

Reinforcement Learning

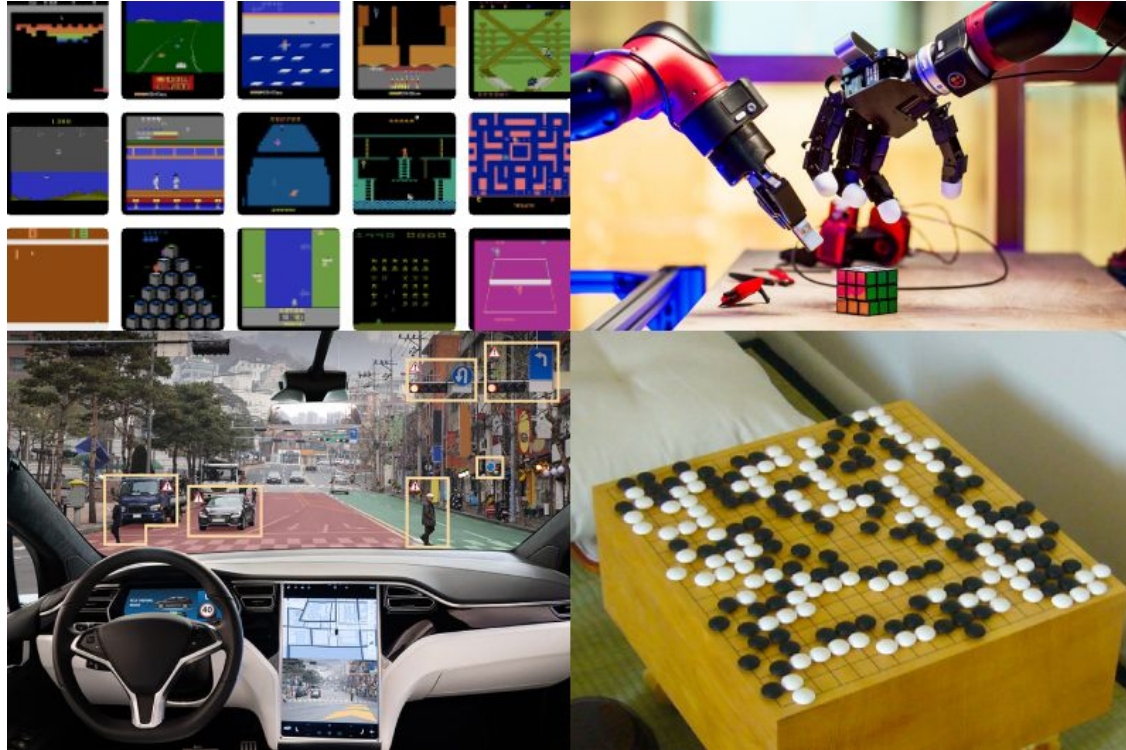
Farid Karimi & Shayan Shahrabi
Spring 2025



Table Of Contents

- Getting Started with Reinforcement Learning
- Bashing with a Classic Problem
- An Overview of Different Approaches
- Focusing on The New Wave & Modern Techniques
- How Reinforcement Learning Is Used Today!

Reinforcement Learning : Robots, Games, Real Word



Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies



Class of Learning Problems : Supervised Learning

Data: (x, y)

x is data, y is label

Goal: function approximation

Examples: Classification, regression,
object detection, semantic
segmentation, image captioning, etc.



→ Cat

Classification

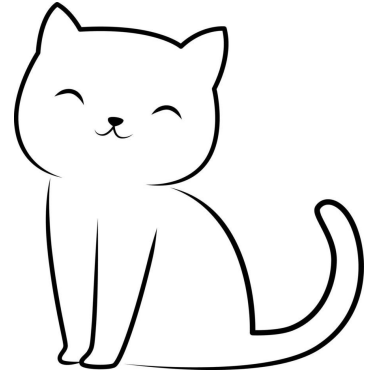
Class of Learning Problems : Unsupervised Learning

Data: x

x is data, no labels!

Goal: Learn underlying structure/pattern

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.



This thing is similar to this other thing!

What is Reinforcement Learning?!

A Not So Formal Definition

The Learning Skill of Humankind

The Big Goal Is To Model 3 Elements

Sensation

Action

Goal

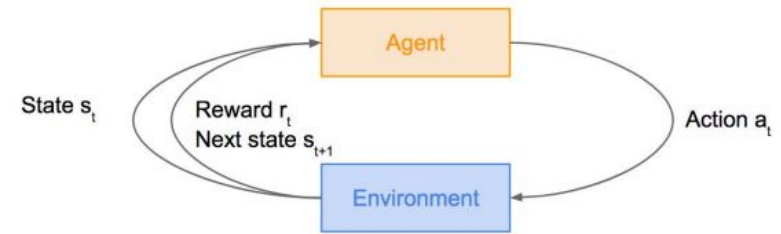
Class of Learning Problems : Reinforcement Learning

Data: State-action pairs

Goal: Learn how to take actions in order to maximize reward

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

N.B There are nice connections with previous classes



The New Trend

Moving from task specific approaches to general solutions

The Elements of an RL Problem

Policy (strategy, a level more abstract)

Reward (easiest to explain) – short term

Value Function (the heart of the whole story) – long term

Model (e.g. model free approaches)

Reinforcement Learning: Key Concepts



Agent

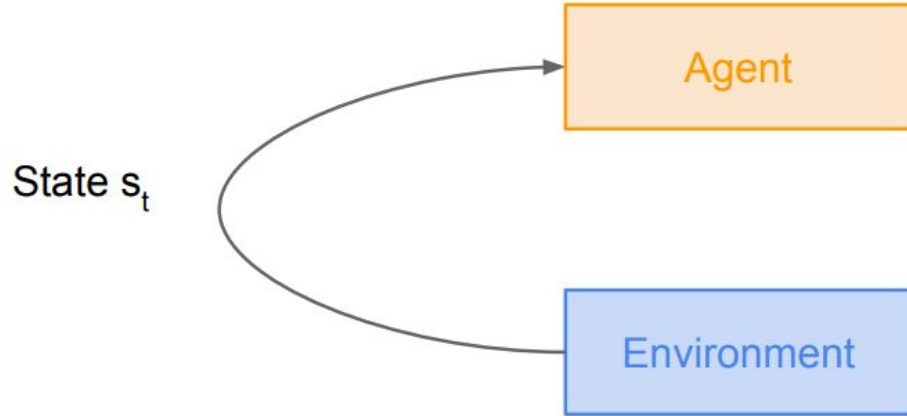
The diagram consists of two rectangular boxes stacked vertically. The top box is orange with an orange border and contains the word 'Agent' in orange text. The bottom box is light blue with a blue border and contains the word 'Environment' in blue text. There are no lines or arrows connecting the two boxes.

Environment

Agent: Takes actions.

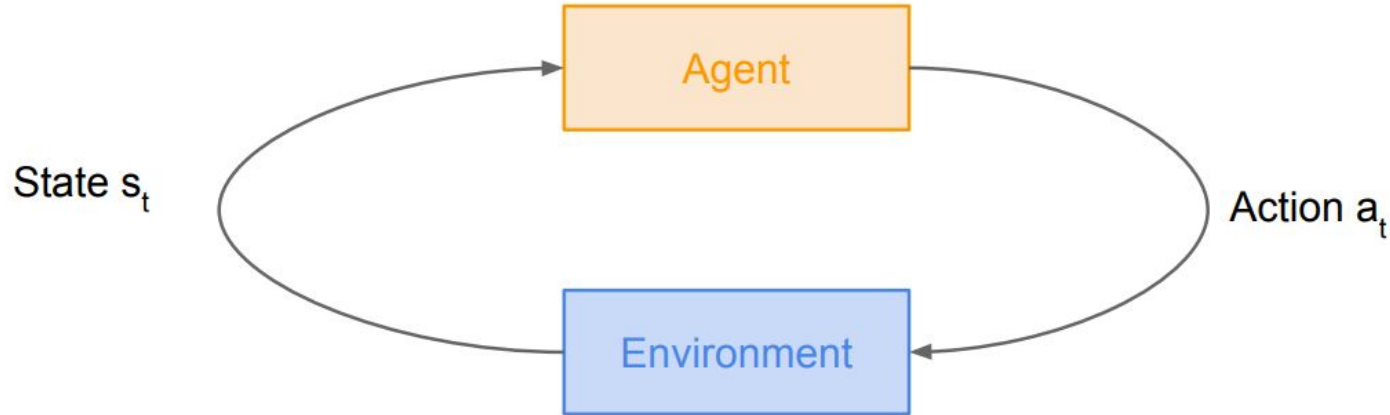
Environment: the world in which agent exists and operates.

Reinforcement Learning: Key Concepts



State: the situation which the agent perceives.

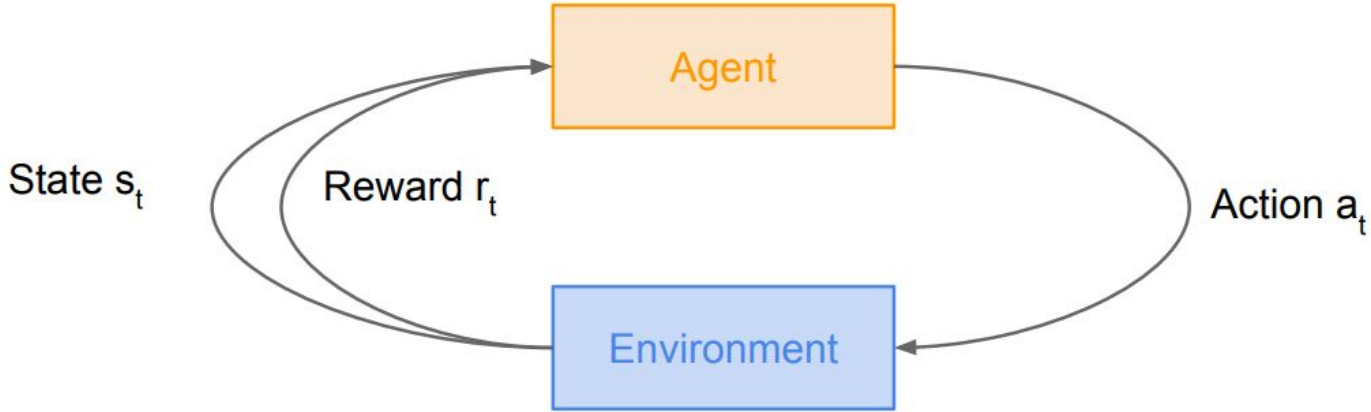
Reinforcement Learning: Key Concepts



Action: a move the agent can make in the environment.

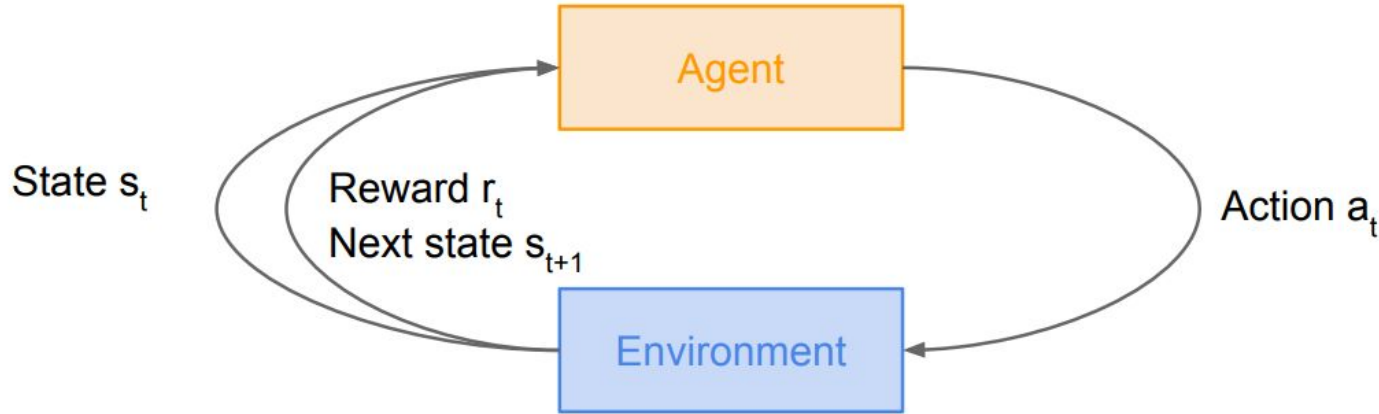
Action space A: the set of possible actions an agent can make in the environment.

Reinforcement Learning: Key Concepts



Reward: feedback that measures the success or failure of the agent's action.

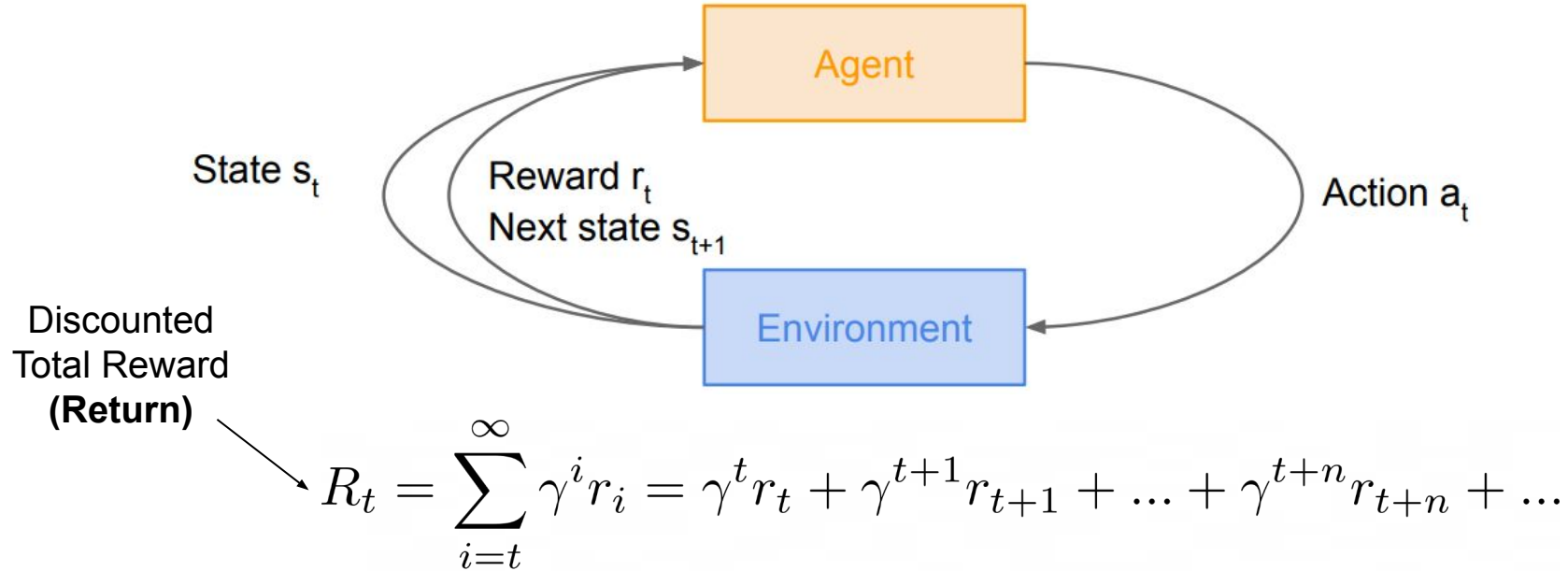
Reinforcement Learning: Key Concepts



Total Reward
(Return)

$$R_t = \sum_{i=t}^{\infty} r_i = r_t + r_{t+1} + \dots + r_{t+n} + \dots$$

Reinforcement Learning: Key Concepts



γ : discount factor; $0 < \gamma < 1$

Also a Famous Dilemma

Exploration/Exploitation

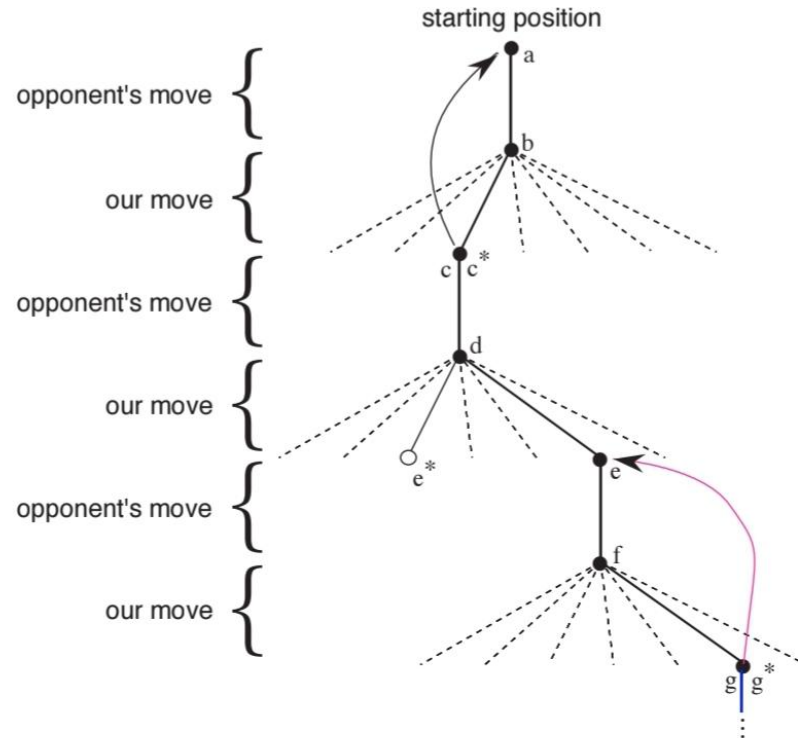
A Simple Example

X	O	O
O	X	X
		X

A Brief Look at Some Game Theory

1. Players play in order
2. Nothing is hidden
3. Deterministic moves only (no dice rolling, etc)

The Sequence Of a Finite- Deterministic Game Tree



The Only Caveat?

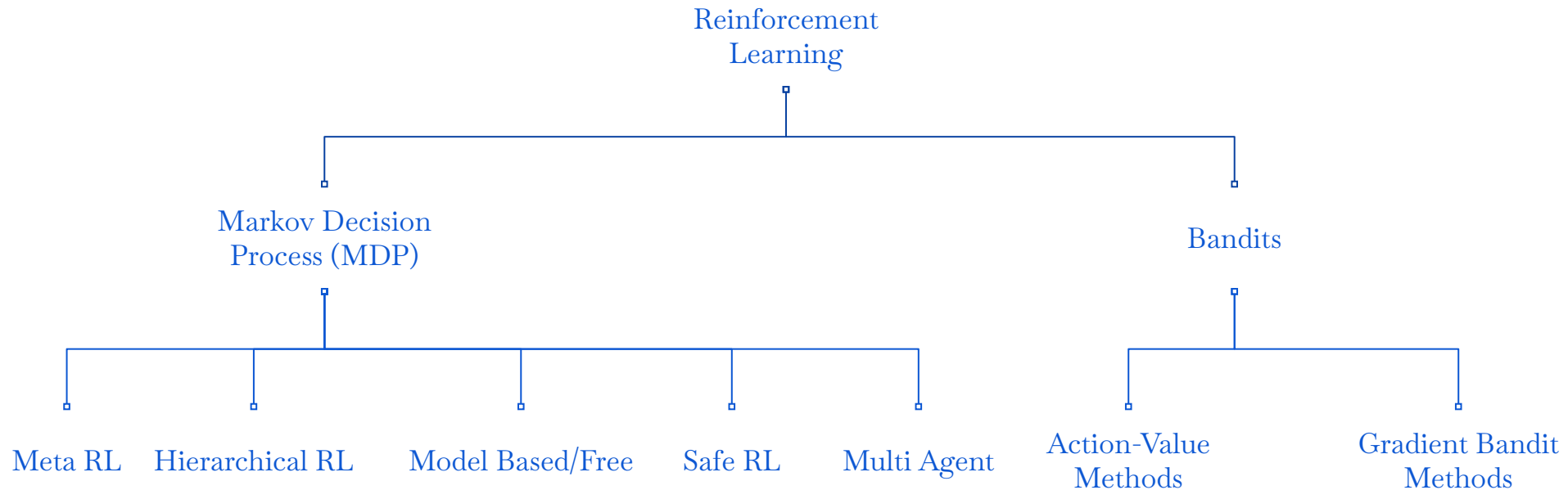
The Tree is just Gigantic!

Solution?

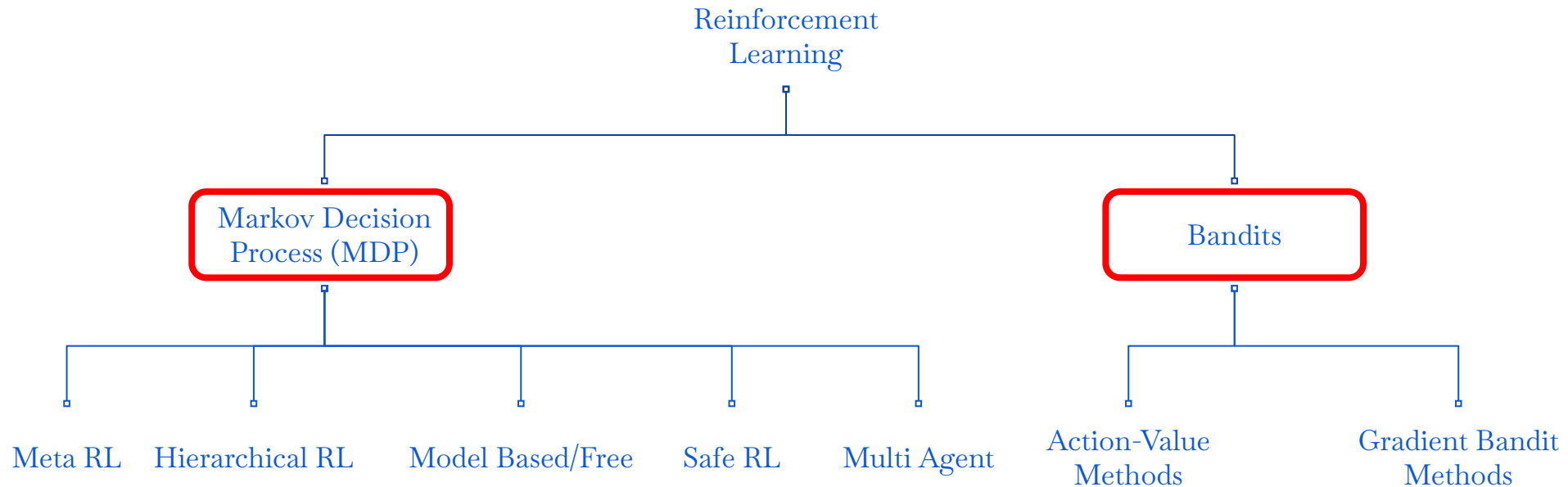
So we suppose having an imperfect opponent

An Overview of Different Categories of RL

Main Approaches



Main Approaches



MDP Vs. Bandits

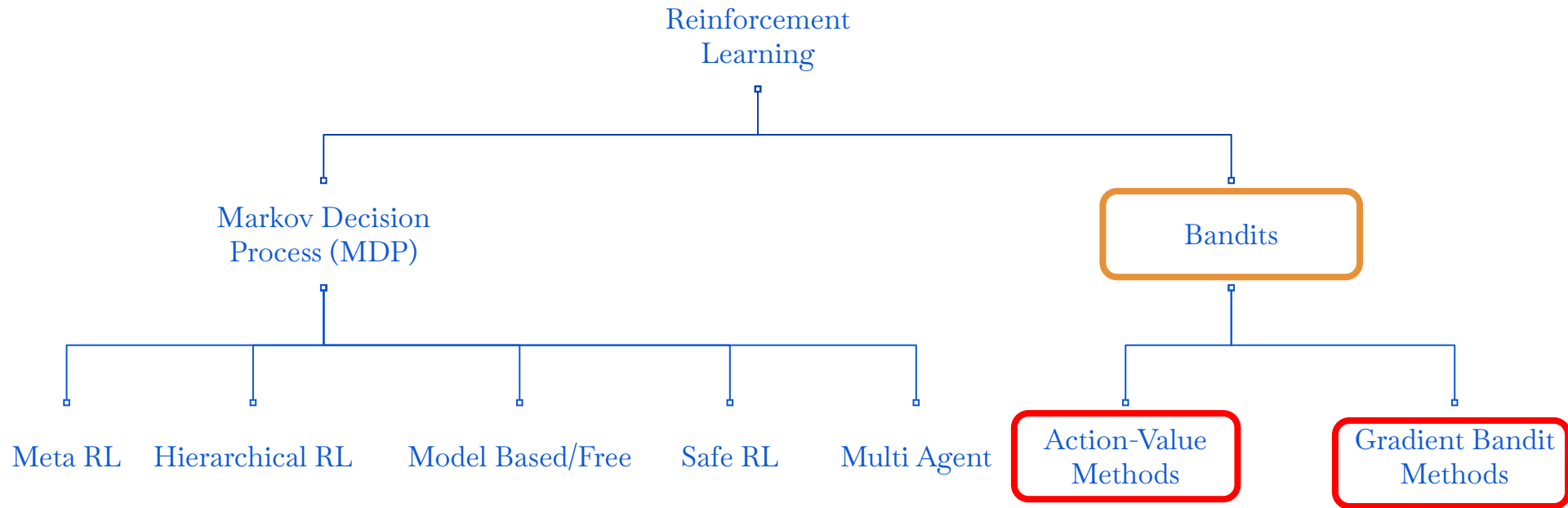
Bandits

- No concept of "state"
- Only choose actions (arms) with unknown rewards
- Focused on *immediate* reward
- Simpler and faster to solve

Markov Decision Processes

- Involves states and transitions
- Actions affect both rewards and future states
- long-term reward
- Requires planning ahead (policy learning)

Main Approaches



Action-Value Methods

Core Idea:

- Estimate the average reward for each action $\rightarrow Q(a)$
- Pick the action with the highest estimated value

Key Methods:

- Incremental update rule, ϵ -greedy strategy

When to Use:

- ✓ Easy to implement
- ✗ Can be slow to adapt if rewards change over time

Gradient Bandit Methods

Core Idea:

- Learn a preference score for each action
- Convert preferences into probabilities using softmax \rightarrow stochastic policy

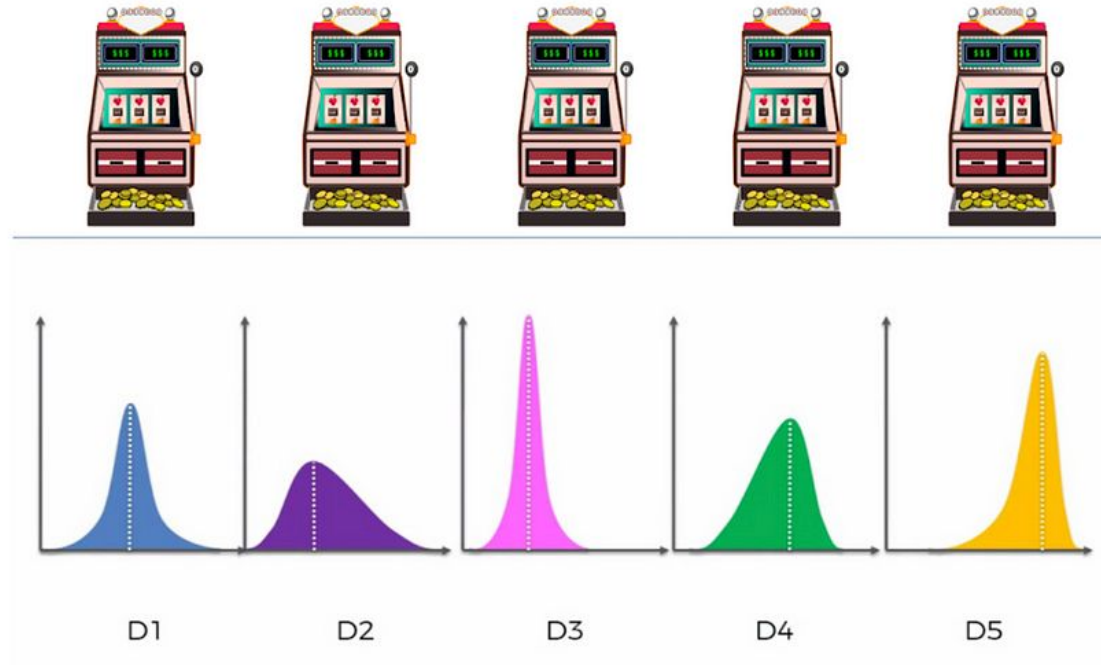
Key Methods:

- Gradient **ascent** on expected reward
- Softmax + baseline to reduce variance

When to Use:

- ✓ Natural way to model stochastic policies
- ✗ Sensitive to learning rate and parameter tuning

A Famous Problem: The Multi-Arm Bandit Problem



The Multi-Arm Bandit Problem – Cont.

Exploration vs. Exploitation dilemma

The Multi-Arm Bandit Problem – Cont.

- **Actions:** Choosing one of k arms.
- **Rewards:** Each arm gives a reward – a fixed & unknown probability distribution
- **No state, no transitions** — *stateless problem*

The Multi-Arm Bandit Problem – Cont.

- **Exploitation:** Choose the arm that has given the best reward so far
- **Exploration:** Try other arms to gather more information — maybe something better exists
- **Challenge:** If you only exploit, you might miss out. If you explore too much, you lose rewards

The Multi-Arm Bandit Problem – Cont.

- **Action-Value Methods**
- Estimate average reward for each arm

$$Q_t(a) = \frac{R_1 + R_2 + \cdots + R_{N_t(a)}}{N_t(a)}$$

- Use ϵ -greedy technique

The Multi-Arm Bandit Problem – Cont.

Gradient Bandit Methods

- Assign preferences to each arm
- Use softmax to convert preferences into probabilities
- Adjust preferences using gradient ascent on expected reward

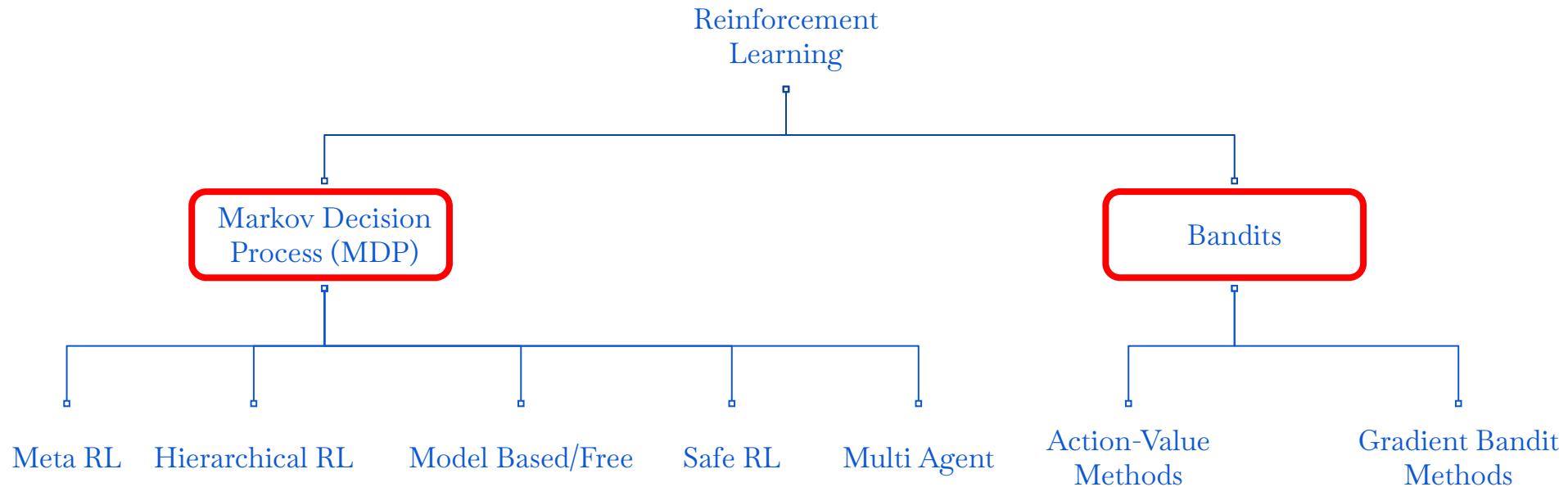
The Multi-Arm Bandit Problem – Cont.

Real-World Applications

- A/B Testing (web design, ads)
- Recommendation systems (Netflix, YouTube)
- Clinical trials (testing different drugs)
- Finance (portfolio selection reward)

The End of Classical Approaches

Main Approaches



Model-Free Reinforcement Learning

The agent learns a policy or value function directly from experience without explicitly building a model of the environment. "Trial and error" focused.

Defining the Q-function

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

Total reward, R_t is the discounted sum of all rewards obtained from t.

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

The Q-function captures the expected total future reward and agent state s , can receive by executing a certain action, a .

What is a "Policy" in Reinforcement Learning?

A policy, denoted by π , is a mapping from states to actions.

- **Input:** The current state of the environment (S).
- **Output:** The action (A) to take.

Two types of policies:

1. **Deterministic ($\pi(s)=a$):** In a given state, the policy always outputs the *same* action.
2. **Stochastic ($\pi(a|s)$):** In a given state, the policy outputs a *probability distribution* over all possible actions. The agent then samples from this distribution to choose its action.

How to take actions given a Q-function ?

$$Q(s_t, a_t) = \mathbb{E}[R_t | s_t, a_t]$$

Ultimately, the agent needs a policy $\pi(s)$, to infer the best action to take at its state, s .

Strategy : the policy should choose an action that maximizes future rewards.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q(s, a)$$

Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \operatorname{argmax}_a Q(s, a)$$

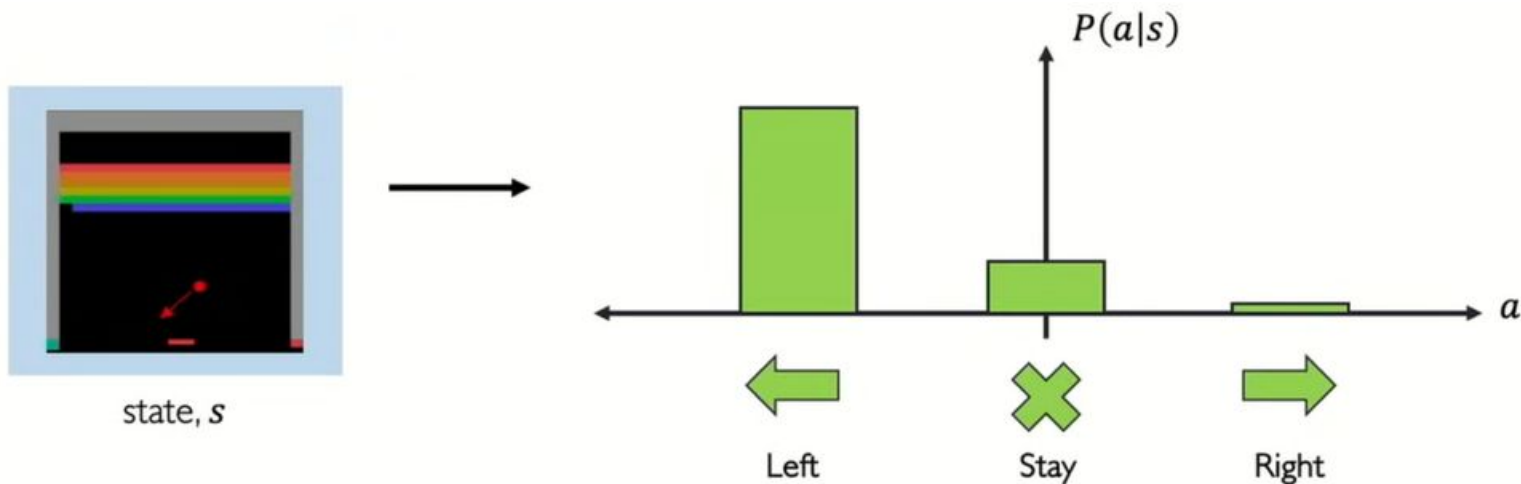
Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

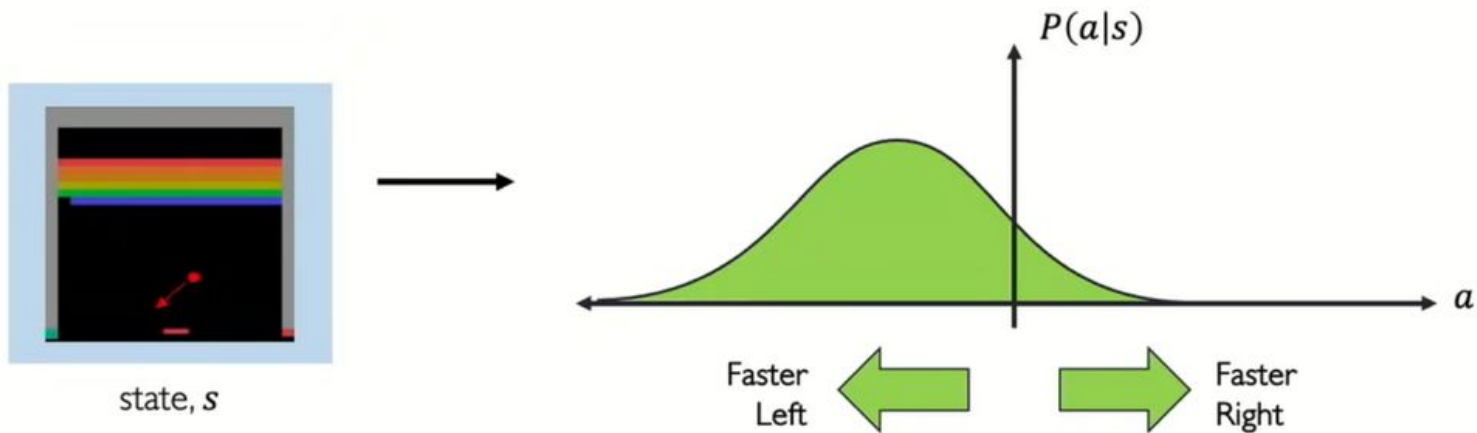
Discrete vs Continuous Action Space

Discrete: which direction should I move ?



Discrete vs Continuous Action Space

Continuous: How fast should I move



Value learning + Policy Learning
=
Actor-Critic

Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \operatorname{argmax}_a Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Two Key Policies: Behavior vs. Target

The Behavior Policy (μ):

- The policy the agent *actually uses* to explore and interact with the environment.
- This policy often includes an exploration component (e.g., ϵ -greedy, where it sometimes takes random actions) to ensure it tries a variety of actions.

The Target Policy (π):

- The policy the agent is trying to learn and improve.
- This is the policy that we ultimately want to become the optimal policy (π^*).
- It is typically a purely greedy or "exploitative" policy that we want to follow once learning is complete.

On & Off Policy Learning

- In **on-policy** learning, the agent learns the value function for the **same policy it is currently using to make decisions and gather experience**. The policy being evaluated and improved is the same policy that dictates the agent's actions
- In **off-policy** learning, the agent learns the value function for a **target policy** π (often the optimal greedy policy) while using a potentially different **behavior policy** μ to explore the environment and gather experience.

Off Policy

Why Q-Learning is Off-Policy

- **The Behavior Policy (μ):** The policy the agent actually uses to explore the environment and generate experience tuples (S,A,R,S') . To learn effectively, this policy must be exploratory (e.g., ϵ -greedy, where it sometimes chooses a random action).
- **The Target Policy (π):** The policy that the agent is trying to learn. In Q-learning, this is the optimal, greedy policy defined by the Bellman Optimality Equation.

Q	left	right	up	down
S1	-0.5	1	2.1	1.3
S2	0.5	0.75	-0.5	1.5
S6	-1.2	1.2	0.7	1.7
.

S1



-1

-1

-1

-1

-1

-1

-10

+10

S1



-1



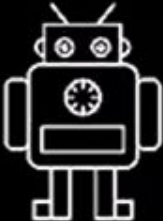
-1

-1

-1

-10

+10

-1	S2 	-1
-1	-1	-1
-1	-10	+10

Bellman Equation

$$Q(S_1, \text{right})_{\text{observed}} = R(S_2) + \gamma \max_a Q(S_2, a)$$

$$= -1 + 0.1 * 1.5 = -0.85$$

Temporal Difference

$$\text{TD Error} = Q(S_1, \text{right})_{\text{observed}} - Q(S_1, \text{right})_{\text{expected}}$$

$$= -0.85 - 1 \quad \quad \quad = -1.85$$

Update the Q-Table based on Target Policy

$$Q(S_1, \text{right}) = Q(S_1, \text{right}) + \alpha \times \text{TD Error}$$

$$= 1 + 0.1 * -1.85 = \mathbf{0.815}$$

Q	left	right	up	down
S1	-0.5	0.815	2.1	1.3
S2	0.5	0.75	-0.5	1.5
S6	-1.2	1.2	0.7	1.7
.

-7

-7

-7

-7

-7

-7

-7

-10



Q	left	right	up	down
S1	0.531	3.421	0.529	3.185
S2	1.695	1.486	0.925	1.658
S6	1.573	2.457	1.520	0.137
.

Target Policy

$$Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$$

Diagram illustrating the Target Policy equation for Q-learning:

- New Q Value**: $Q(s, a)$ (left side)
- Old Q Value**: $Q(s, a)$ (first term on the right)
- Learning Rate**: α ($0 \sim 1$)
- Reward**: r
- Discount Rate**: γ ($0 \sim 1$)
- Maximum Q value of transition destination state**: $\max_{a'} Q(s', a')$ (underlined in red)
- TD error**: $r + \gamma \max_{a'} Q(s', a') - Q(s, a)$ (bracketed in red)

Reinforcement Learning & Deep Q-Learning

Reinforcement Learning

- An agent selects the best action based on the current state to maximize rewards.
- Example: In Pac-Man, the state includes the game world, enemies, and pac-dots; actions are movements (up/down/left/right).

Q-Learning (State-Action Value)

- RL learns the value of (state, action) pairs over time, stored as $Q(\text{state}, \text{action})$.
- Maintaining a full Q-table becomes infeasible as state-action space grows.

Deep Reinforcement Learning (Deep RL)

- Neural Networks approximate Q values instead of storing a table.
- **$Q = \text{Neural Network}(\text{state}, \text{action}) \rightarrow$** Predicts rewards.
- **Deep Neural Networks (DNNs)** generalize better for complex problems.

Reinforcement Learning & Deep Q-Learning

Reinforcement Learning

- An agent selects the best action based on the current state to maximize rewards.
- Example: In Pac-Man, the state includes the game world, enemies, and pac-dots; actions are movements (up/down/left/right).

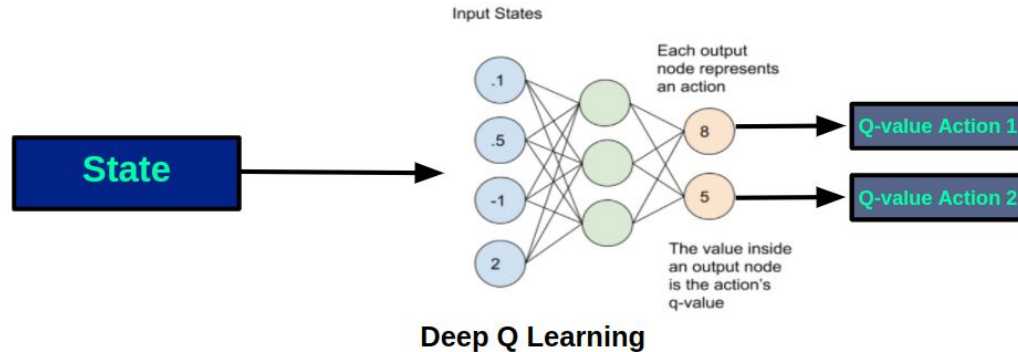
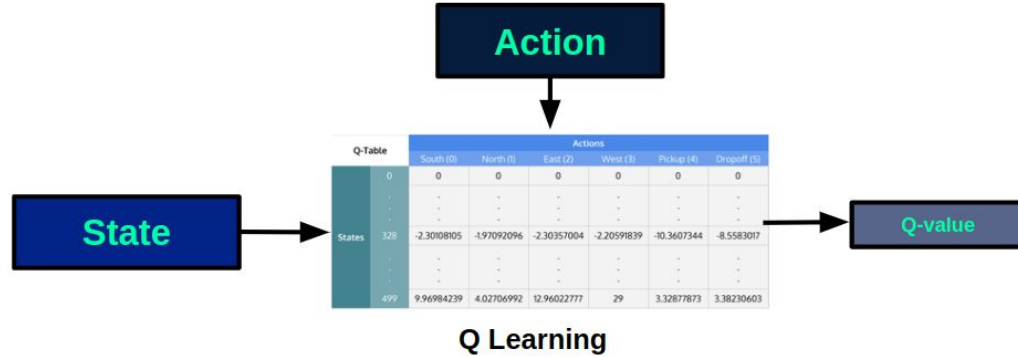
Q-Learning (State-Action Value)

- RL learns the value of (state, action) pairs over time, stored as $Q(\text{state}, \text{action})$.
- Maintaining a full Q-table becomes infeasible as state-action space grows.

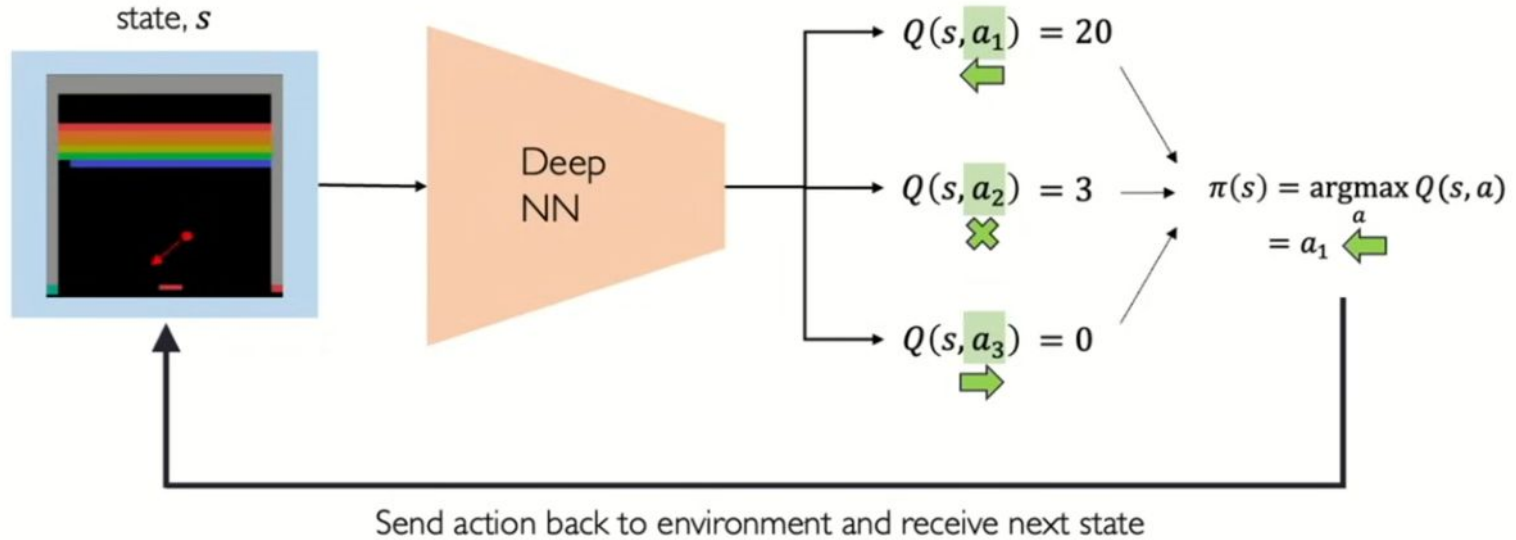
Deep Reinforcement Learning (Deep RL)

- Neural Networks approximate Q values instead of storing a table.
- **$Q = \text{Neural Network}(\text{state}, \text{action}) \rightarrow$** Predicts rewards.
- **Deep Neural Networks (DNNs)** generalize better for complex problems.

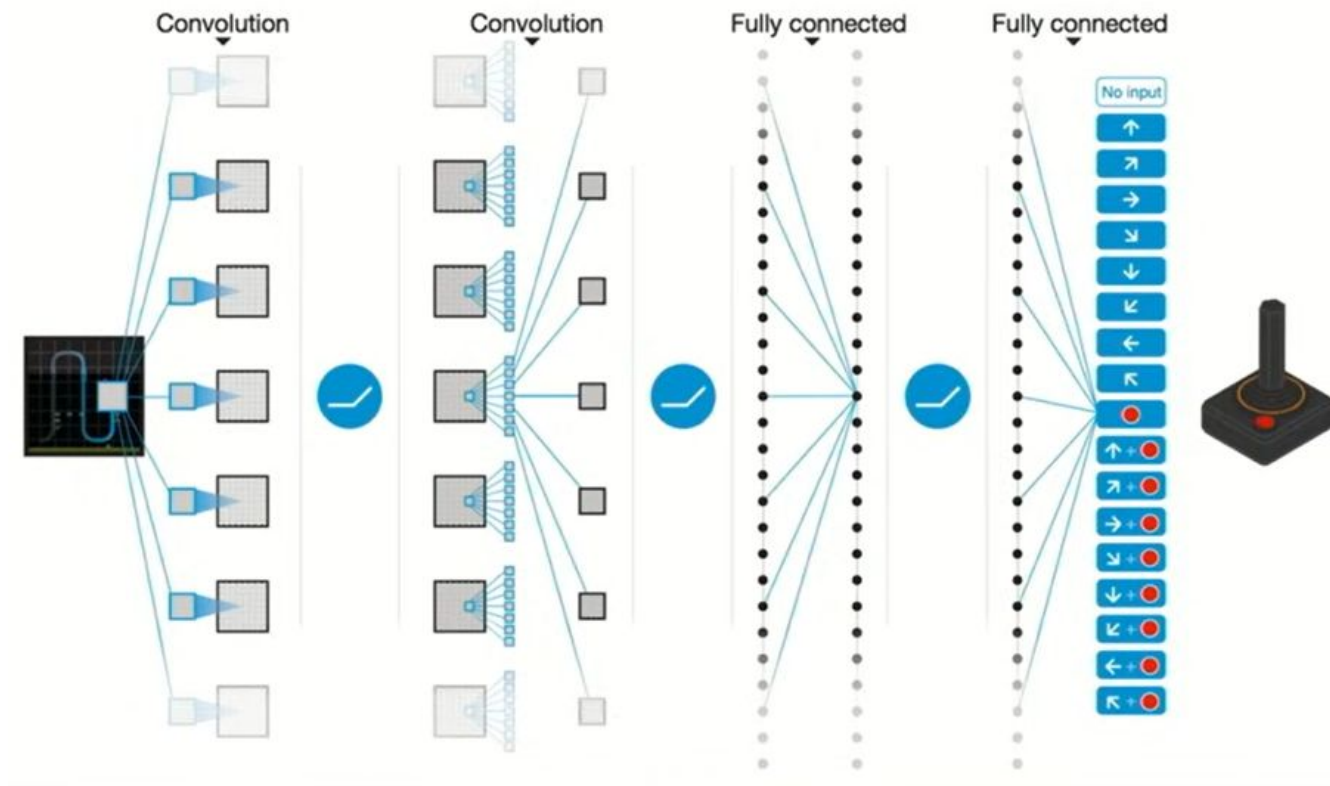
Reinforcement Learning & Deep Q-Learning



Deep Q Network Summary

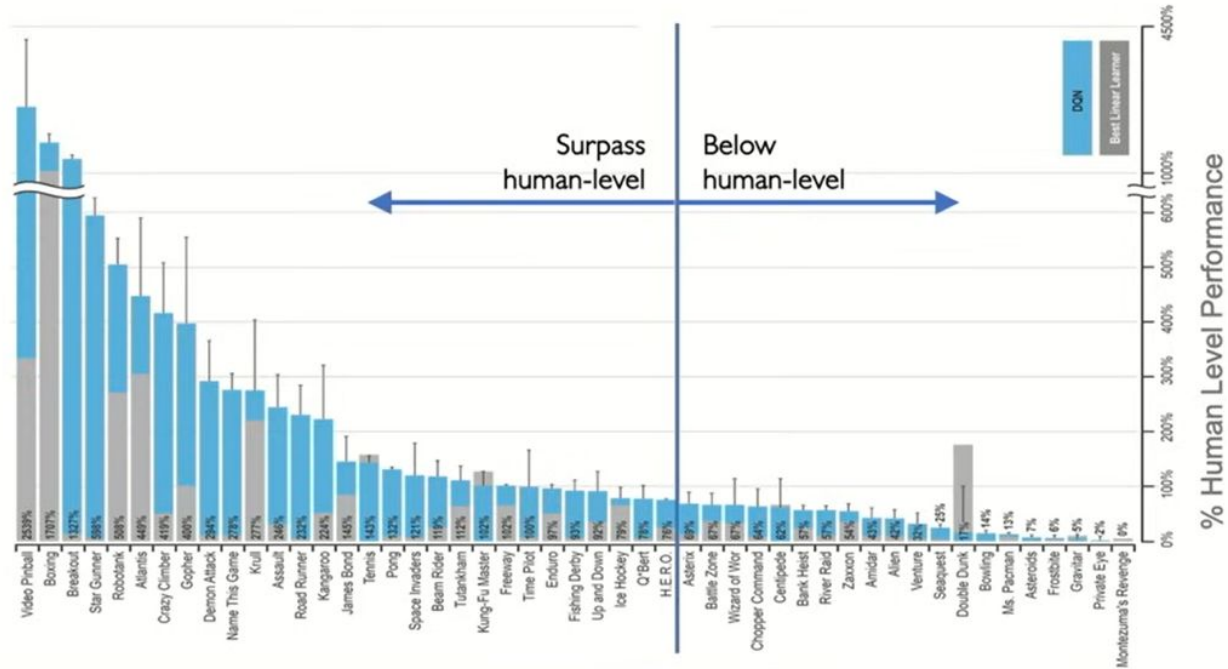


Deep Q Network Atari



Deep Q Network Atari

Just by getting the pixel on the screen of all the atari games, it surpassed human level in more than 50% of the games



Downsides of Q-learning

Complexity:

- Can model scenarios where the action space is discrete and small
- Cannot handle continuous action spaces

Flexibility:

- Policy is deterministically computed from the Q function by maximizing the reward → cannot learn stochastic policies

**To Address these, consider a new class of RL training algorithms:
Policy gradient methods**

On Policy

SARSA: Learning on the Job

How it Works:

- Learns an action-value function $Q(s,a)$.
- Update rule: $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
- It uses the quintuple (S,A,R,S',A') to update $Q(S,A)$:
 1. In state S , take action A .
 2. Observe reward R and next state S' .
 3. From S' , pick action A' *using the current policy*.
 4. Update $Q(S,A)$ based on $Q(S',A')$

Reinforcement Learning Algorithms

Value Learning

Find $Q(s, a)$

$$a = \operatorname{argmax}_a Q(s, a)$$

Policy Learning

Find $\pi(s)$

Sample $a \sim \pi(s)$

Understanding Policy Optimization: Gradient-Based vs. Gradient-Free Approaches

Optimizing Our Agent's "Brain"

- **Goal in Policy-Based RL:** Directly learn a policy, $\pi_{\theta}(a|s)$.
 - θ = policy parameters (e.g., neural network weights).
- **Objective:** Find parameters θ that maximize expected total reward, $J(\theta)$ where $J(\theta)$ is usually the expected total reward.
- **Question:** How do we find the best θ ?

Gradient-Based Policy Methods : The Slope-Climbers

If we can figure out the "slope" of our reward mountain, we can climb it!

What they do:

- Calculate or estimate the **gradient**: $\nabla_{\theta} J(\theta)$.
 - This gradient points in the direction of the steepest increase in rewards.
- Update policy parameters θ by moving in that direction (gradient ascent):
 - $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$

Gradient-Based Methods - How?

1. Interact with the environment using the current policy π_θ .
2. Estimate $\nabla_\theta J(\theta)$ based on the experiences.
3. Update θ .
4. Repeat.

Actor-Critic Methods

Combines the strengths of policy-based and value-based learning by using two components:

1. **The Actor (Policy):**

- Learns and executes the policy $\pi(a | s; \theta)$.
- **Job:** Decides which action A to take in state S .
- *The "Performer" or "Decision-Maker."*

2. **The Critic (Value Function):**

- Learns a value function (e.g., $V(s; w)$ or $Q(s, a; w)$).
- **Job:** Evaluates how good the Actor's actions are or how good states are.
- *The "Evaluator" or "Feedback Provider."*

Goal: Learn a better policy by having the Actor learn from the Critic's feedback.

Actor-Critic - Interaction & Advantages

1. **Actor acts:** Chooses action A in state S.
2. **Environment responds:** Agent gets reward R, moves to state S'.
3. **Critic evaluates:** Calculates a feedback signal (e.g., TD Error or Advantage) based on (S,A,R,S').
 - *TD Error example:* $\delta = R + \gamma V(S') - V(S)$
 - This signal indicates if the action was better (+) or worse (-) than expected.
4. **Actor updates:** Adjusts its policy θ based on the Critic's feedback (e.g., if $\delta > 0$, make action A more likely).
5. **Critic updates:** Improves its own value function w to make future evaluations more accurate.

Advanced Gradient-Based Methods - Careful Climbers

- These methods **STILL USE GRADIENTS!**
- Problem with simple gradient ascent: Taking too large a step can lead to performance collapse.
- follow the slope, but take **careful**, constrained steps to ensure stability and avoid falling off a performance cliff.

Advanced Gradient-Based - How They Differ?

TRPO (Trust Region Policy Optimization):

- Maximizes an objective but constrains policy changes using KL divergence.
- Ensures the new policy stays "close" to the old one.

PPO (Proximal Policy Optimization):

- Simpler to implement than TRPO.
- Uses a "**clipped**" objective function or a KL penalty to limit how much the policy changes per update.

Gradient-Free Policy Methods - The Explorers

- **Core Idea:** Optimize policy parameters θ *without* explicitly calculating gradients ($\nabla_{\theta} J(\theta)$).
- **Analogy:** Instead of checking the slope, send out a team of explorers (different policies) in many directions and see who finds the highest ground.
- Treats policy optimization more like a **black-box optimization problem**.

True Gradient-Free - Pros & Cons

Pros:

- Can work even if policy/objective is not differentiable.
- Potentially more robust to very noisy rewards or sharp local optima.
- Often highly parallelizable.

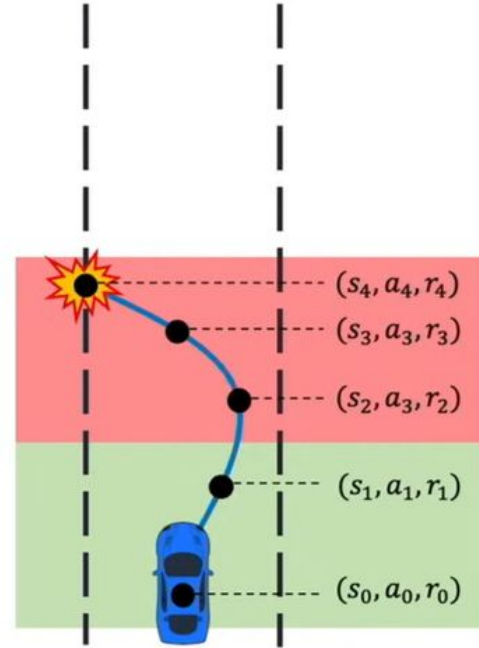
Cons:

- Can be less sample efficient (need more interactions) for high-dimensional θ (like deep NNs) compared to gradient-based methods that have a "sense of direction."
- Parameter space exploration can be challenging.

Training Policy Gradient

Training Algorithm:

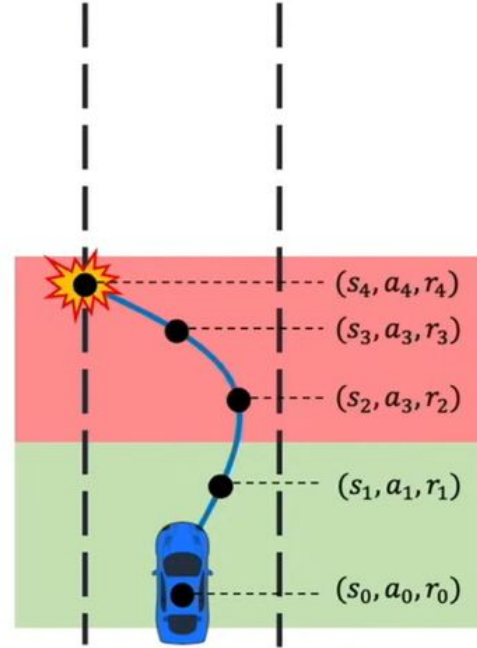
1. Initialize the agent.
2. Run a policy till it terminates.
3. Record all states, actions, rewards.
4. Decrease probability of actions that resulted in low reward.
5. Increase probability of actions that resulted in high rewards.



Training Policy Gradient

Training Algorithm:

1. Initialize the agent.
2. Run a policy till it terminates.
3. Record all states, actions, rewards.
4. **Decrease probability of actions that resulted in low reward.**
5. **Increase probability of actions that resulted in high rewards.**



Training Policy Gradient

Training Algorithm:

1. Initialize the agent.
2. Run a policy till it terminates.
3. Record all states, actions, rewards.
4. **Decrease probability of actions that resulted in low reward.**
5. **Increase probability of actions that resulted in high rewards.**

$$\text{loss} = - \overset{\text{Log-likelihood of action}}{\log P(a_t|s_t)} \underset{\text{reward}}{R_t}$$

$$w' = w + \nabla \text{loss}$$

$$w' = w + \nabla \log P(a_t|s_t) R_t$$

Policy gradient !

Model-Based Reinforcement Learning

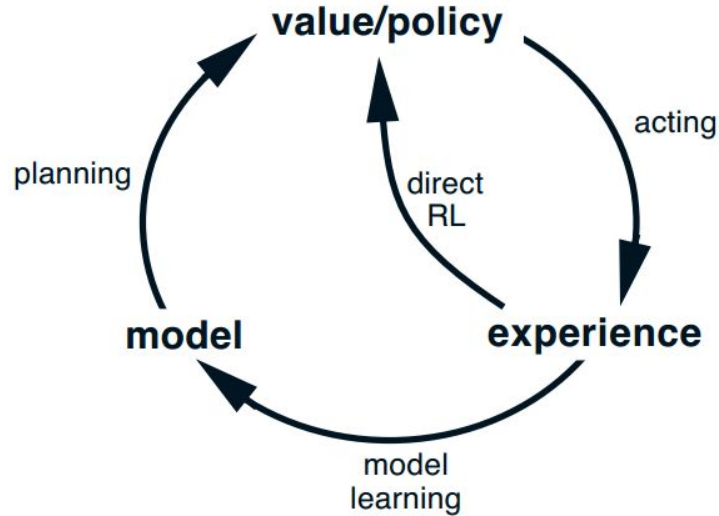
The agent learns a model of the environment (i.e., how the world works – transition probabilities and rewards) and then uses this model for planning or to generate simulated experiences.

Dyna-Q: Learning a Model and Learning from Experience

How it Works:

- Simultaneously learns a model of the environment (from real experiences) and a policy/value function.
- After each real interaction, it performs several "planning" steps:
 - Randomly samples a previously seen state and action.
 - Uses the learned model to predict the next state and reward.
 - Updates its value function/policy using this simulated experience (e.g., with Q-learning).

Dyna-Q: Learning a Model and Learning from Experience



Dyna-Q: Learning a Model and Learning from Experience

Why Invented/Used:

- **Data Efficiency:** Makes much better use of limited real-world experience by supplementing it with cheaper, simulated experience.
- Useful when real interactions are costly or slow.

Hierarchical Reinforcement Learning

Thinking in Layers

The Problem with "Flat" Reinforcement Learning

Standard RL struggles with:

- **Long-Horizon Tasks:** When a reward is very far in the future (e.g., "get a PhD"), it's hard to connect early actions to the final reward.
- **Sparse Rewards:** If rewards are rare, the agent wanders aimlessly without feedback.
- **Sample Inefficiency:** Learning every single low-level action (e.g., "flex this specific muscle") for a high-level task is incredibly slow.

How HRL Works: Options & Goals

Structures the policy in **layers**, much like a manager and employees.

A Common HRL Framework: Options

- **High-Level Policy (Manager):**
 - Does not choose low-level actions (e.g., `move_left`, `move_right`).
 - Instead, it chooses from a set of "Options" or sub-goals.
 - *Example:* In a large building, the manager's options are: "Go to the kitchen," "Go to the elevator," "Exit the building."
- **Low-Level Policy (Employee):**
 - Receives an "Option" (e.g., "Go to the kitchen") from the manager.
 - Its job is to execute the low-level actions required to complete that specific sub-goal.
 - It gets its own "internal" rewards for completing the sub-goal.

Safe Reinforcement Learning

Why Can't We Just "Trial and Error" Everything?

Reinforcement Learning in Real Life

Training Algorithm:

1. Initialize the agent.
2. Run a policy till it terminates.
3. Record all states, actions, rewards.
4. Decrease probability of actions that resulted in low reward.
5. Increase probability of actions that resulted in high rewards.

Reinforcement Learning in Real Life

Training Algorithm:

1. Initialize the agent.
2. Run a policy till it terminates.
3. Record all states, actions, rewards.
4. Decrease probability of actions that resulted in low reward.
5. Increase probability of actions that resulted in high rewards.



Safe Reinforcement Learning

Standard RL: Agents explore freely to maximize rewards, often through extensive trial and error.

The Problem: In the real world, errors can be catastrophic!

- A robot trying to learn to walk can't fall down a staircase.
- An autonomous car can't experiment with collisions.
- A medical system can't test harmful treatments.

Safe RL: A subfield of RL focused on creating algorithms that learn and operate while respecting critical safety constraints, both during and after training.

Approaches to Safe Reinforcement Learning

Modify the Exploration Process:

- Change how the agent explores to avoid dangerous actions.
- **Example:** Use a "teacher" or a baseline safe policy to guide exploration. The agent can only take actions that are deemed safe by this supervisor.
- **Example:** Use a "safety shield" that overrides any unsafe action the learning agent tries to take.

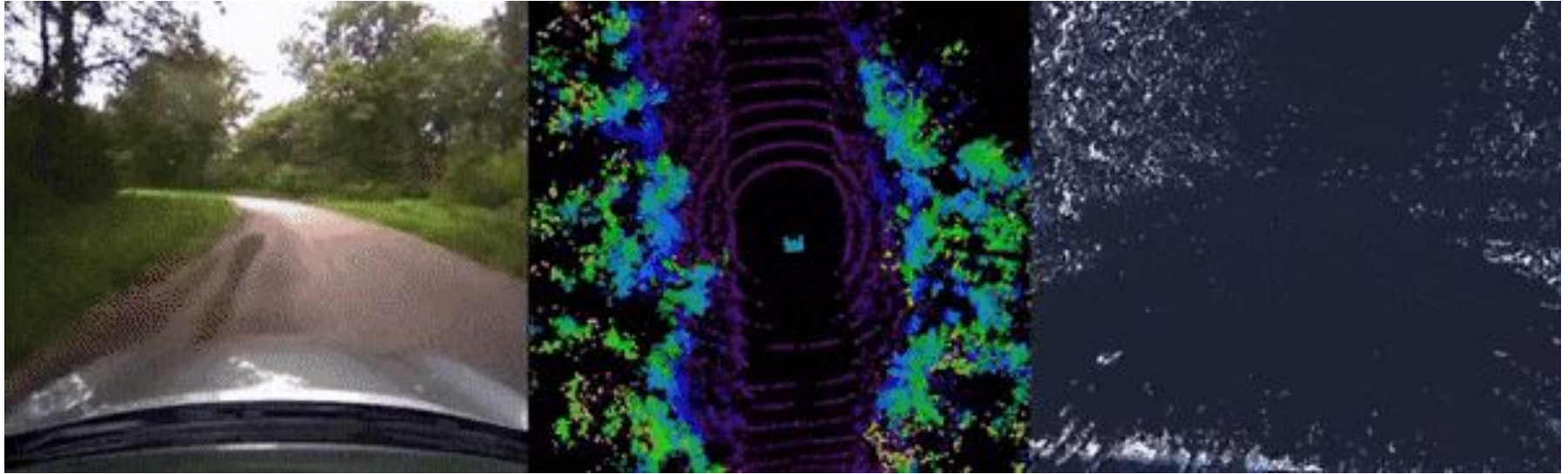
Approaches to Safe Reinforcement Learning

Modify the Optimization Criteria:

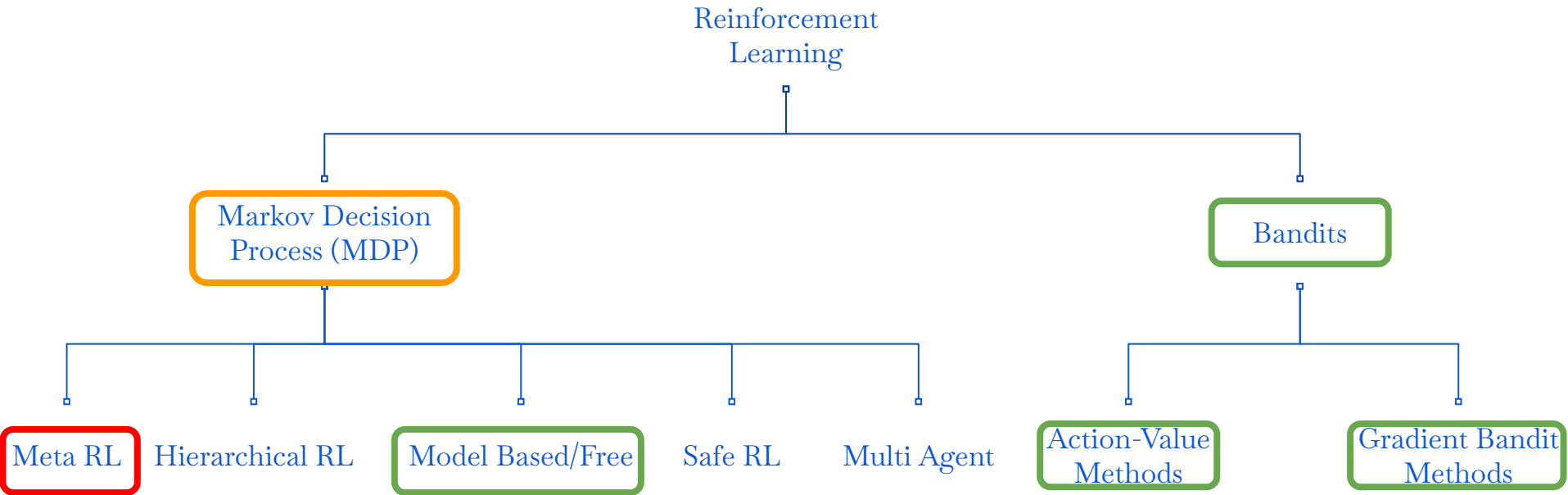
- Change what the agent is trying to maximize. Instead of just maximizing rewards, it has to satisfy safety constraints.
- **Example: Constrained Markov Decision Processes (CMDPs):** The agent must maximize rewards *subject to* the constraint that the expected cost (or "harm") stays below a certain threshold.
- *Analogy:* "Get the highest score possible, but you are not allowed to break any of the rules."

Data-Driven simulation for Autonomous Vehicles

Vista: photorealistic and high-fidelity for training and testing self driving cars



Main Approaches



Meta Reinforcement Learning

The agent learns how to learn. Instead of just learning a policy for a single task, the agent is trained across many tasks to quickly adapt to new ones by leveraging prior experience. It effectively learns an internal learning algorithm, allowing it to generalize and perform well on unseen tasks with minimal data.

Meta Reinforcement Learning

Core Idea:

- Learns how to adapt quickly to new tasks
- Uses prior experience across tasks to "learn to learn"

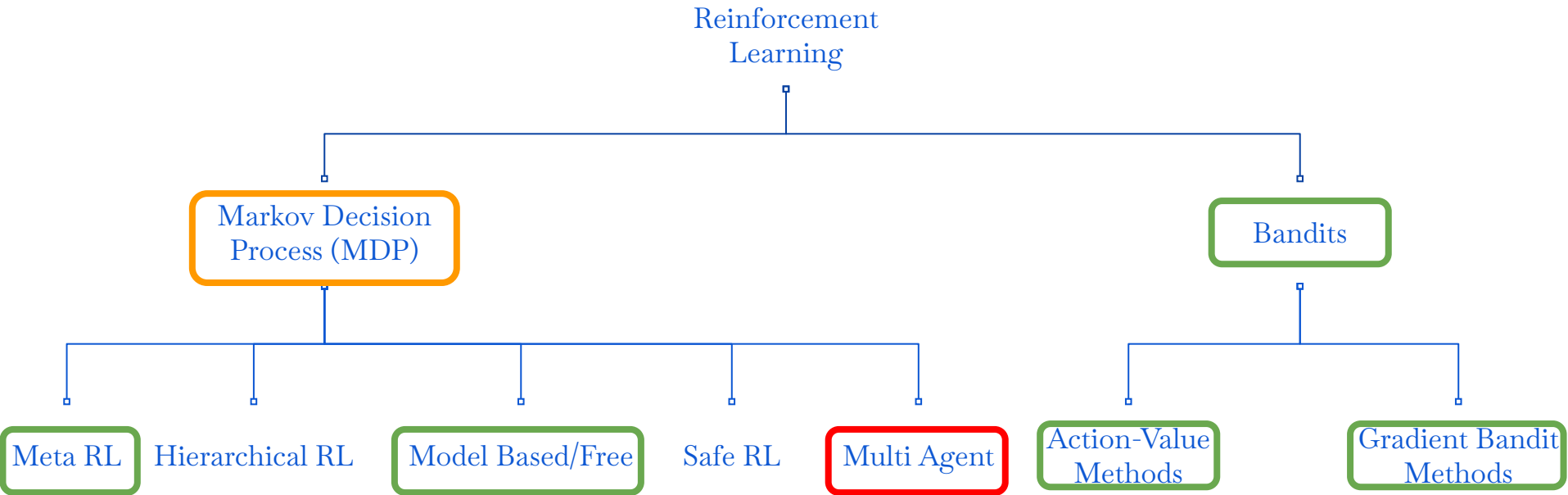
Key Methods:

- MAML (Model-Agnostic Meta-Learning), RL^2 , PEARL

When to Use:

- ✓ Fast adaptation to new environments
- ✗ Needs many tasks during training

Main Approaches



Multi-Agent Reinforcement Learning

Multiple agents learn and interact within a shared environment, either cooperatively, competitively, or both, influencing each other's learning and behavior.

Meta Reinforcement Learning

Core Idea

- Multiple agents learn and make decisions simultaneously, adapting to and influencing each other's strategies.

Key Methods

- Independent Q-Learning, MADDPG, QMIX

When to Use

- ✓ Environments with multiple interacting agents (e.g., games, traffic, markets)
- ✗ Non-stationarity makes training unstable or hard to converge

Main Approaches

