Introduction to Agentic AI

-- Reinforcement Learning Basics

Instructor: Guangjing Wang

guangjingwang@usf.edu

Last Lecture

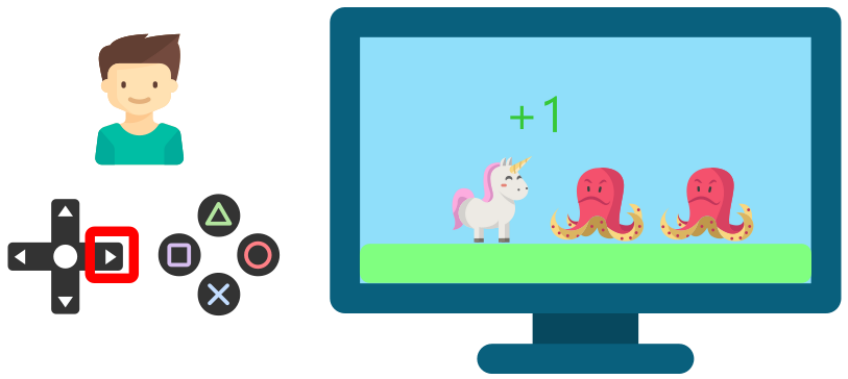
- Short-term and Working Memory
  - Context Window
- Long-term Memory
  - Retrieval Augmented Generation
- Memory Design in Research

This Lecture

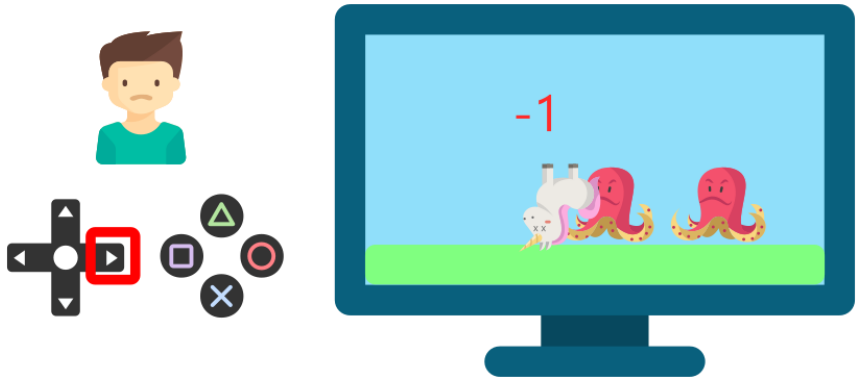
- Reinforcement Learning
- Policy-based and Value-based Methods
- Monte Carlo and Temporal Difference Learning
- Q-learning

Reinforcement Learning

- A computational approach for solving control tasks (also called decision problems) by building agents that learn from the environment by interacting with it through trial and error and **receiving rewards** (positive or negative) as unique feedback.



Got a coin, +1 reward



Captured by an enemy , -1 reward

Goal of Reinforcement Learning

- The agent's goal is to **maximize cumulative reward**
- A reward R_t is a scalar feedback signal
  - Indicating how well agent is doing at step t

- Reinforcement learning is based on the reward hypothesis:
  - **All goals can be described by the maximization of expected cumulative rewards.**

Rewards and the Discounting

The cumulative reward at each time step **t** can be written as:

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \dots$$

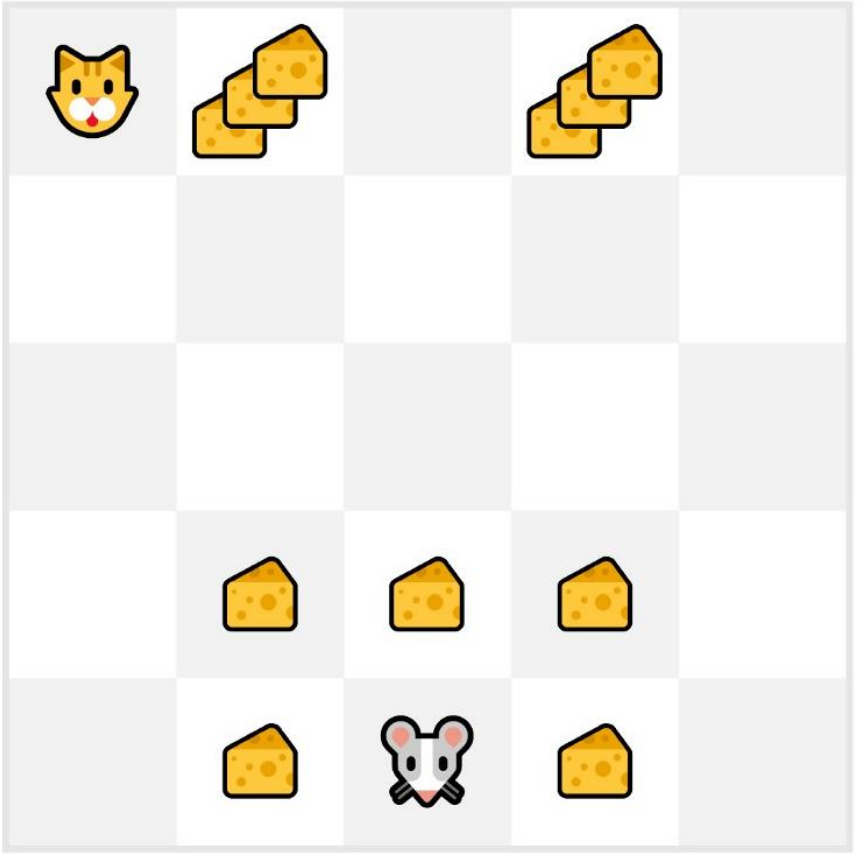
Return: cumulative reward

Trajectory (read Tau)
Sequence of states and actions

However, in reality, **we can't just add them like that.** The rewards that come sooner (at the beginning of the game) **are more likely to happen** since they are more predictable than the long-term future reward.

Rewards and the Discounting (Cont'd)

- The tiny mouse that can move one tile each time step, and your opponent is the cat (that can move too).
- The mouse's goal is **to eat the maximum amount of cheese before being eaten by the cat.**
- The reward near the cat, even if it is bigger (more cheese), will be more discounted since we are not sure we will be able to eat it.



Rewards and the Discounting (Cont'd)

- To discount the rewards, we proceed like this:
- We define a discount rate called gamma. **It must be between 0 and 1.** Most of the time between **0.95 and 0.99**.
  - The larger the gamma, the smaller the discount. This means our agent **cares more about the long-term reward.**
  - On the other hand, the smaller the gamma, the bigger the discount. This means our **agent cares more about the short-term reward (the nearest cheese).**
- Each reward will be discounted by gamma to the exponent of the time step. As the time step increases, the cat gets closer to us, **so the future reward is less and less likely to happen.**

Rewards and the Discounting (Cont'd)

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \dots$$

Return: cumulative reward

Gamma: discount rate

Trajectory (read Tau)
Sequence of states and actions

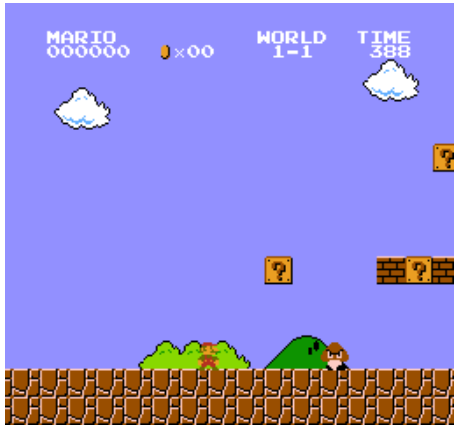
$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

State and Observation

- Observations and States are the **information our agent gets from the environment.**
- *State s*: is **a complete description of the state of the world** (there is no hidden information) in a fully observed environment.
- *Observation o*: is a **partial description of the state** in a partially observed environment.

Have access to the whole check board information

See the part of the level close to the player



RL Loop: State, Action, Reward, and Next State

- Our Agent receives **state** S_0 from the **Environment**
  - E.g., we receive the first frame of our game (Environment).
- Based on that **state** S_0 , the Agent takes **action** A_0
  - E.g., our Agent will move to the right.
- The environment goes to a **new state** S_1
  - E.g., new frame.
- The environment gives some **reward** R_1 to the Agent — we're not dead *(Positive Reward +1)*.

Exploration and Exploitation

- Reinforcement learning is like trial-and-error learning
- The agent should discover a good policy:
  - From its experiences of the environment
  - Without losing too much reward along the way
- **Exploration** exploring the environment by trying random action to find more information about the environment
  - Try restaurants you never went to before, with the risk of having a bad experience **but the probable opportunity of a fantastic experience.**
- **Exploitation** exploits known information to maximize reward
  - You go to the same one that you know is good every day and **take the risk to miss another better restaurant.**
- It is usually important to explore as well as exploit

The Type of Tasks

- Episodic Task
  - We have a starting point and an ending point **(a terminal state).**
  - **This creates an episode**: a list of States, Actions, Rewards, and new States.
  - E.g., Super Mario Bros: an episode begins at the launch of a new Mario Level and ends **when you are killed, or you reached the end of the level.**
- Continuing Task
  - These are tasks that continue forever (**no terminal state**)
  - The agent must **learn how to choose the best actions and simultaneously interact with the environment.**
  - E.g., an agent that does automated stock trading until we manually stop it.

RL for Sequential Decision-Making Problem

- Goal: select actions to maximize total future reward
- Actions may have long term consequences
- Reward may be delayed
- It may be better to sacrifice immediate reward to gain more long-term reward
- Examples:
  - A financial investment (may take months to mature)
  - Re-fueling a helicopter (might prevent a crash in several hours)
  - Blocking opponent moves (might help winning chances many moves from now)

Two Problem Categories where RL is Helpful

- No examples of desired behavior: e.g. because the goal is to go beyond human performance or there is no existing data for a task.

- Enormous search or optimization problem with delayed outcomes.

- The RL process is called a **Markov Decision Process**
  - The agent needs **only the current state to decide** what action to take and **not the history of all the states and actions** they took before.

Approach for Solving RL Problems

- The Policy **π** is the **brain of an Agent**, it is the function that tells us what **action to take given the state we are in.**
- The Policy **is the function we want to learn**, our goal is to find the optimal policy π*, the policy that **maximizes expected return** when the agent acts according to it. We find this π* **through training.**
  - Directly**,** by teaching the agent to learn which **action to take,** given the current state: **Policy-Based Methods.**
  - Indirectly, **teach the agent to learn *which state is more valuable*** and then take the action that **leads to the more valuable states**: **Value-Based Methods.**

Policy-based Method

- A mapping from each state to the best corresponding action
  - *Deterministic*: a policy **will output one action** at a given state **.**

$$a = \pi(s)$$

  - *Stochastic*: outputs **a probability distribution over actions.**

$$\pi(a|s) = P[A|s]$$

Probability Distribution over the set of actions given the state

Value-based Method

- **A value function** that maps a state to the expected value **of being at that state.**
- The **value of a state** is the **expected discounted return** the agent can get if it **starts in that state and then acts according to our policy.**

$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s\right]$$

Value function

Expected discounted return

Starting at state s

State-Value Function

$$V_\pi(s) = \mathbf{E}_\pi[G_t | S_t = s]$$

Value of state s

Expected return

If the agent starts at state s

And uses the policy to choose its actions for all time steps

For each state,
the state-value function outputs
the expected return
if the agent starts in that state
and then follows the policy forever after.

For the state-value function, we calculate the value of a state S_t

Action-Value Function

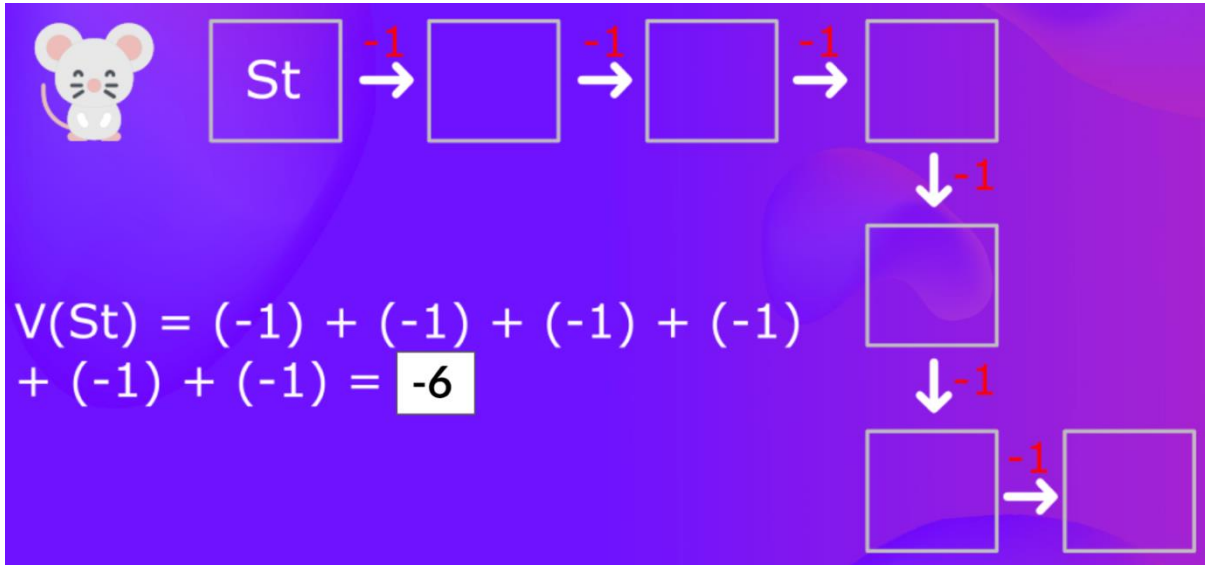
$$Q_\pi(s,a) = \mathbf{E}_\pi[G_t | S_t = s, A_t = a]$$

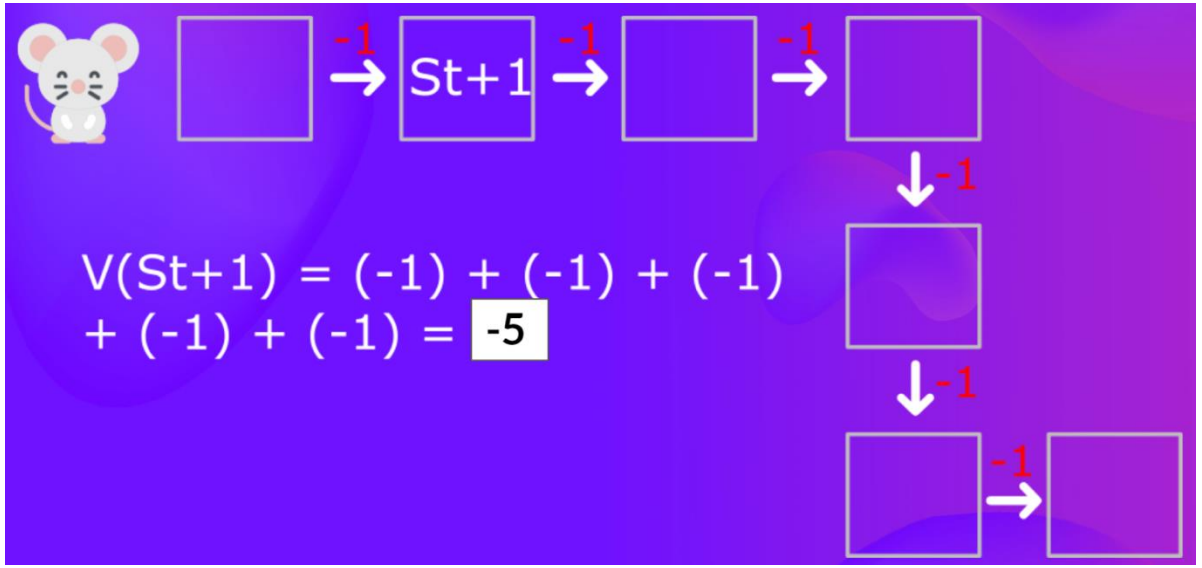
- $Q_\pi(s,a)$: Value of state-action pair s,a
- $\mathbf{E}_\pi$: Expected return, and then uses the policy to choose its actions for all time steps
- $S_t = s$: If the agent starts at state s
- $A_t = a$: and chooses action a

For each state and action,
the action-value function outputs
the expected return
if the agent starts in that state
and takes the action
and then follows the
policy forever after.

For the action-value function, we calculate **the value of the state-action pair (S_t, A_t)** hence the value of taking that action at that state.

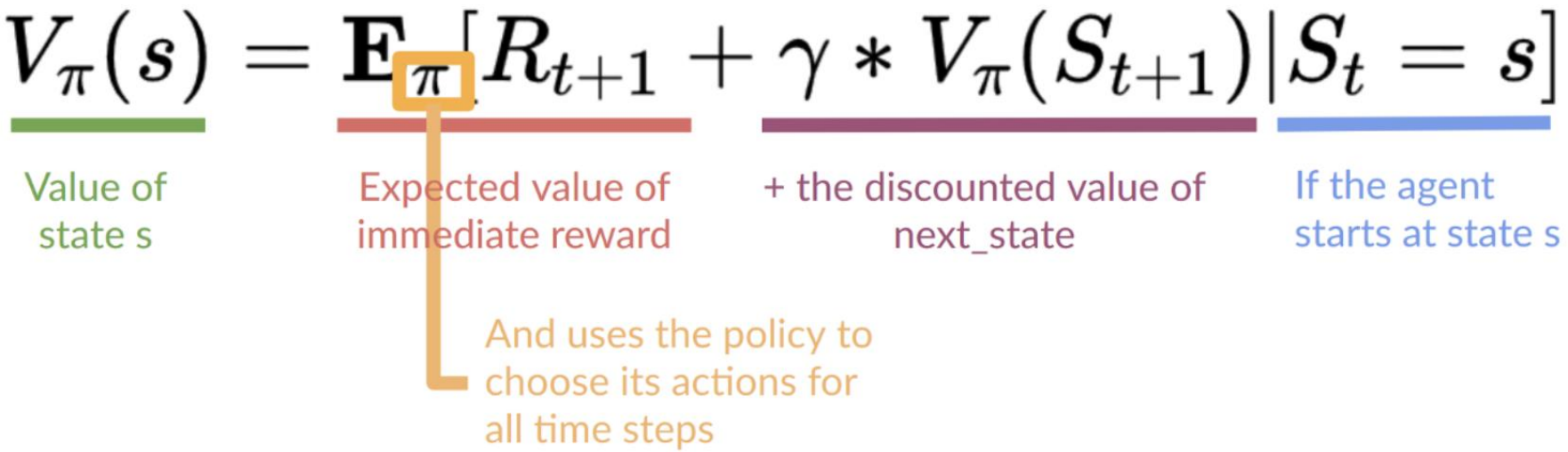
The Sum of Expected Reward





To calculate the $V(S_{t+1})$,we need to calculate the return starting at that state S_{t+1}.

The Bellman Equation



The idea of the Bellman equation is that instead of calculating each value as the sum of the expected return, **which is a long process**, we calculate the value as **the sum of immediate reward + the discounted value of the state that follows.**

Policy-based VS Value-based

- *Policy-based methods:* **Directly train the policy** to select what action to take given a state (or a probability distribution over actions at that state). In this case, we **don't have a value function.**
  - **We do not define by hand the behavior of our policy;**
  - **The optimal policy (denoted π*) is found by training the policy directly.**
- *Value-based methods:* **Indirectly, by training a value function** that outputs the value of a state or a state-action pair.
  - Since the policy is not trained, **we need to specify its function. Finding an optimal value function (Q* or V*) leads to having an optimal policy.**

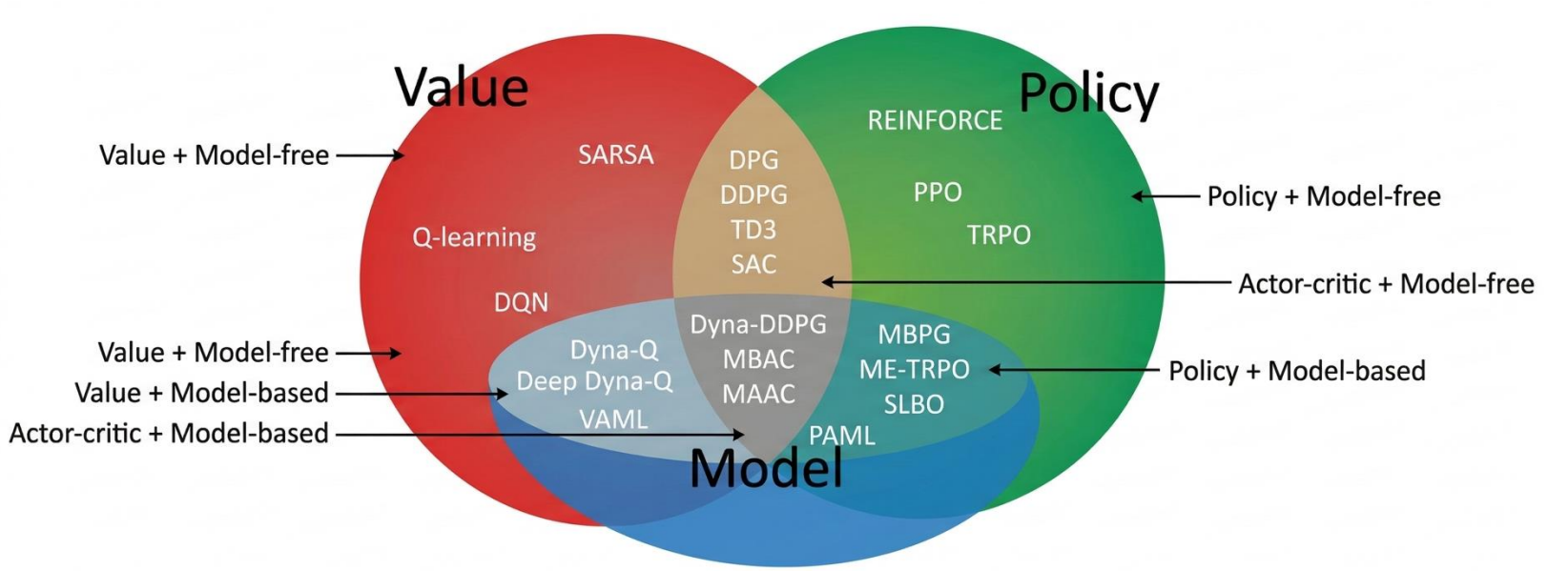
$$\pi^*(s) = \arg\max_a Q^*(s, a)$$

Pi’s Coffee Making Robot

https://website.pi-asset.com/pi06star/PiFinal.mp4

Major Components of a RL Agent

- An RL agent may include one or more of these components:
  - **Policy**: agent's behavior function (**action**)
    - Policy is mapping from past experience (states) to action
  - **Value function**: how good is each state and/or action
  - **Model**: representation of the environment (**observation**)



Reinforcement Learning and Planning

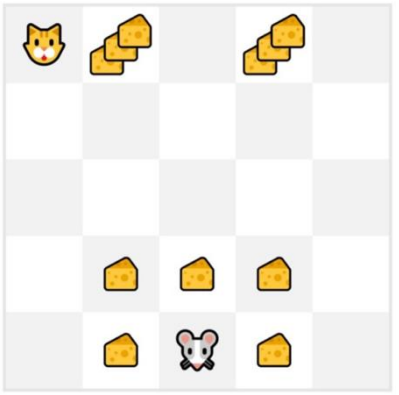
- Planning:
  - A model of the environment is known
  - The agent performs computations with its model (without any external interaction)
  - The agent improves its policy a.k.a. deliberation, reasoning, introspection, thought, search

- Reinforcement Learning:
  - The environment is initially unknown
  - The agent interacts with the environment
  - The agent improves its policy

Monte Carlo and Temporal Difference Learning

- Idea: given the experience and the received reward, the agent will update its value function or policy.
- Two different **strategies on how to train the value function or the policy function:**
  - Monte Carlo: uses **an entire episode of experience before learning**
  - Temporal Difference Learning: uses **only a step ($S_t, A_t, R_{t+1}, S_{t+1}$) to learn**

Monte Carlo



- We always start the episode **at the same starting point.**
- **The agent takes actions using the policy**.
  - For instance, a policy that alternates between exploration (random actions) and exploitation.
- We get **the reward and the next state.**
- Suppose we terminate the episode if the cat eats the mouse or if the moves > 10 steps.
- At the end of the episode, **we have a list of State, Actions, Rewards, and Next States tuples.**
  - For instance [[State tile 3 bottom, Go Left, +1, State tile 2 bottom], [State tile 2 bottom, Go Left, +0, State tile 1 bottom]…]

Monte Carlo (Cont'd)

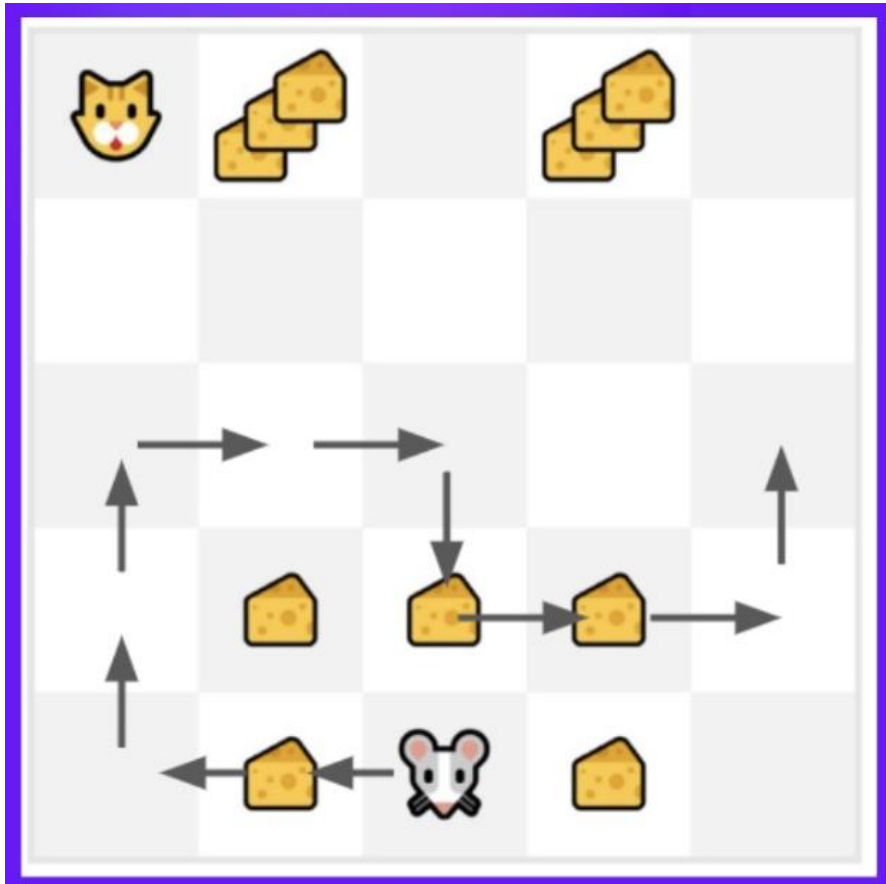
- **The agent will sum the total rewards G_t** to see how well it did.
- It will then **update $V(s_t)$ based on the formula**

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

- $V(S_t)$: New value of state t
- $V(S_t)$: Former estimation of value of state t (= Expected return starting at that state)
- α: Learning Rate
- G_t: Return at timestep t
- $V(S_t)$: Former estimation of value of state t (= Expected return starting at that state)

- Then start a new game with this new knowledge.
- By running more and more episodes, the agent will learn to play better and better.

Monte Carlo (Cont'd)

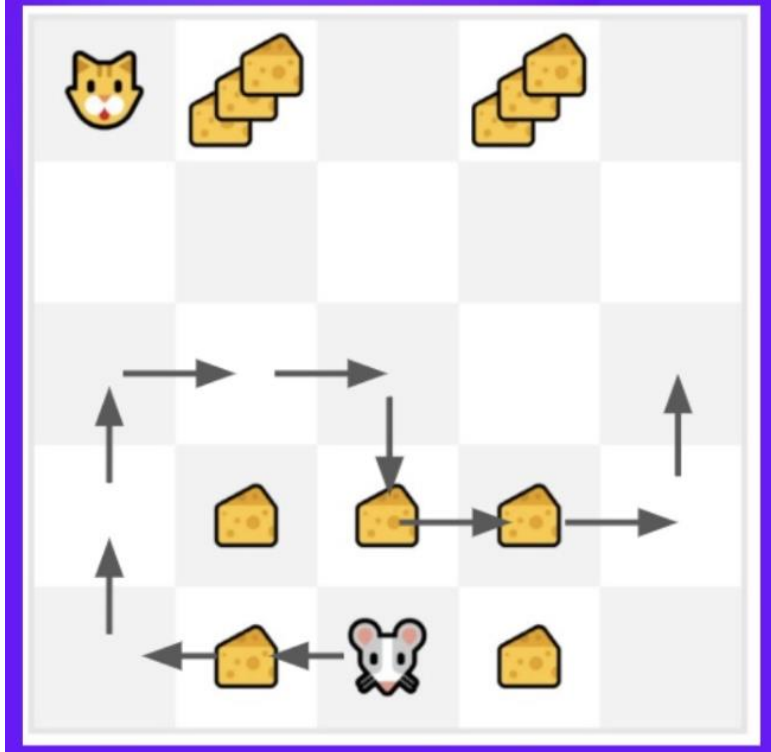


- We just started to train our Value function **so it returns 0 value for each state.**
- Learning rate (lr) is 0.1 and our discount rate is 1 **(no discount)**
- Our mouse, **explore the environment** and take random actions
- The mouse **made more than 10 steps**, so the episode ends.

Suppose mouse will stop if step_number > 10

Monte Carlo (Cont'd)

Monte Carlo Approach:



- **Calculate the return Gt.**

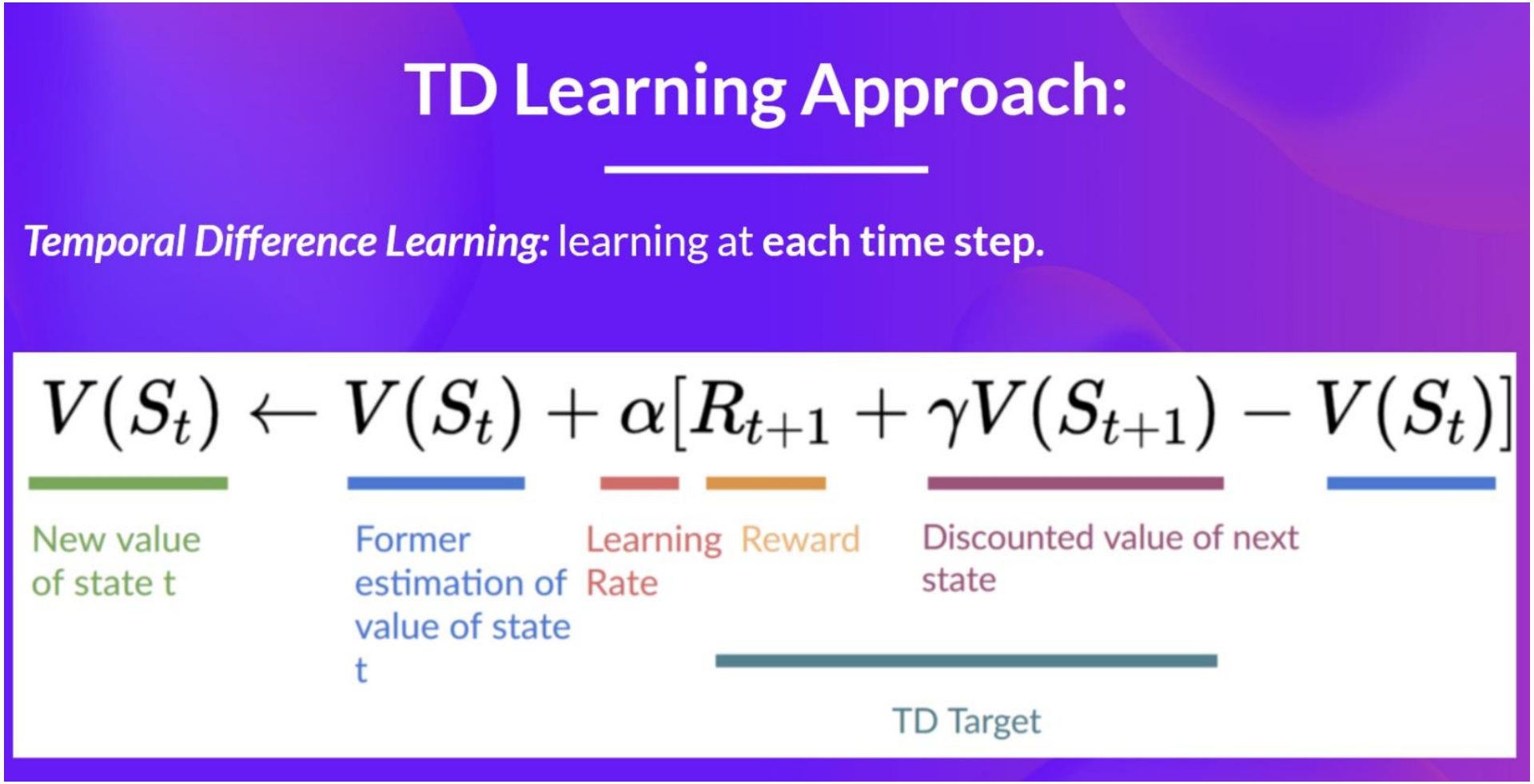
Gt = Rt+1 + Rt+2 + Rt+3…
Gt = 1 + 0 + 0 + 0+ 0 + 0 + 1 + 1+ 0 + 0
Gt= 3

- **We can now update V(S0).**

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)]$$

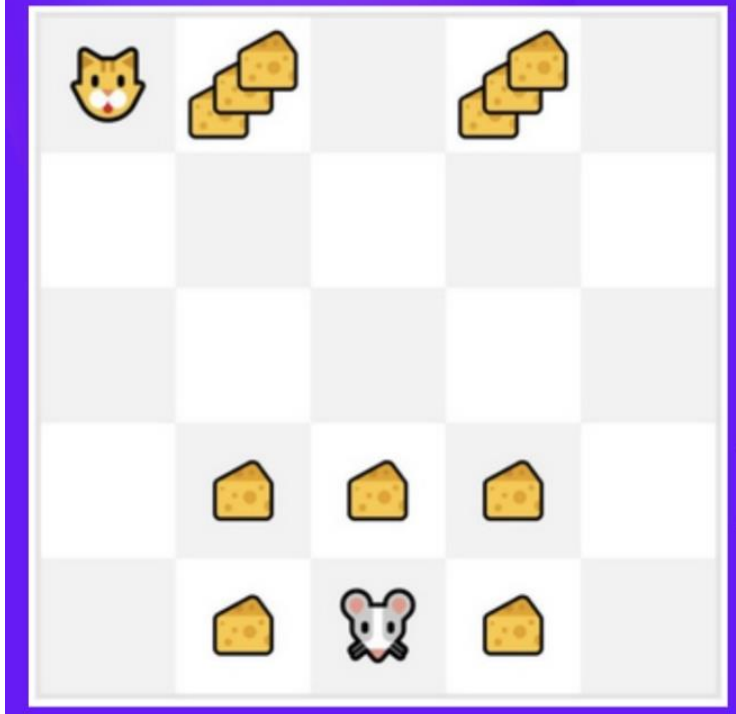
New V(S0) = V(S0) + lr * [Gt-V(S0)]
New V(S0) = 0 + 0.1 * [3 –0]
New V(S0) = 0.3

Temporal Difference Learning



Temporal Diffenence Learning (Cont'd)

TD Approach:



At the end of one step (State, Action, Reward, Next State):

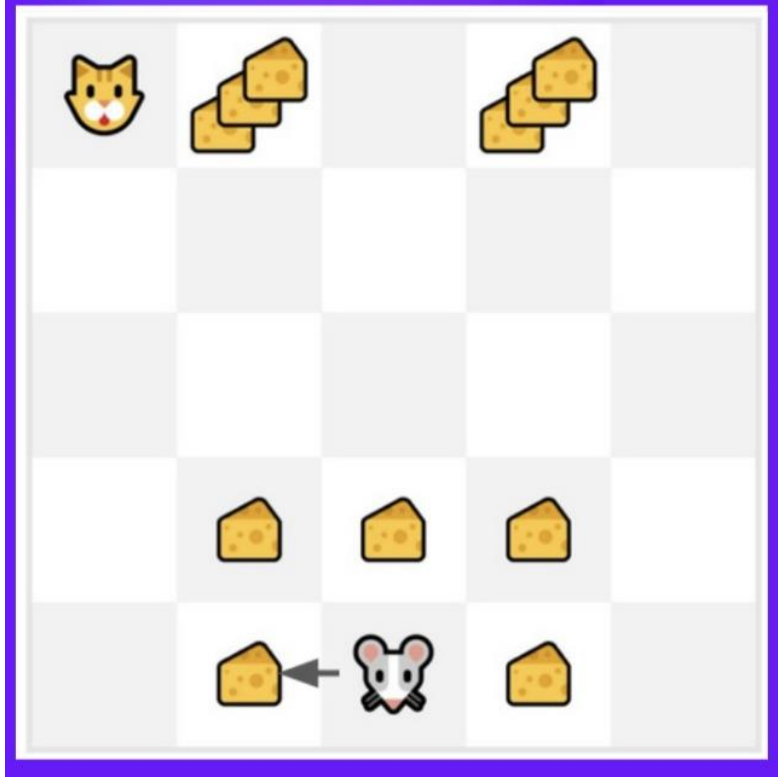
- We have Rt+1 and St+1
- We update V(St):
  - **We estimate Gt** by adding Rt+1 and the discounted value of next state.
    TD target : Rt+1 + gamma * V(St+1)

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

Now we **continue to interact with this environment** with our updated value function.
By running more and more steps, **the agent will learn to play better and better.**

Temporal Diffenence Learning (Cont'd)

TD Approach:



- We can now update V(S0):

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)]$$

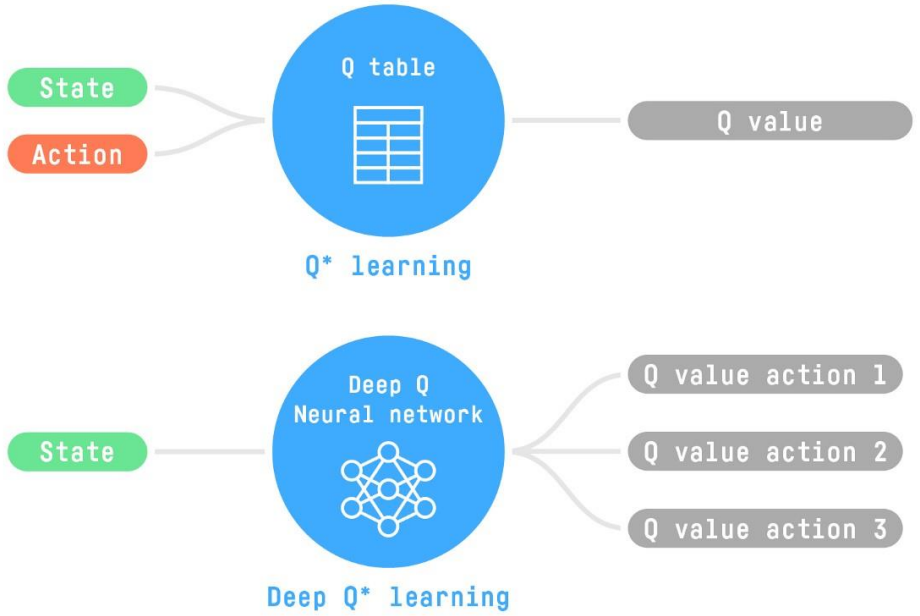
New V(S0) = 0 + 0.1 * [1 + 1 * 0–0]
The new V(S0) = 0.1

So we just updated our **value function for State 0.**

Now **we continue to interact with this environment with our updated value function.**

Q-Learning

- **Q-Learning (classic Reinforcement Learning)**
  - We use a traditional algorithm to create a Q table that helps us find what action to take for each state.
- **Deep Q-Learning**
  - we will use a Neural Network (to approximate the Q value).



Q-Learning

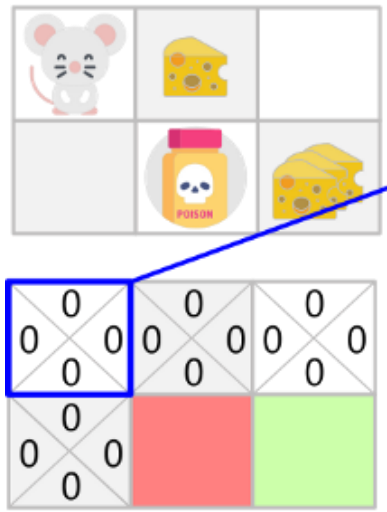
- Q-Learning is the algorithm we use to train our Q-function
- Q-function is an action-value function that determines the value of being at a particular state and taking a specific action at that state.

$$Q_\pi(s, a) = \mathbf{E}_\pi[G_t | S_t = s, A_t = a]$$

- The ***value of a state*, or a *state-action pair*** is the expected cumulative reward our agent gets if it starts at this state (or state-action pair) and then acts accordingly to its policy.
- The **reward** is the feedback it gets from the environment after performing an action at a state.

Q-table

- Q-function is encoded by **a Q-table, a table where each cell corresponds to a state-action pair value;**
- Think of this Q-table as **the memory or cheat sheet of our Q-function.**



Actions

States

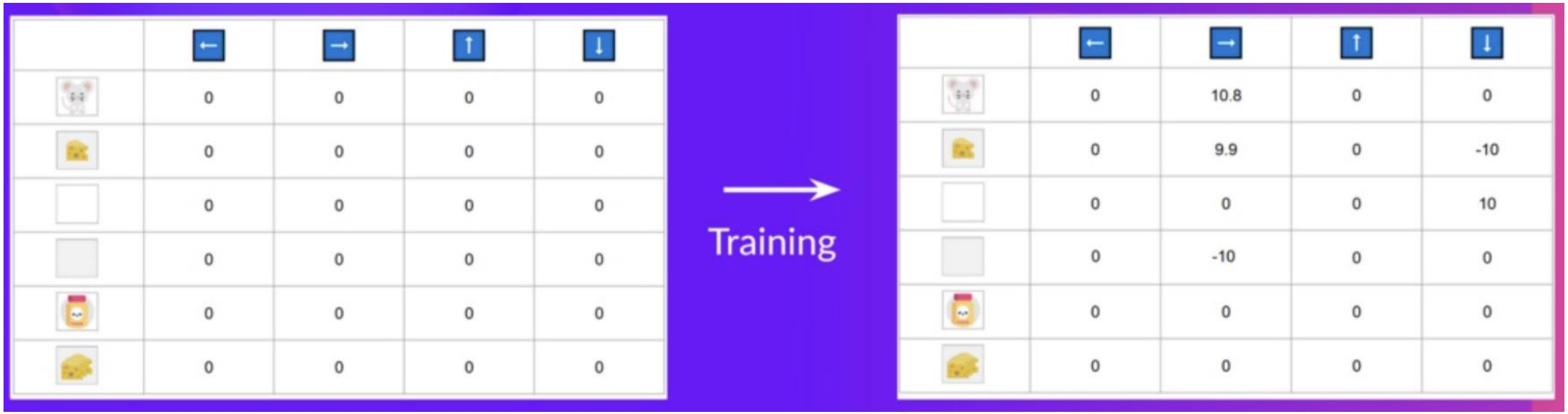
	←	→	↑	↓
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0
	0	0	0	0

Q-Learning Algorithm

- Trains a *Q-function* (an **action-value function**), which internally is a **Q-table that contains all the state-action pair values.**
- Given a state and action, our Q-function **will search its Q-table for the corresponding value.**
- When the training is done, **we have an optimal Q-function, which means we have optimal Q-table.**
- And if we **have an optimal Q-function**, we **have an optimal policy** since we **know the best action to take at each state.**

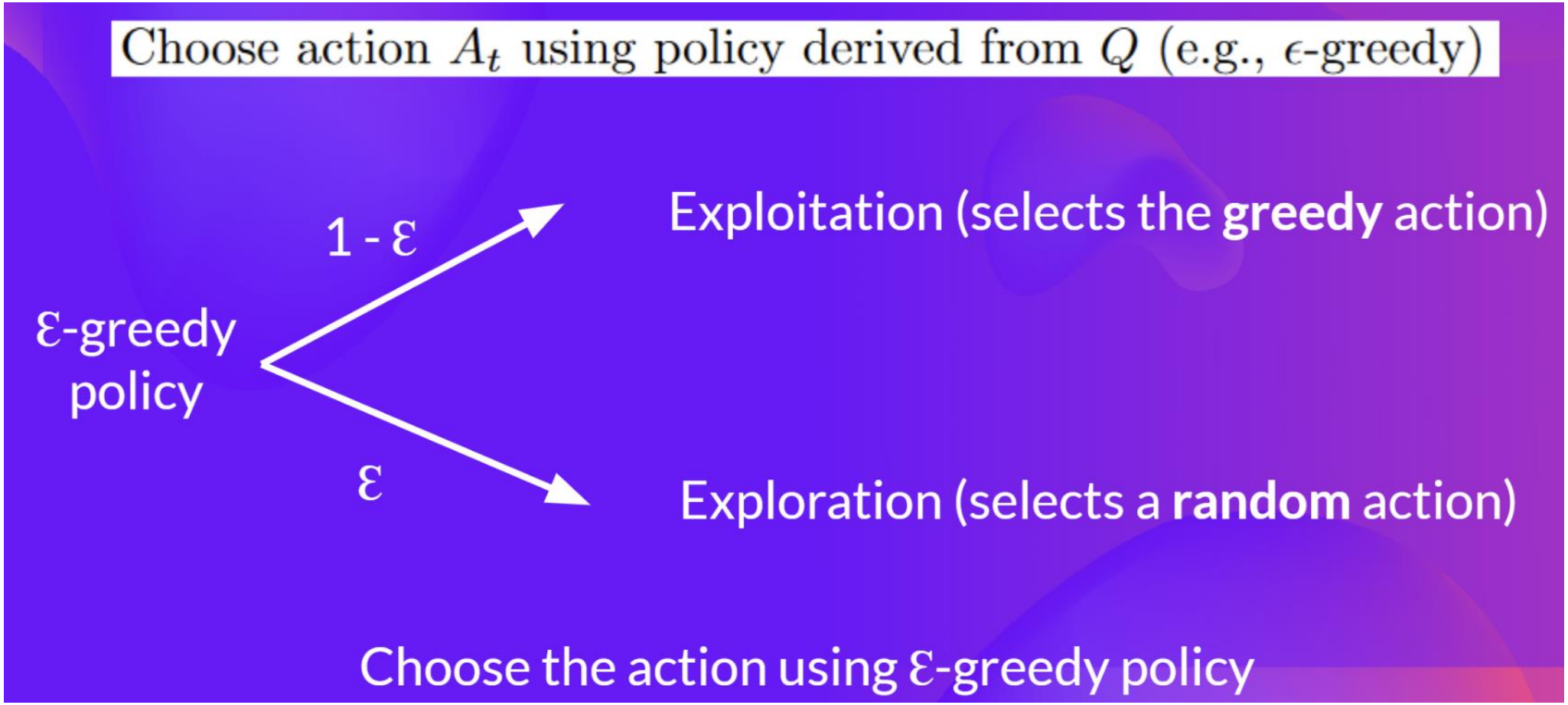
Q-Learning Algorithm (Cont'd): Initialization

- In the beginning, **our Q-table is useless since it gives arbitrary values for each state-action pair** (most of the time, we initialize the Q-table to 0).
- As the agent **explores the environment and we update the Q-table, it will give us a better and better approximation** to the optimal policy.



Q-Learning Algorithm (Cont'd): Choose Action

After Q-table initialization, choose an action:



Q-Learning Algorithm (Cont'd)

- **Perform action At, get reward Rt+1 and next state St+1**
- **Update Q(St, At)**

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

New Q-value estimation

Former Q-value estimation

Learning Rate

Immediate Reward

Discounted Estimate optimal Q-value of next state

Former Q-value estimation

TD Target

TD Error

- To get the **best state-action pair value** for the next state, we use a greedy policy to select the next best action.
- Note that this is not an epsilon-greedy policy, this will always take the action with the highest state-action value.

References

- https://huggingface.co/learn/deep-rl-course/unit0/introduction