **cumulative reward over time**. Every decision shapes not only the next step, but the entire future trajectory.

The Reward Sequence

At each timestep, the agent receives a reward signal. The goal is not to maximize any single reward, but the **sum of all future rewards** — the return.

The return G_t represents the total discounted reward the agent can expect from time t onward.

Why Sequential Decisions Matter

Each action influences not only the immediate reward, but **all subsequent rewards**. This temporal dependency is what makes RL fundamentally different from static optimization.

- Actions have long-range consequences
- Short-term sacrifice may yield long-term gain
- The agent must plan, not just react

Return & Discounting

In infinite processes, summing all future rewards naively can diverge. The **discount factor γ ∈ [0, 1)** elegantly solves this — and encodes how much the agent values the future.

Discounted Return

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$$

γ close to 0

Myopic agent: prioritizes immediate rewards, largely ignores the future.

γ close to 1

Far-sighted agent: treats future rewards almost as seriously as immediate ones.

The discount factor has a dual role: it ensures **mathematical convergence** of the sum, and it models **behavioral preference** toward present vs. future outcomes.

Episodic/Continuing Tasks

Many RL problems are structured as **episodes** — finite sequences of interactions that end in a **terminal state**.

- A chess or Atari game match
- A robot navigating to a goal
- A single planning mission

The return is simply the sum of rewards until episode end: $G_t = R_{t+1} + \gamma R_{t+2} + \ldots + \gamma^{T-t-1} R_T$ up to T. The natural endpoint makes these problems easier to analyze — most introductory RL algorithms are developed in this setting.

Some problems have **no natural endpoint**. The agent interacts with the environment indefinitely.

- Industrial control systems
- Telecommunication network management
- Online recommendation engines

The return becomes an infinite discounted sum: $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots$. Here $\gamma < 1$ is essential — without it the sum diverges. Discounting also keeps the problem temporally local, giving more weight to near-term consequences.

The Markov Property

The Future Depends Only on the Present

A state satisfies the **Markov property** if it contains **all information necessary** to predict the future — no history needed.

$$P(S_{t+1} \mid S_t)$$

In chess, the board configuration is a fully sufficient state — past moves are irrelevant given the current position. This property is the **theoretical foundation of MDPs** and the entire RL framework.

Markov Decision Processes (MDPs)

An MDP is the **mathematical model** that formalizes every RL problem. It captures both the environment's dynamics and the uncertainty of outcomes.

State Space S

All possible situations the agent can be in

Action Space A

All possible decisions available to the agent

Transition Function p

Probability of reaching state s' and reward r given (s, a)

Reward Function R

Expected immediate reward for taking action a in state s

Most RL algorithms solve MDPs **without full knowledge** of the transition and reward Functions
they learn from interaction alone.

Optimal Policies

The ultimate goal of RL: find a **policy** $\pi*$ that maximizes the expected return from every state.

Optimal State Value — V*(s)

The maximum expected cumulative reward obtainable starting from state *s*, following the best possible policy

Optimal Action Value — Q*(s, a)

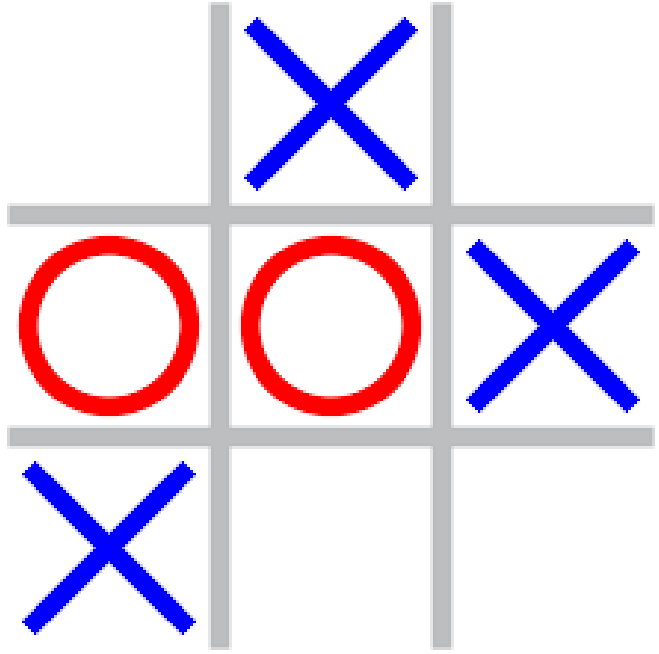
The expected return when taking action *a* in state *s* and thereafter following $\pi*$:

Deriving the Optimal Policy

Once Q* is known, the optimal policy is simply: $\pi^*(s) = \arg\max_a q^*(s,a)$

Many RL algorithms — Q-learning, DQN, SARSA — aim to estimate Q* directly.

The Tic-Tac-Toe Game



Why This Game?

Tic-Tac-Toe is a perfect RL testbed: two players compete on a 3×3 grid, placing X or O symbols. The winner is whoever aligns **three symbols in a row** — horizontally, vertically, or diagonally. If no one succeeds, the game ends in a draw.

Sequential

Decisions unfold over time

Discrete

Finite states & actions

Clear Goal

Win, lose, or draw

Formulating Tic-Tac-Toe as an RL Problem

1. **State**
   The current configuration of the board — every arrangement of X and O symbols defines a unique state.
2. **Action**
   Any empty cell the agent can place its symbol in. The action space shrinks as the game progresses.
3. **Reward**
   +1 for a win, **−1** for a loss, **0** for a draw — assigned at the end of the episode.
4. **Policy**
   Improves over repeated games as the agent learns which board configurations lead most often to victory.

The Value Function

The key idea: assign a **value** to every state — interpreted as the estimated probability of winning from that position.

Win State	Neutral State	Loss State
Value = **1.0**	Value ≈ **0.5** (initial)	Value = **0.0**

These values are not fixed — they are updated continuously through experience. When choosing a move, the agent evaluates all reachable states and selects the one with the **highest value**.

Greedy Action Selection

How the Agent Decides

The simplest strategy is the **greedy policy**: always pick the action that leads to the state with the maximum estimated value.

This fully exploits current knowledge — but it has a critical flaw: the agent may **never discover better moves** it hasn't tried yet.

This is why RL systems must also incorporate a mechanism for **exploration**.

$$a = \arg\max_a V(s')$$

Exploration vs. Exploitation

To avoid being trapped in a suboptimal strategy, the agent occasionally makes **non-greedy, exploratory moves**.

Exploitation

Most of the time, the agent chooses the move with the highest known value — leveraging its experience.

Exploration

Sometimes, the agent picks a random move to discover new states and gather fresh information.

The Balance

This trade-off is one of the **central challenges** in all of reinforcement learning.

Value Backup: Propagating Information

How Updates Work

When a game ends, the agent knows the outcome. This result carries information about **every state visited** during the episode.

The key insight: if a state is frequently followed by states that lead to victory, its own value should be **high**. Information flows backward through the game.

Updates are made **incrementally** — estimates improve gradually, enabling **online learning** during play.

The Update Rule: Temporal Difference

The agent updates state values using a **Temporal Difference (TD)** rule:

$$V(s) \leftarrow V(s) + \alpha[V(s') - V(s)]$$

$V(S_t)$

The current estimate of the value of state S_t.

$V(S_{t+1}) - V(S_t)$

The **prediction error** — how much the next state's value differs from the current estimate.

α (step-size)

Controls the **learning rate**: how strongly each new observation updates the existing estimate.

If the next state turns out better than expected, the current state's value is increased and vice versa.

Convergence to Optimal Play

With enough games, the estimated values **converge to the true probabilities of winning** from each state.

Accurate estimates emerge

The greedy policy over these values becomes the **optimal policy** against that opponent.

No prior knowledge needed

The agent doesn't know the opponent's strategy or all possible move sequences — it learns purely from experience.

Power of RL

Effective strategies emerge in complex, uncertain environments through repeated interaction alone.

Why This Example Matters

Tic-Tac-Toe is a **microcosm of modern RL**. The same ideas that power this simple game underlie the most advanced algorithms today.

Value Functions

Mapping states to expected outcomes — the foundation of RL decision-making.

TD Learning

Online, incremental updates that power Q-Learning, DQN, and Actor-Critic methods.

Exploration

Balancing known rewards with the search for better strategies — central to all RL systems.

Policy Improvement

Using accumulated experience to progressively refine behavior toward optimality.

Limits of Reinforcement Learning

Despite its power, RL faces several critical practical challenges that limit real-world deployment.

🐢 Slow Learning

Many RL algorithms require **millions of interactions** before converging to an effective policy.

💸 Costly Exploration

Trying unknown actions can be **dangerous or expensive** — robots may fall, machines may break, financial strategies may fail.

🌐 Complex Environments

Real-world state spaces are often **enormous or continuous**, making complete exploration infeasible.

🔍 Partial Observability

Agents often cannot observe the **full state** of the environment, introducing uncertainty.

🎯 Reward Design

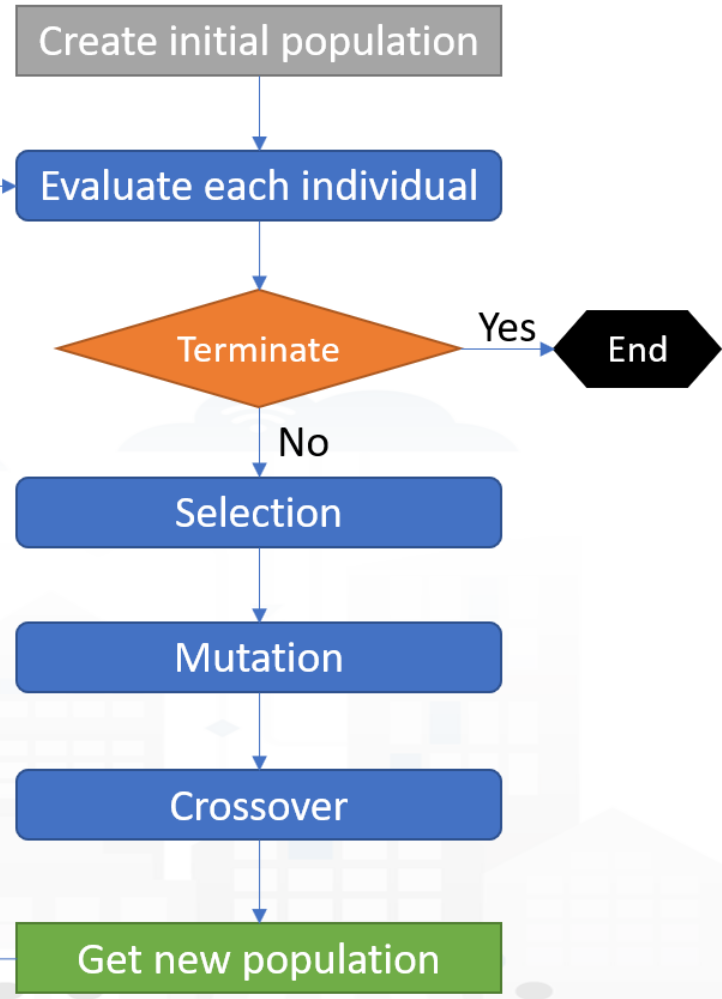
A poorly designed reward function can lead to unexpected or undesirable learned behaviors.

Evolutionary Methods

An alternative to value-function-based RL: search **directly in the space of policies**, inspired by natural selection.

How It Works

1. Generate a **population** of candidate policies
2. **Evaluate** each policy by simulating the agent in the environment
3. **Select** the best-performing policies
4. **Combine and mutate** them to create the next generation



Key Examples

- Genetic Algorithms
- Genetic Programming
- Evolutionary Strategies

Best suited for environments where state perception is imperfect or hard to model.

RL vs. Evolutionary Methods

The core difference lies in **how information is used** during learning.

Evolutionary Methods

Evaluate only the **final outcome** of an entire policy. In Tic-Tac-Toe: count wins over many games — all intermediate states are ignored.

Reinforcement Learning

Uses **local, step-by-step information**. Every visited state contributes to learning, making RL generally more data-efficient.

Online vs. Offline

RL learns **online** — while the agent interacts with the environment. Evolutionary methods typically evaluate complete policies offline.

Both approaches are valid, and in some advanced systems they can be **combined** for greater performance.

RL and Optimization

RL seeks to **maximize reward** — but this does not guarantee finding the global optimum.

Insufficient Exploration

The agent may never visit all relevant states.

Stochastic Environments

Randomness makes consistent optimal behavior harder to achieve.

Approximate Models

Function approximators introduce bias and generalization error.

Computational Limits

Real-time constraints prevent exhaustive search.

Unlike classical optimization algorithms, RL is an **iterative and adaptive process** — the goal is often a *sufficiently good* policy, not a mathematically perfect one.

Connection with Dynamic Programming

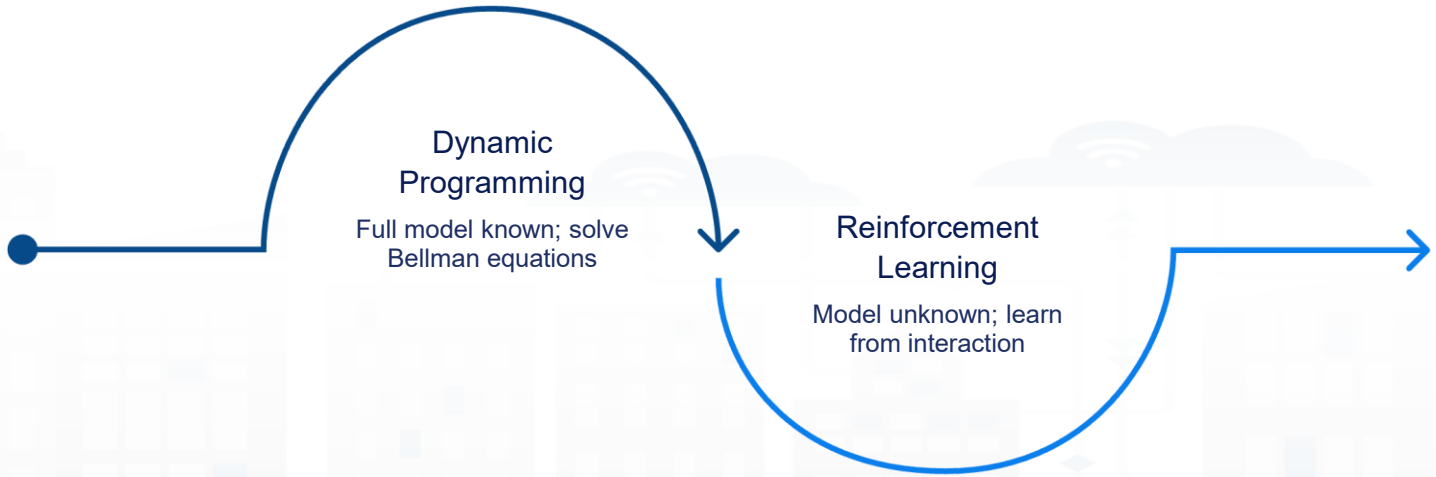
The Bellman Equation

Developed by **Richard Bellman** in the 1950s for optimal control problems. It decomposes a decision into:

- States and actions
- Transition probabilities
- Immediate and future rewards

This is exactly the structure of a **Markov Decision Process (MDP)**.

Key insight: RL extends Dynamic Programming to cases where the environment model is **unknown** — the agent learns from data instead.



Three Historical Roots of RL

Modern reinforcement learning emerged from the **convergence of three independent research traditions** in the 1980s.

Animal Psychology

Thorndike and others studied **trial-and-error learning** in animals — how behaviors are reinforced by positive outcomes.

Optimal Control Theory

Developed the **mathematical tools** for sequential decision-making, including Dynamic Programming and the Bellman equation.

Temporal Difference Methods

Combined **online learning** with Bellman-style value updates, bridging psychology and mathematics.

RL and Neuroscience

The Dopamine Connection

Research suggests the human brain implements mechanisms strikingly similar to RL algorithms.

- **Better than expected** → dopaminergic neurons fire strongly (positive TD error)
- **Worse than expected** → dopamine signal decreases (negative TD error)

This mirrors the **Temporal Difference (TD) error** formula used in RL algorithms:

This biological parallel has made RL a key tool for understanding **how the brain learns**.

Value Functions in RL

Value functions answer a fundamental question: **how good is it to be in a given state — or to take a given action?** They form the backbone of RL decision-making by estimating long-term expected reward.

State-Value Function

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s]$$

Expected return starting from state *s*, following policy π

Action-Value Function

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$

Expected return taking action *a* in state *s*, then following policy π

State Value vs. Action Value

Both functions evaluate the future, but from different perspectives. Understanding the distinction is key to designing learning algorithms.

vπ(s) — Evaluates Situations

Measures how promising a **state** is, regardless of the next action taken. Useful for understanding the landscape of the environment.

$$v_\pi(s) = \mathrm{E}_\pi[G_t \mid S_t = s]$$

qπ(s,a) — Evaluates Decisions

Measures how good a specific **action** is in a given state. Enables direct comparison between available choices — the basis for algorithms like **Q-learning**.

$$q_\pi(s, a) = \mathrm{E}_\pi[G_t \mid S_t = s, A_t = a]$$

The action-value function **q(s,a)** is especially powerful: it lets the agent select the best action by simply comparing values — no model of the environment needed.

The Bellman Equations

The Bellman equations express a **recursive relationship**: the value of a state equals the immediate reward plus the discounted value of the next state. This decomposition is the mathematical engine behind most RL algorithms.

Bellman Equation for $v_\pi(s)$

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

Bellman Equation for $q_\pi(s, a)$

$$q_\pi(s, a) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})]$$

The recursive structure enables **iterative computation**: values can be refined step by step, without solving the entire problem at once. This is the foundation of Dynamic Programming and TD methods.

Tabular Solution Methods

When States and Actions Are Finite

In tabular methods, value functions are stored as simple lookup tables — one entry per state (or state-action pair). Values are updated iteratively through experience.

Interpretable

Easy to inspect and analyze mathematically

Iterative Updates

Values refined with each new experience

Limited Scale

Only feasible for relatively small state spaces

Dynamic Programming

DP uses Bellman equations **directly** to compute exact value functions — but requires a complete model of the environment (transition probabilities + reward function).

1

Policy Evaluation

Compute *v_{\pi}* for the current policy by iterating the Bellman equation until convergence

2

Policy Improvement

Update the policy greedily: select actions that lead to higher values

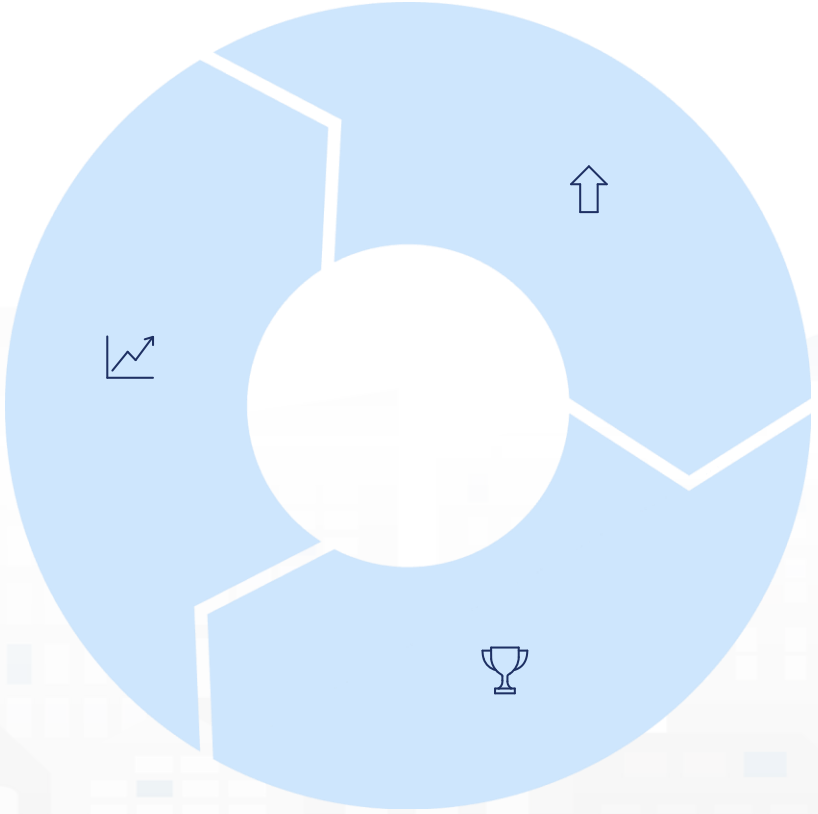
3

Optimal Policy

Repeat until the policy stabilizes — guaranteed convergence to the optimum

Generalized Policy Iteration (GPI)

GPI is the unifying principle behind virtually all RL algorithms. Evaluation and improvement interact continuously, each driving the other toward optimality.



Policy Improvement

Select actions with higher estimated value

Policy Evaluation

Estimate state values under the current policy

Convergence

Cycle continues until an optimal policy is reached

Most RL algorithms — from value iteration to actor-critic methods — can be viewed as specific implementations of this GPI cycle.

Monte Carlo Methods

Learning from Experience

Monte Carlo methods are **model-free**: the agent learns purely from observed episodes, without any knowledge of transition dynamics or rewards in advance.

Core idea: Run many episodes, collect returns, and estimate value functions as **empirical averages** over observed outcomes.

$$v_\pi(s) \approx \frac{1}{N} \sum_{i=1}^{N} G_t^{(i)}$$

Model-Free

No environment model required

Episodic

Learns from complete episodes

Unbiased

Uses actual sampled returns

Simple

Easy to implement and understand

Monte Carlo Methods

Despite their simplicity, Monte Carlo methods face practical constraints that motivated the development of more advanced techniques.

1

Must Wait for Episode End

Value updates happen only **after the full episode** completes — no online learning during the episode.

2

Slow on Long Episodes

In environments with lengthy sequences, learning is significantly **delayed and inefficient**.

3

Not for Continuing Tasks

Unsuitable when there is **no terminal state** and interaction runs indefinitely.

4

High Variance

Estimates based on single-episode returns can be **noisy**, requiring many samples to converge.

These limitations led to the development of **Temporal Difference (TD) methods**, which blend the model-free nature of Monte Carlo with the online updates of Dynamic Programming.