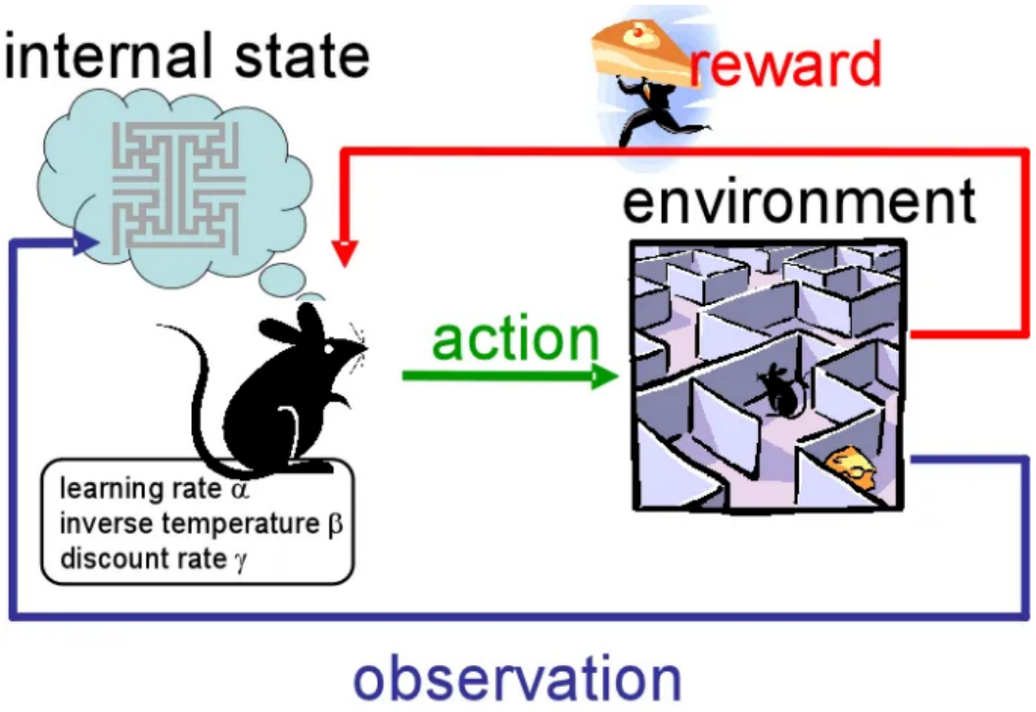
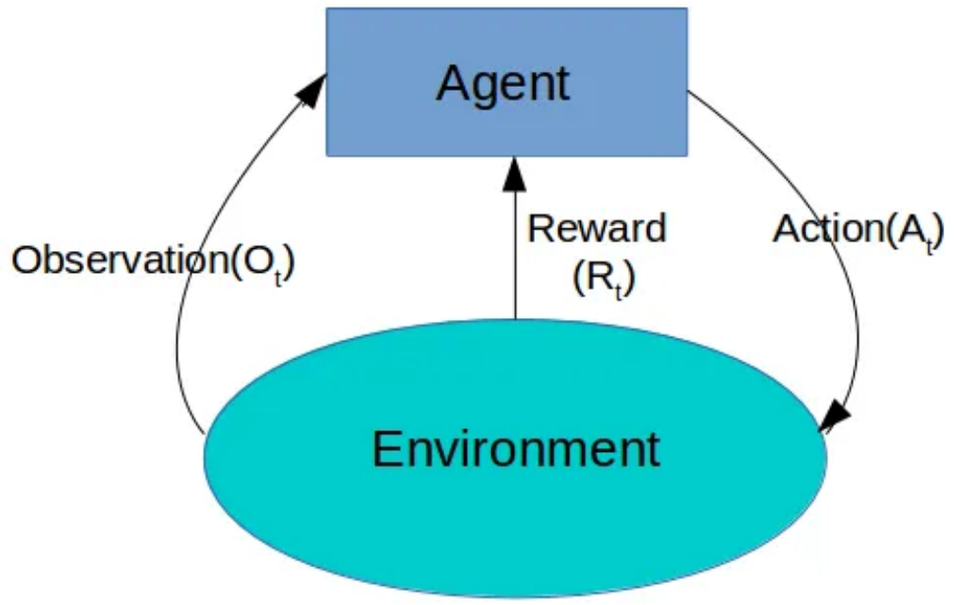
연세대학교
YONSEI UNIVERSITY

Introduction to RL – Part 1

TA. Bogyeong Suh

Reinforcement Learning



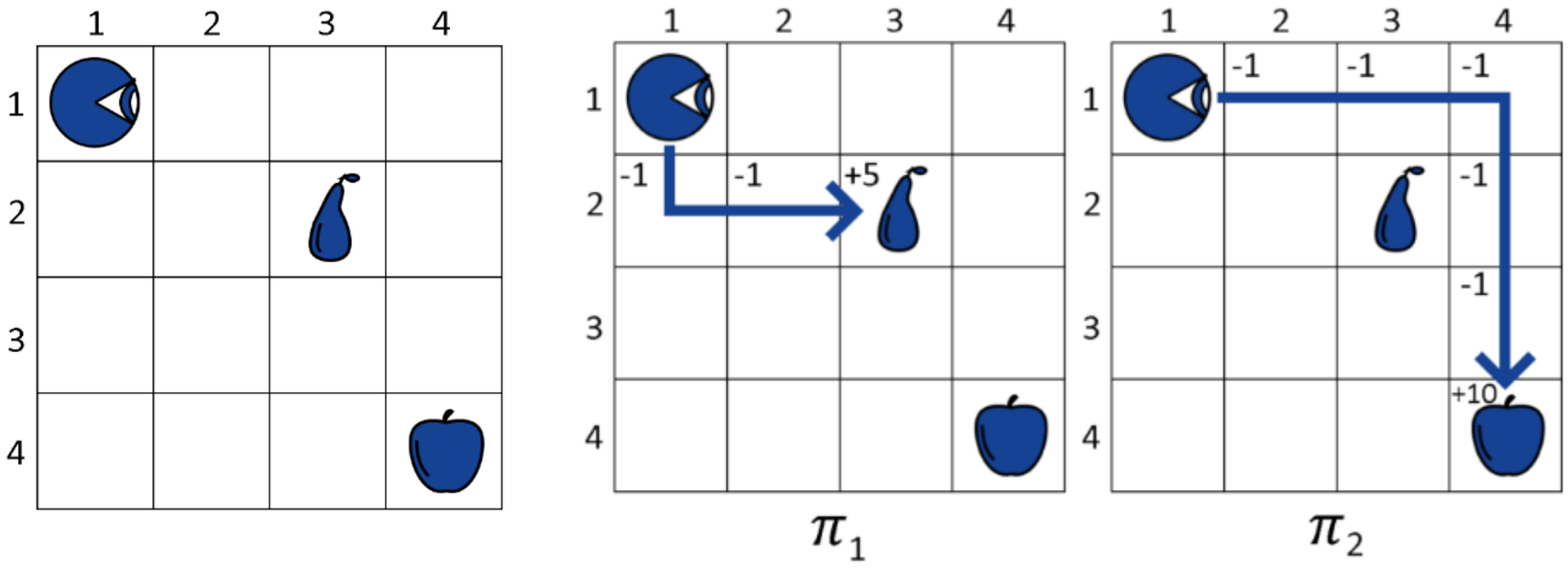


https://becominghuman.ai/the-very-basics-of-reinforcement-learning-154f28a79071

- Goal-oriented Learning, where the primary objective is to **train models to make sequences of decisions by discovering strategies that maximize a reward signal**
- Two primary entities in RL are **agent** and **environment**. Agent learns and make decisions, and environment is where the agent operates.
- Agent interacts with the environment, takes an action based on a policy, receives a reward and observes the next state.

Reinforcement Learning

- States s: A representation of the situation or environment at a given time.
- Action a: Decisions made by the agent that affect the environment.
- Reward r: Feedback received after taking an action in a particular state.
- Policy π: A strategy or mapping from states to actions. Defines the agent's behavior.

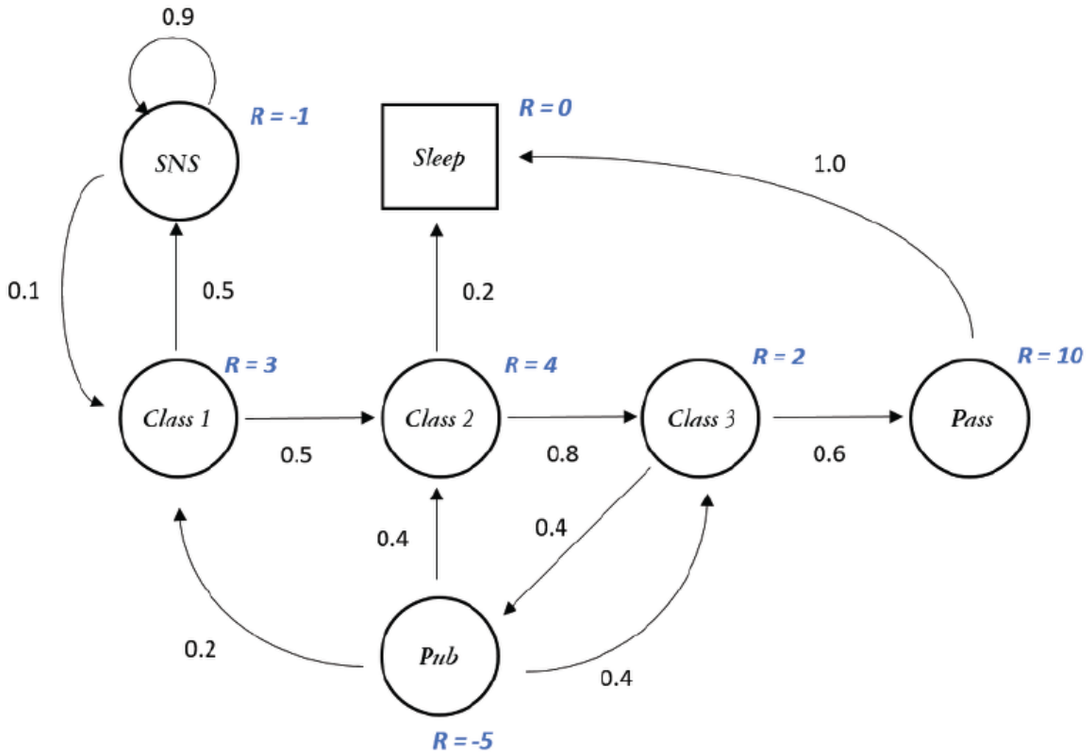


https://www.baeldung.com/cs/ml-policy-reinforcement-learning

- States $s_0 = (1,1)$
- Action a: Up, Down, Left, Right
- Reward r: No fruit (-1), Pear (+5), Apple (+10)
- Policy π_1=down, right, right, π_2=right, right, right, down, down, down

Markov Decision Process

- The process of RL is usually given as **Markov Decision Processes (MDPs)**, which is a mathematical framework used for modeling decision making in situations where the outcomes are partly random and partly under the control of a decision maker.



Markov Reward Process (MRP)

	Class 1	Class 2	Class 3	SNS	Pass	Pub	Sleep
Class 1		0.5		0.5			
Class 2			0.8				0.2
Class 3					0.6	0.4	
SNS	0.1			0.9			
Pass							1
Pub	0.2	0.4	0.4				
Sleep							1

State Transition Probability Matrix

https://untitledtblog.tistory.com/139

- Markov property: $Pr(S_{t+1} = s' | S_0, S_1, \dots, S_{t-1}, S_t) = Pr(S_{t+1} = s' | S_t)$
- State Transition Probability Matrix: $P_{ss'} = Pr(S_{t+1} = s' | S_t = s)$
- Immediate reward: $R_s = E[r_{t+1} | S_t = s]$
- Total reward: Return=$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$
- In MDP, we need a **Value function,** which defines the sum of expected rewards from a given state or given action. Value function is needed for evaluating the policy, and then iteratively updating these value functions until they converge to their optimal values, which derives the optimal policy.

Value function

-Estimation of expected rewards from a given state or given action

State-Value function (V)

*γ=discount factor

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|S_t = s] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|S_t = s\right]$$
$$= \mathbb{E}_{\pi}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots |S_t = s]$$
$$= \mathbb{E}_{\pi}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \cdots |S = s]$$
$$= \mathbb{E}_{\pi}[R_{t+1} + \gamma V(s_{t+1})|S_t = s]$$

-expected return of starting at state s and following policy π. **How good is our current state?**

Action-Value function (Q)

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t|S_t = s, a_t = a] = \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|S_t = s\,, a_t = a\right]$$

-expected return of starting at state s, taking action a, and then following policy π. **How good it is for the agent to take a certain action in a certain state?**

State-value function with Action-value function

$$V^{\pi}(s) = \sum_{a \in A} \pi(a|s) q_{\pi}(s,a)$$

Value when action a is done at state s

Probability of choosing action a at state s

Action-value function with State-value function

$$q^{\pi}(s,a) = R_s^a + \gamma \sum_{s' \in S} Pr_{ss'}^a V_{\pi}(s')$$

Bellman Expectation equation

$$V_{\pi}(s) = \mathbb{E}_{\pi}[R + \gamma V_{\pi}(s')]$$

$$\Rightarrow V_{\pi}(s) = \sum_{a \in A} \pi(a|s) \left(R_s^a + \gamma \sum_{s' \in S} Pr_{ss'}^a V_{\pi}(s') \right)$$

$$q^{\pi}(s,a) = R_s^a + \gamma \sum_{s' \in S} Pr_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_{\pi}(s',a')$$

→We use Bellman Expectation equation for **evaluating a certain policy**

Bellman Optimality equation

$$V_*(s) = \max_{\pi} \mathbb{E}_{\pi}[R + \gamma V_*(s') | s_t = s, a_t = a]$$

$$\Rightarrow V_*(s) = \max_{\pi} \left(R_s^a + \gamma \sum_{s' \in S} Pr_{ss'}^a V_{\pi}(s') \right)$$

$$q^*(s,a) = \max_{\pi} (R_s^a + \gamma \sum_{s' \in S} Pr_{ss'}^a V_{\pi}(s'))$$

→We use Bellman Optimality equation for **choosing the optimal policy π^* which produces the maximum value**

Algorithms for solving MDP

Dynamic Programming (DP)

- Refers to a collection of algorithms that can be used to compute optimal policies **given a perfect model of the environment as a MDP**
- It predicts the value function of current policy (prediction) and optimize the policy (control)

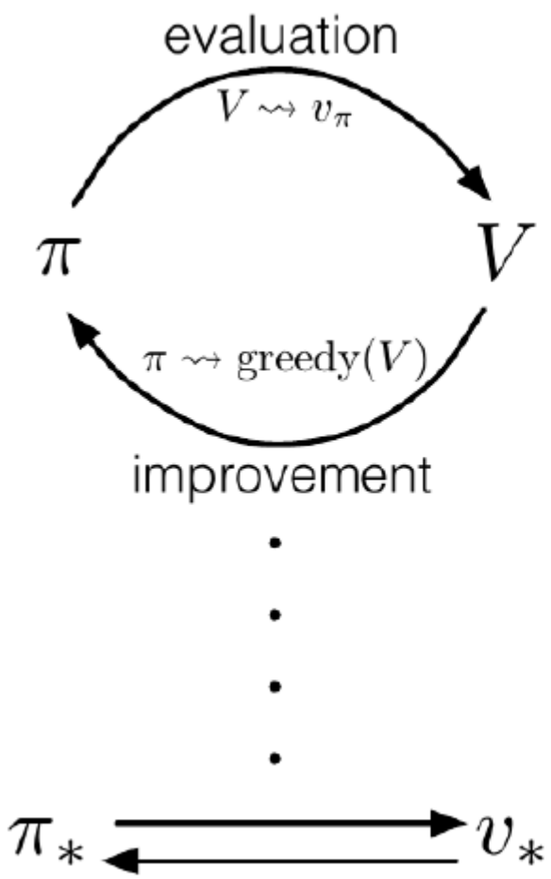
- **Policy Iteration**
- Iterative process of **policy evaluation** and **policy improvement** to find the optimal policy
- Uses Bellman expectation equation

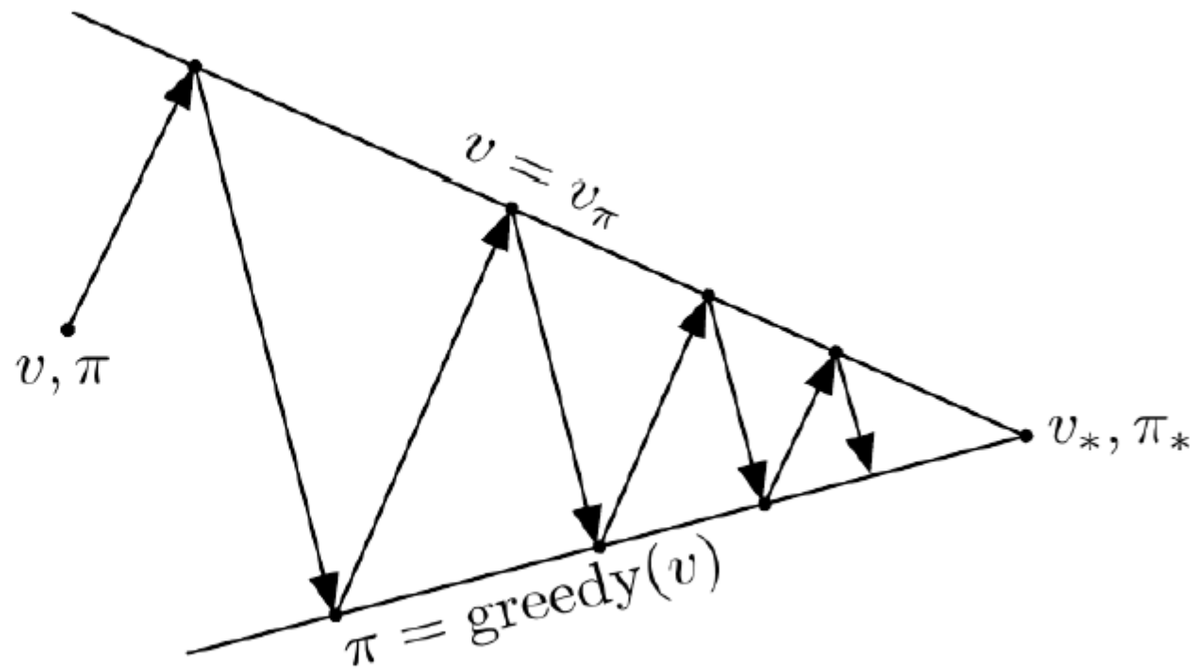
- **Value Iteration**
- Simplification of policy iteration that **combines the policy evaluation and policy improvement steps.**
- The policy at each state is updated by choosing the action that leads to the maximum sum.
- Uses Bellman optimality equation

Policy Iteration

1) Policy Evaluation

-Given an arbitrary policy, we calculate value function V for **all states** under this policy. We iterate through each state and update V until it converges. ➔ We find true value function with iteration

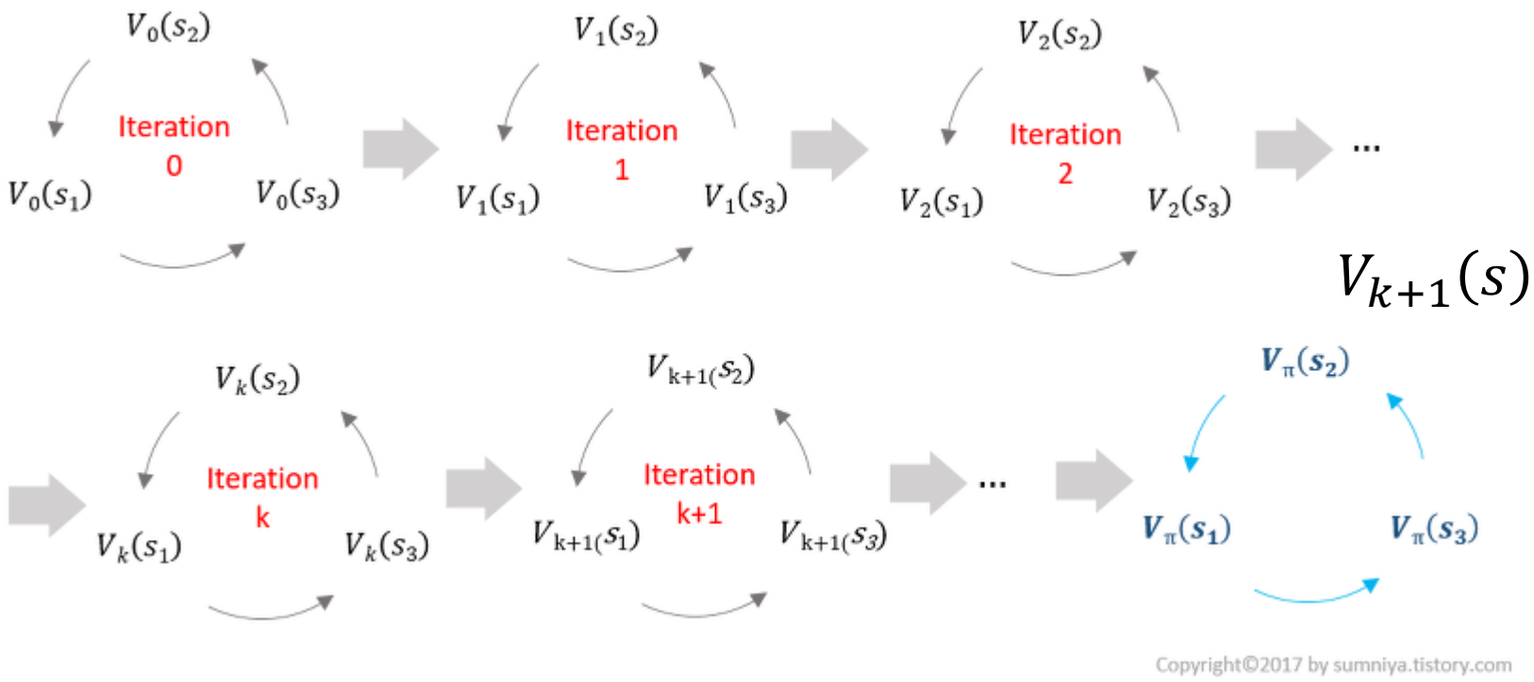




Policy Iteration

1) Policy Evaluation

-Given an arbitrary policy, we calculate value function V for **all states** under this policy. We iterate through each state and update V until it converges. ➔ We find true value function with iteration



$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s \in S} Pr_{ss'}^a V_\pi(s'))$$

2) Policy Improvement

-For each state, we choose the action that gives the highest expected return using V. If this results in an improved policy, we repeat the process with the new policy.

-Greedy policy improvement (pick next state with max value)

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

We repeat the process of Evaluation → Improvement → Evaluation ….

Policy Iteration

$k = 0$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

at (1,2) state

Up $V_1(s) = 0.25 \text{ X } (-1 + 0)$
Down $V_1(s) = 0.25 \text{ X } (-1 + 0)$
Left $V_1(s) = 0.25 \text{ X } (-1 + 0)$
Right $V_1(s) = 0.25 \text{ X } (-1 + 0)$

$\therefore V_1(s) = 4 \text{ X } 0.25 \text{ X } (-1) = \mathbf{-1}$

$k = 1$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,2) state

Up $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Down $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Left $V_2(s) = 0.25 \text{ X } (-1 + 0)$
Right $V_2(s) = 0.25 \text{ X } (-1 + -1)$

$\therefore V_2(s) = 3 \text{ X } 0.25 \text{ X } (-2) + 0.25 \text{ X } (-1) = \mathbf{-1.75}$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,3) state

Up $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Down $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Left $V_2(s) = 0.25 \text{ X } (-1 + -1)$
Right $V_2(s) = 0.25 \text{ X } (-1 + -1)$

$\therefore V_2(s) = 4 \text{ X } 0.25 \text{ X } (-2) = \mathbf{-2}$

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

$k = 0$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

$k = 1$

0	-1.75	-2	-2
-1.75	-2	-2	-1
-1	-2	-2	-1.75
-1	-1	-1.75	0

$k = 2$

0	-2.438	-2.938	-3
-2.438	-2.938	-3	-2.938
-2.938	-3	-2.938	-2.438
-3	-2.938	-2.438	0

$k = 3$

...

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

$k = \infty$

Policy Evaluation

Policy Iteration

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

True value func.

at state 1

Up $q_\pi(1, 0) = -1 + (-14)$

Down $q_\pi(1, 1) = -1 + (-18)$

Left $q_\pi(1, 2) = -1 + (0)$

Right $q_\pi(1, 3) = -1 + (-20)$

$\therefore max\, q_\pi(1, a) = q_\pi(1, \textbf{\textit{Left}})$

Greedy

0	-14.	-20.	-22.
-14.	-18.	-20.	-20.
-20.	-20.	-18.	-14.
-22.	-20.	-14.	0

at state 5

Up $q_\pi(5, 0) = -1 + (-14)$

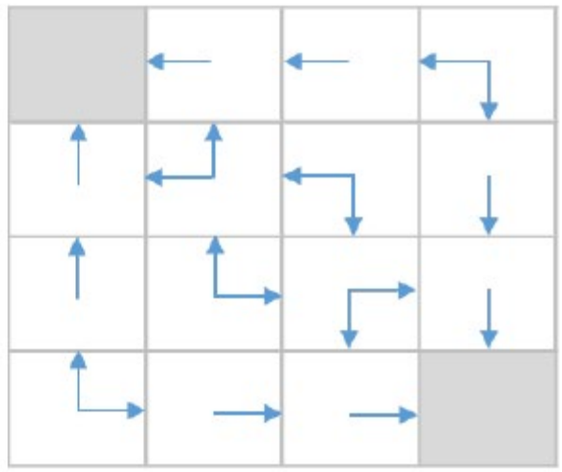
Down $q_\pi(5, 1) = -1 + (-20)$

Left $q_\pi(5, 2) = -1 + (-14)$

Right $q_\pi(5, 3) = -1 + (-20)$

$\therefore max\, q_\pi(5, a) = q_\pi(5, \textbf{\textit{Up}})$ or $q_\pi(5, \textbf{\textit{Left}})$

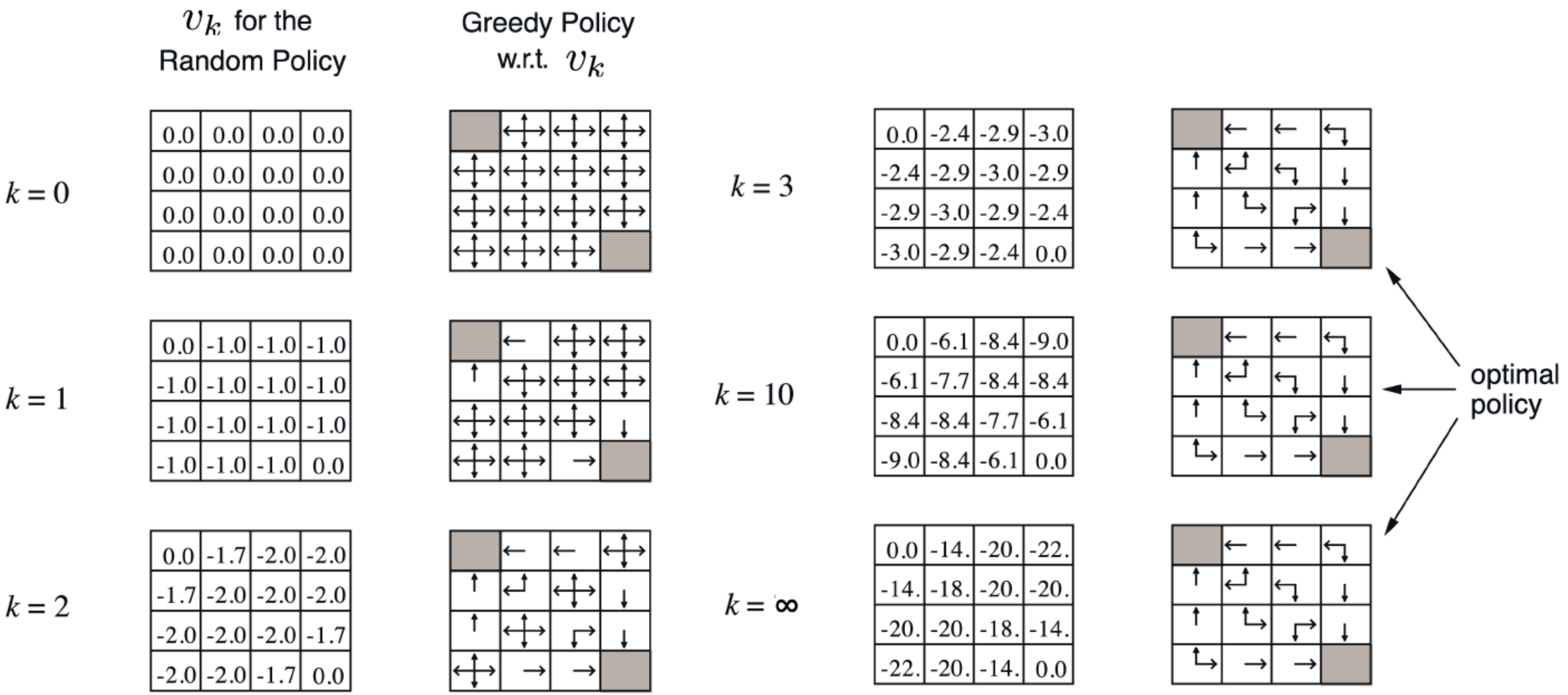
	1	2	3
4	5	6	7
8	9	10	11
12	13	14	



Results

Policy Improvement

Policy Iteration



Value Iteration

- Instead of summing all possible states' value functions, we only take the max value.

$$V_{k+1}(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s \in S} Pr_{ss'}^a V_\pi(s')) \quad \Rightarrow \quad V_{k+1}(s) = \max_{a \in A} \left(R_s^a + \gamma \sum_{s \in S} Pr_{ss'}^a V_\pi(s') \right)$$

Policy Iteration **Value Iteration**

- Thus, policy improvement is included in the iteration itself, because we choose the action with maximum value function.
- Policy evaluation in value iteration (which finds the true value function) implies the improvement to optimal policy.

Value Iteration

k = 0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

at (1,2) state : state 1

Up $V_1(s) = -1 + 0$
Down $V_1(s) = -1 + 0$
Left $V_1(s) = -1 + 0$
Right $V_1(s) = -1 + 0$

$\therefore V_1(1) = \max V_1(s) = \mathbf{-1}$

k = 1

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,2) state : state 1

Up $V_2(s) = -1 + (-1)$
Down $V_2(s) = -1 + (-1)$
Left $V_2(s) = -1 + (0)$
Right $V_2(s) = -1 + (-1)$

$\therefore V_2(s) = \max V_2(s) = \mathbf{-1}$

0	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	-1
-1	-1	-1	0

at (1,3) state : state 2

Up $V_2(s) = -1 + (-1)$
Down $V_2(s) = -1 + (-1)$
Left $V_2(s) = -1 + (-1)$
Right $V_2(s) = -1 + (-1)$

$\therefore V_2(s) = \max V_2(s) = \mathbf{-2}$

0	-1	-2	-2
-1	-2	-2	-2
-2	-2	-2	-1
-2	-2	-1	0

k = 2

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

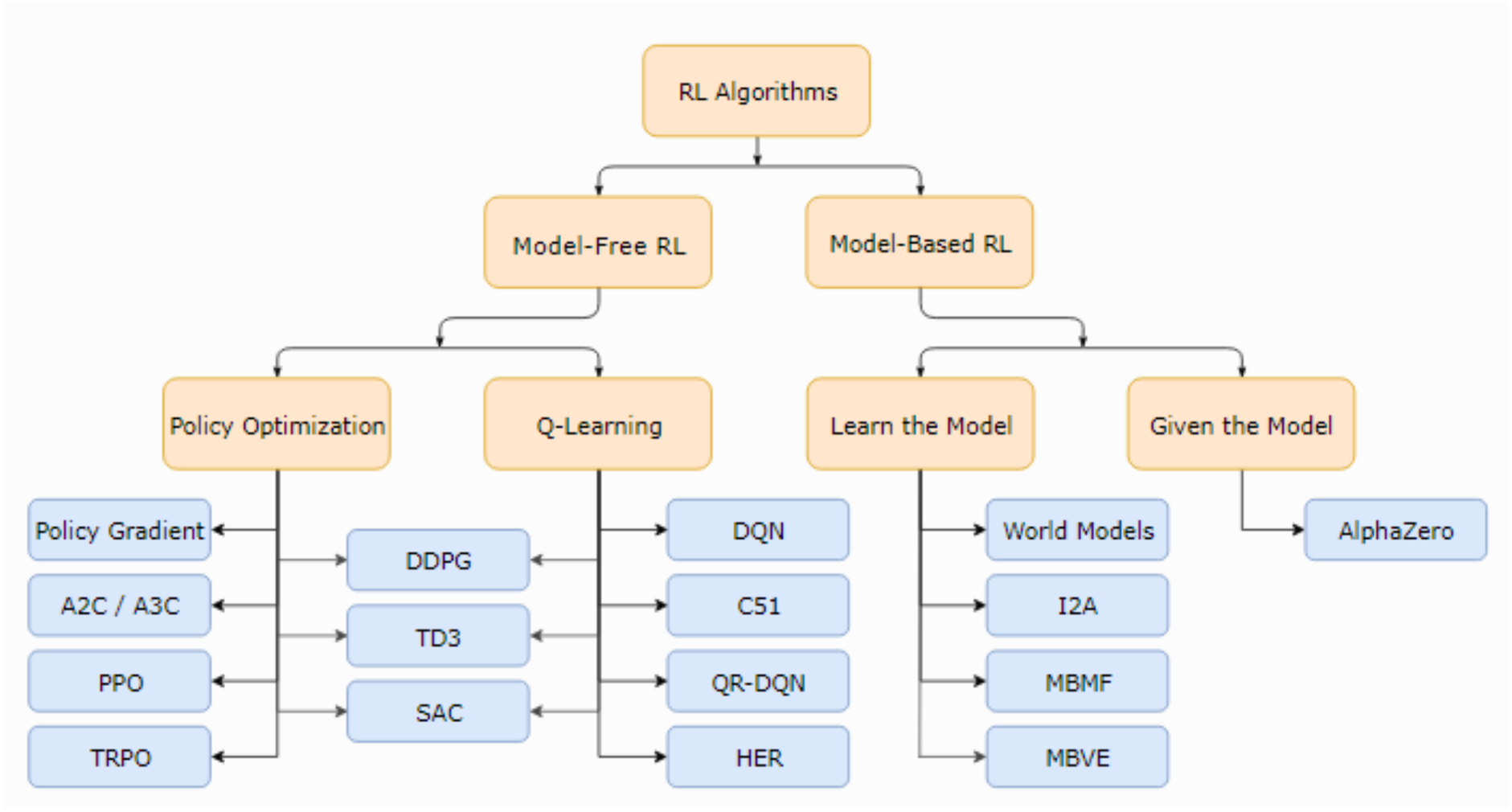
k = 3

...

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

k = ∞

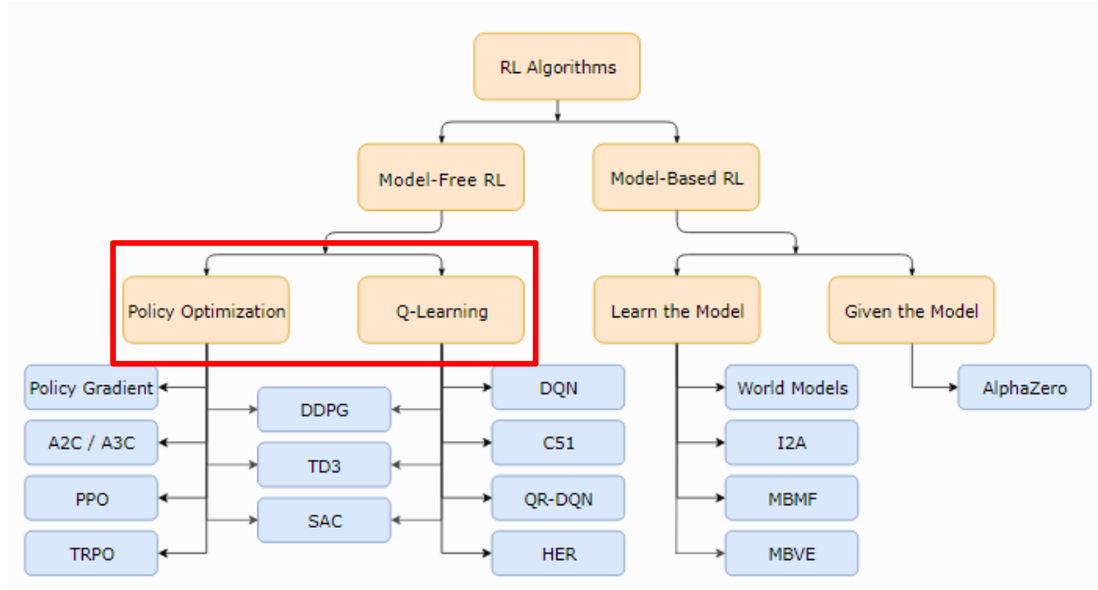
Taxonomy of RL algorithms



Model-free vs Model-based

- Do the agent have access to (or learns) a model of the environment? Can we know the function which predicts state transitions ($P^a_{ss'}$) and rewards (R^a_s)?
- While model-free methods forego the potential gains in sample efficiency from using a model, they tend to be easier to implement and tune.

Taxonomy of RL algorithms



Q-Learning (Value-based RL)

- Learn a Q-value function $Q_\theta(s, a)$ and then find optimal policy
- Pros: More sample-efficient as it bootstraps from its current knowledge.
- Cons: We can find only one optimal policy because we choose the optimal policy based upon the Q function.

Policy Optimization (Policy-based RL)

- Without learning Q-function, we will directly approximate the policy itself, by parameterizing the policy.
- Optimize the parameters θ defining $\pi_\theta(a|s)$ by gradient ascent on the performance objective $J(\pi_\theta)$.
- Pros: It can handle continuous action spaces efficiently, and learn stochastic policies
- Cons: May converge slowly as it has to explore the environment thoroughly to estimate the policy gradient accurately

Interpolating between Policy Optimization and Q-learning

- There are trade-off between the strengths and weaknesses of either methods.
- e.g) SAC: maintain both a policy (the actor) and a value function (the critic), using the value function to critique the policy's action and adjust the policy parameters accordingly.

Multi-Armed Bandit

- Agent faces a choice between several actions, but is uncertain about the rewards each action will yield. Over time, agent updates its estimate of the action's expected reward.
- ε-greedy or UCB strategies can be used to dictate how the agent balances exploration (trying new action) and exploitation (choose action with the highest estimated reward)

Markov Decision Process (MDP)

- Models the environment where each state has associated actions the agent can take.

Temporal Difference Learning (TD Learning)

-The agent updates its value estimates based on the most recent time step, using the difference between the estimated value of the current state and the estimated value of the next state (temporal difference)

Q-Learning

-Model-free-off-policy algorithm. After taking an action in a state and observing the reward and next state, the agent updates the Q-values $Q(s, a)$ of taken action in the current state.

-Q-value represents the expected cumulative discounted future reward the agent will receive starting from state s, taking action a, and then following an optimal policy thereafter.

-Update is done using the maximum Q-value of the next state, which represents the agent's estimate of future rewards

State-Action-Reward-State-Action (SARSA)

-Instead of using maximum Q-value of the next state as in Q-learning, it uses the Q-value of the action that's actually taken next.

Deep Reinforcement Learning (Deep RL)

-Combines neural networks with RL. A neural network approximates the value function or Q-function, and the parameters of the neural network are updated to minimize the difference between the predicted and target values or Q-values.

Inverse Reinforcement Learning (Inverse RL)

-Instead of learning a policy from given rewards, it seeks to understand the reward function by observing an expert's behavior. Once the reward function is estimated, the agent can then use standard RL techniques to determine the optimal policy.