KISTI 한국과학기술정보연구원
Korea Institute of Science and Technology Information

2022

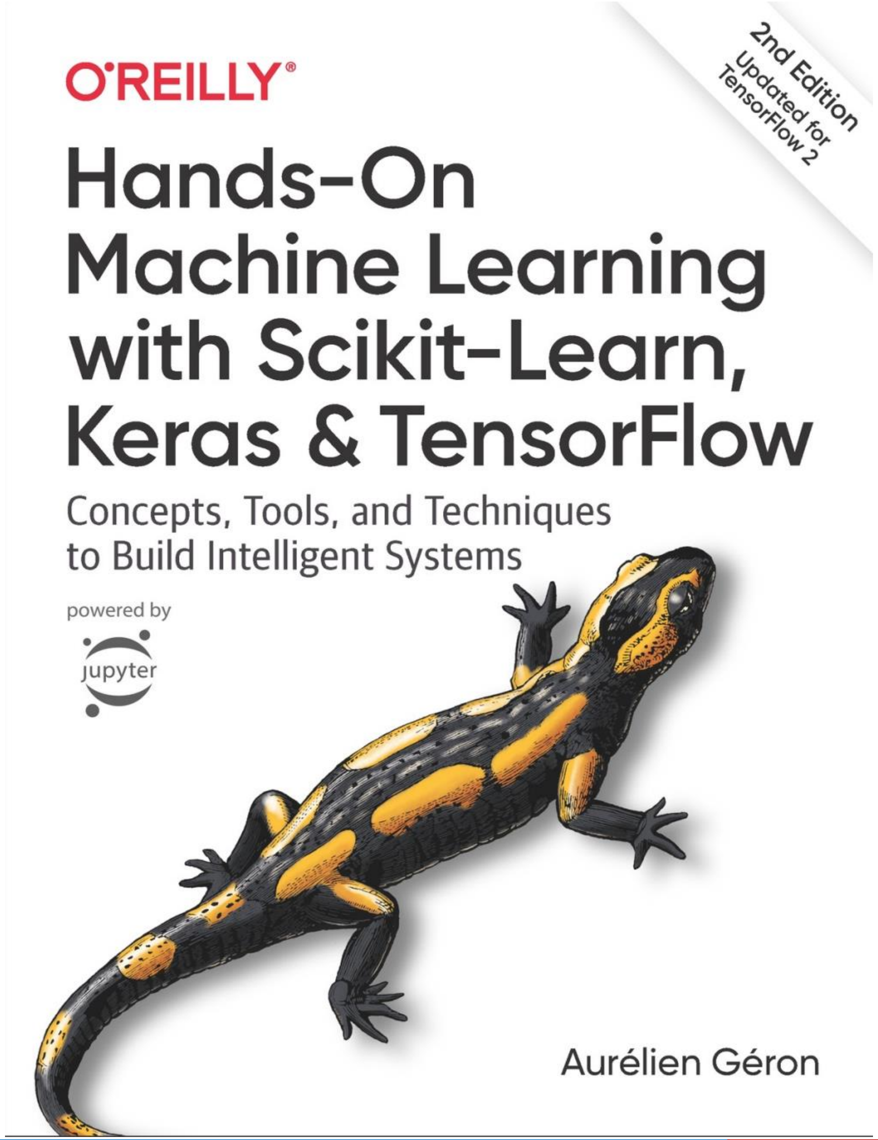
# Advanced Topic in Research Data-centric Deep Learning

## Lec 14: Introduction to Deep Reinforcement Learning

hsyi@kisti.re.kr
Hongsuk Yi (이홍석)

## ❖ Background material

- ✓ Reinforcement Learning: An Introduction, Sutton & Barto 2018
  - http://incompleteideas.net/book/the-book-2nd.html
- ✓ Reinforcement Learning Lecture Series 2021
  - https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021
- ✓ https://huggingface.co/blog/deep-rl-intro



# What is reinforcement learning?

# State Values and Policy Evaluation

# Introduction

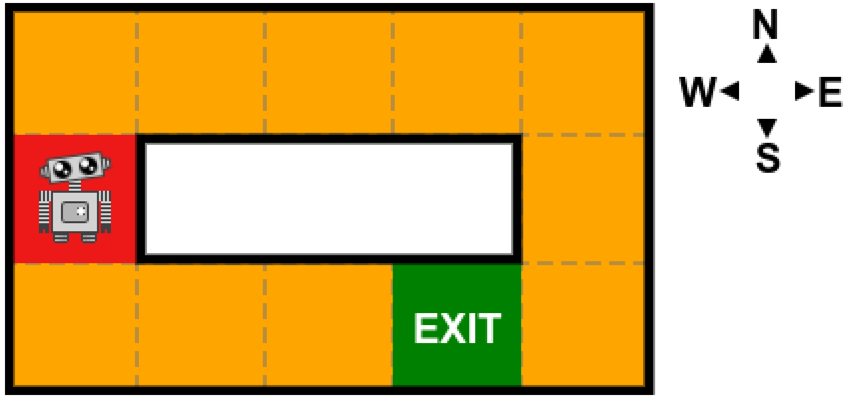
- Reinforcement Learning can be thought of as learning from trial and error
  - ✓ An *agent*, that interacts with its *environment*, receives *rewards* that reflect its ability to accomplish some predefined goal.
- Reinforcement Learning can progressively move towards an agent that gives the maximum amount of reward and that solves the task at hand.
- RL consist of two distinct parts:
  - ✓ The *Prediction Problem*, in which the performance of the agent is evaluated.
  - ✓ The *Control Problem*, where the policy, used by the agent to select its actions, is modified to improve performance

❖ Once upon a time there was a Baby Robot who got lost in the mall

✓ reward: a single numerical value that is used to measure how well the task at hand has been performed

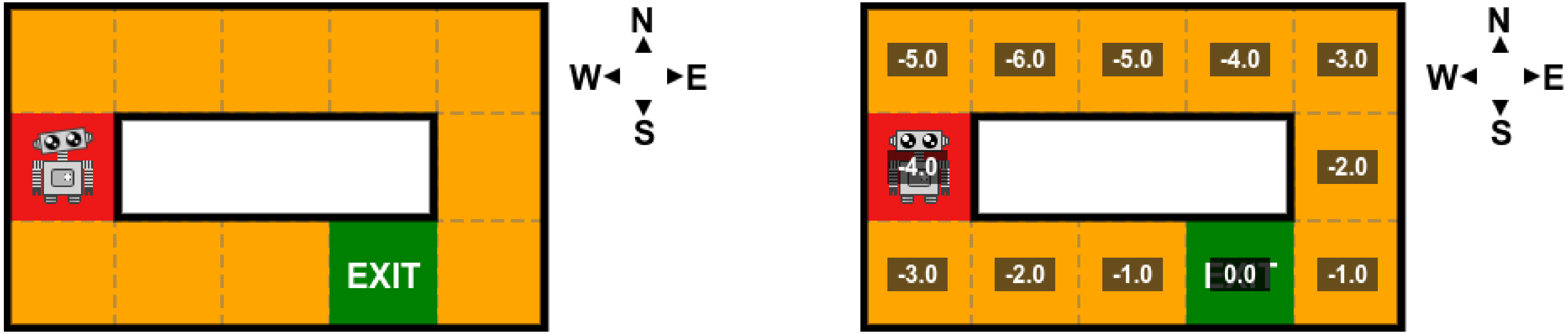
✓ state: each of these squares environment that we're working in.

- When a state is independent of the prior states it is said to satisfy the *Markov Property*.



# The Terminology of Reinforcement Learning (RL)

- Value: how good it is to be in a particular state
- Return: The expected total amount of reward
- Policy: The strategy used to select the next action
  - ✓ optimal : move in the direction of increasing value we are actually following best policy



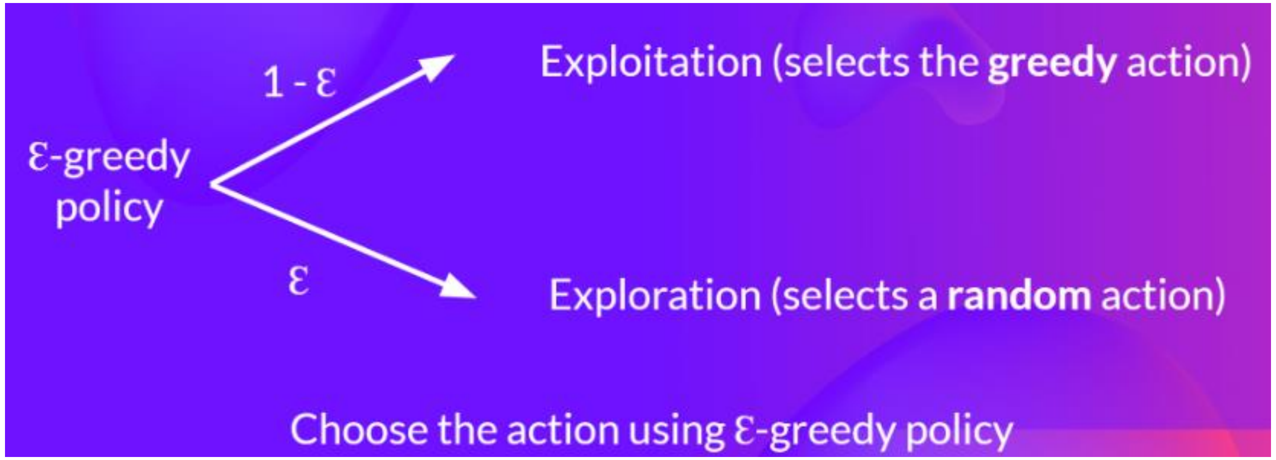
## ❖ The ε-greedy policy

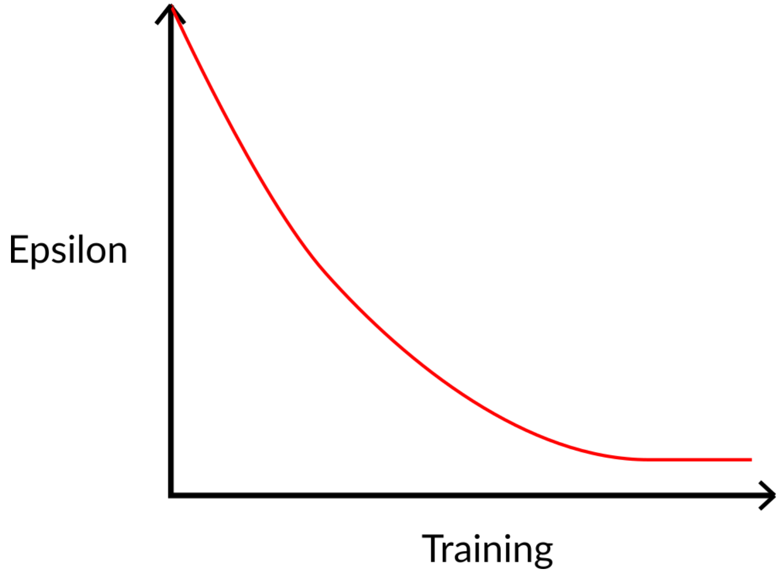
✓ with probability 1 — ε

- **we do exploitation :** our agent selects the action with the highest state-action pair value

✓ With probability ε:

- **we do exploration** : trying random action and then gradually reduce it



# Basic Mathematics

- *r = reward*
- *a = action*
- *s = state, s′ = next state*
- The rewards, states and actions are actually random variables:
  - ✓ there's a probability of getting a certain reward, taking a specific action/state
    - these probabilities are referred to using capital letters.
- the expected reward for a state-action pair:

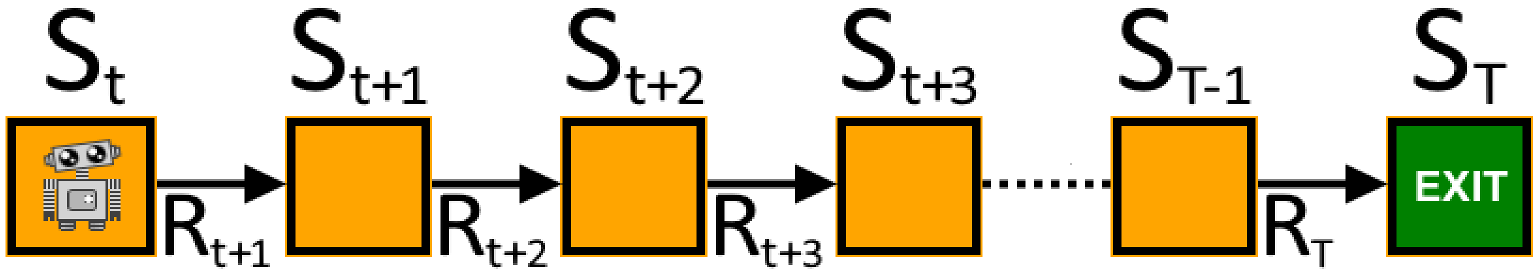
$$r(s, a) = \mathbb{E}[R_t \mid S_{t-1} = s, A_{t-1} = a]$$

# Basic Mathematics: Return G

- Return '$G_t$': the total amount of reward accumulated over an episode, starting at time 't'.
  - an *episode* refers to all the time steps that occur between entering and exiting a level.

✓ the return is just the sum of the future rewards

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$



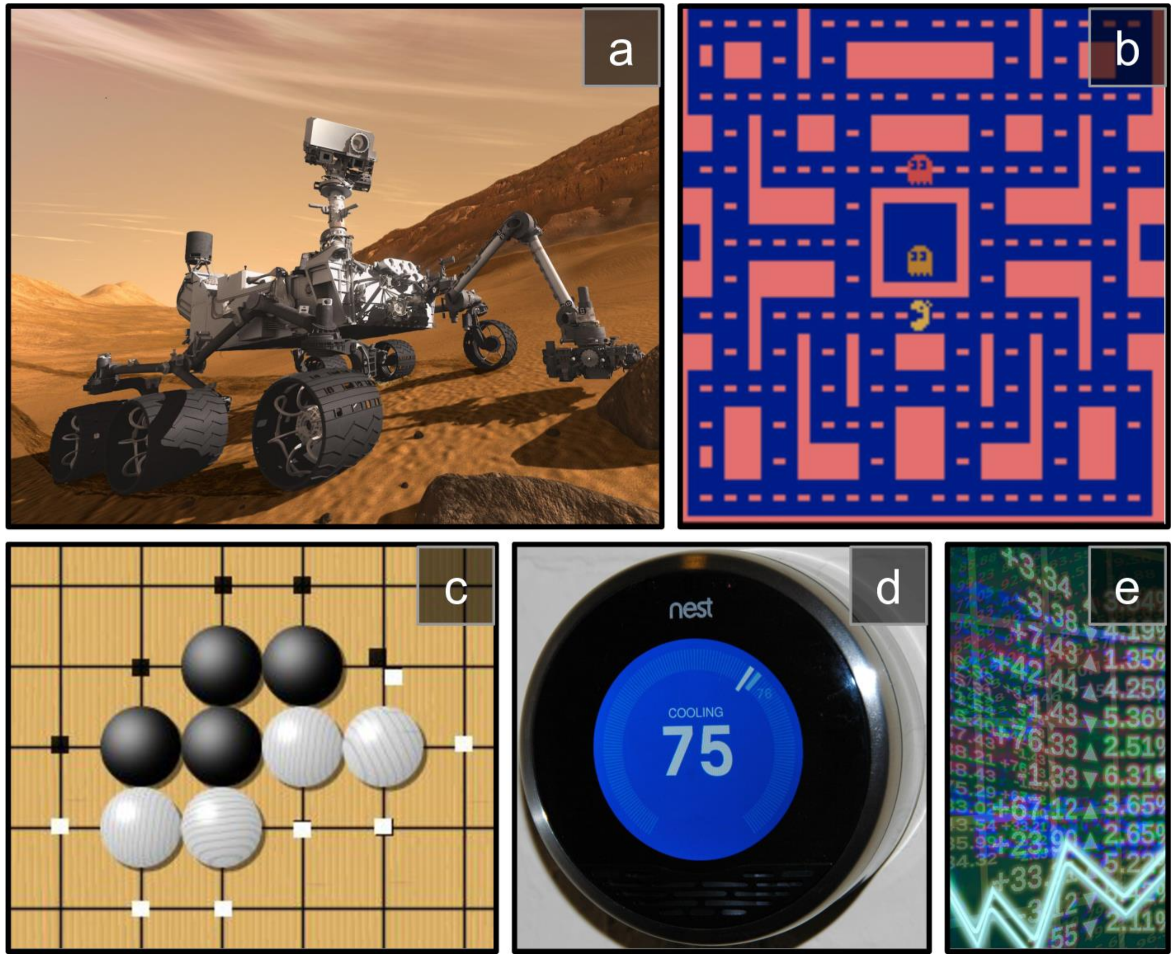
KISTI SINCE 1962

- Policy: software agent uses to determine its actions which is commonly denoted by the symbol $\pi$
  - ✓ So the value for state $s$ under policy π is simply the expected return:

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

$$v_\pi(s) = r + v_\pi(s')$$

# Reinforcement Learning examples:




(a) robotics,
(b) Ms. Pac-Man,
(c) Go player,
(d) thermostat,
(e) automatic trader

# ❖ DeepMind's AlphaGo Zero is really a scientific breakthrough

✓ Role of Deep Reinforcement Learning in achieving Strong AI

- **Mastering the game of Go without human knowledge , David Silver, et al. Nature(2017)**

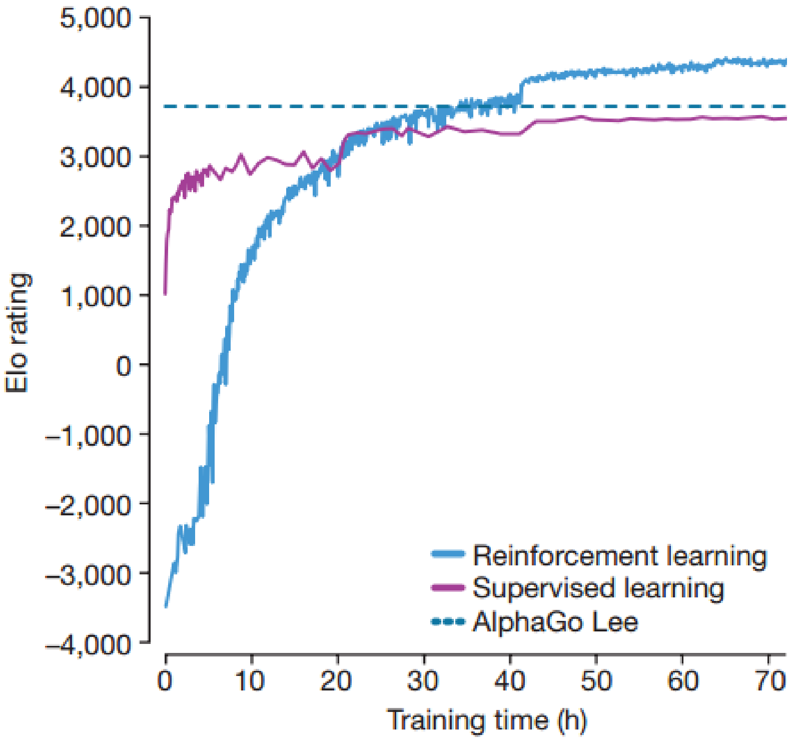




**Figure 3 | Empirical evaluation of AlphaGo Zero.**

# Reinforcement Learning examples: robotics

- The agent :
  - ✓ the program controlling a robot.
- The environment :
  - ✓ the real world
- Observation : information from the env.
  - ✓ partial description of the state of the world
- State:
  - ✓ complete description of the state of the world
- Action :
  - ✓ consist of sending signals to activate motors.
- Reward: positive/negative
  - ✓ it approaches the target destination,
  - ✓ it wastes time or goes in the wrong direction

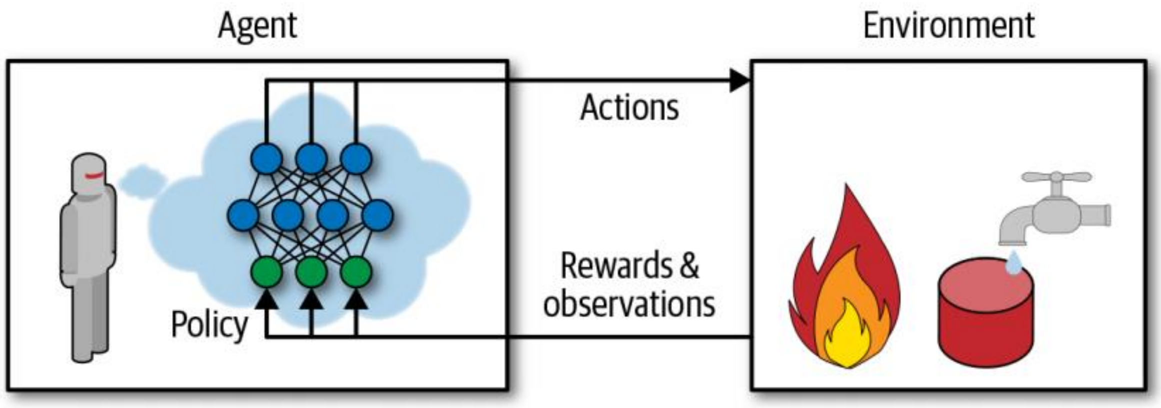
# Q-Learning

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{current value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\bigg( \underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{current value}} \bigg)}^{\text{temporal difference}}$$

# Policy Search

## ❖ Policy

- ✓ The algorithm a software agent uses to determine its actions
- ✓ A neural network taking observations as inputs and outputting the action to take

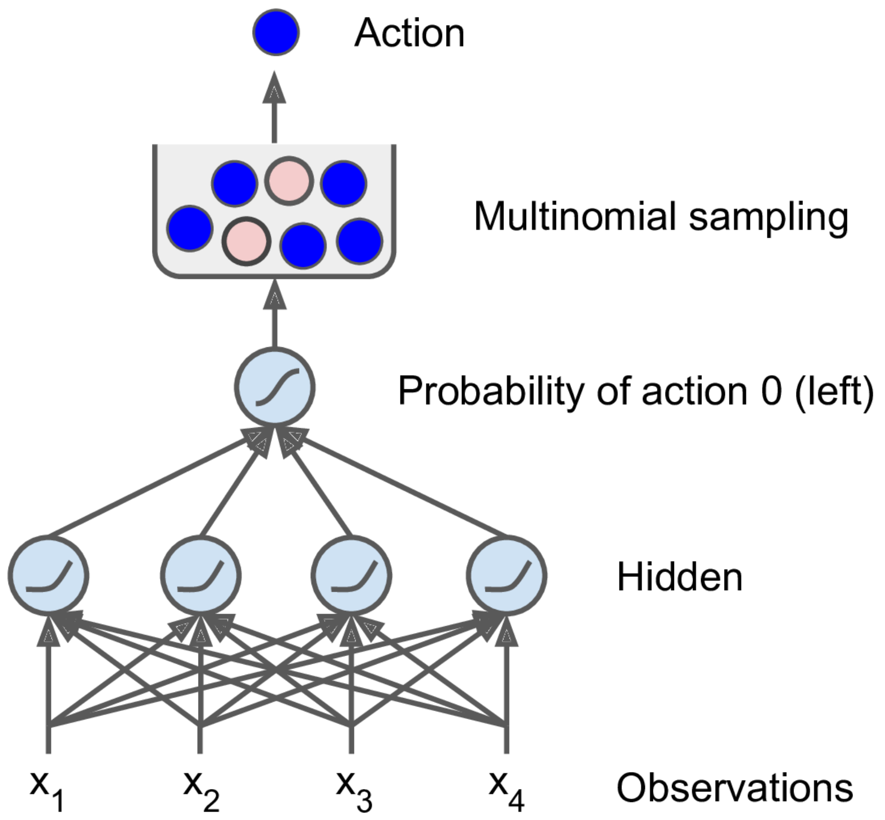




## ❖ Policy Search : Policy Gradient (PG)

- ✓ One approach is to use optimization technologies, by evaluating the gradients of the rewards with respect to policy paramenters

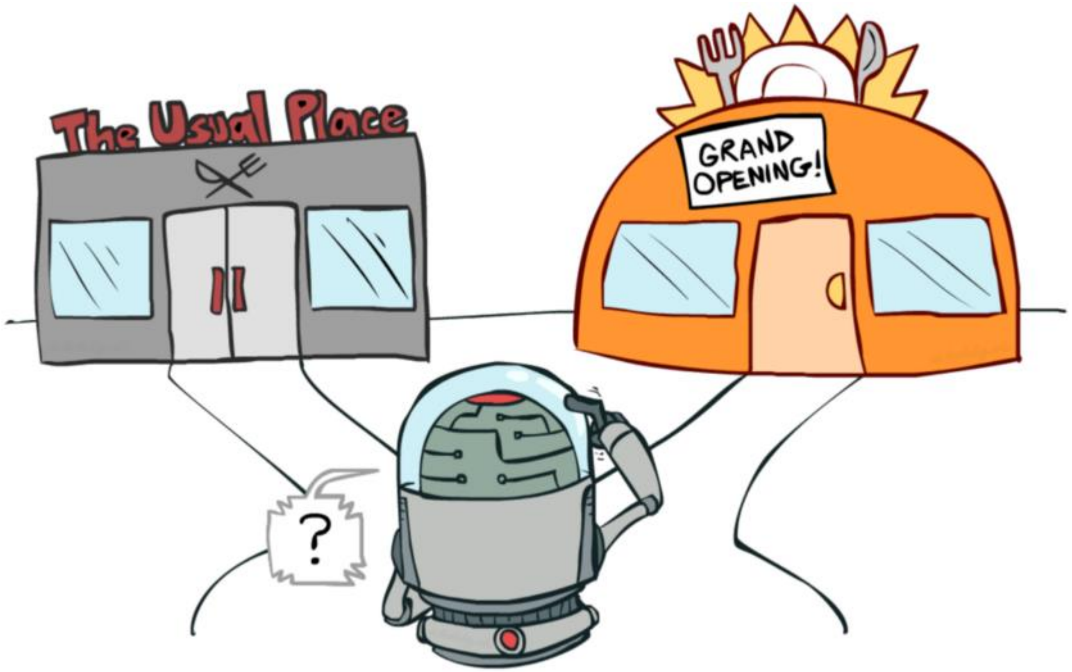
## ❖ Two possible actions

- the probability p of action 0 (left) and the probability of action 1 (right) will be 1–p



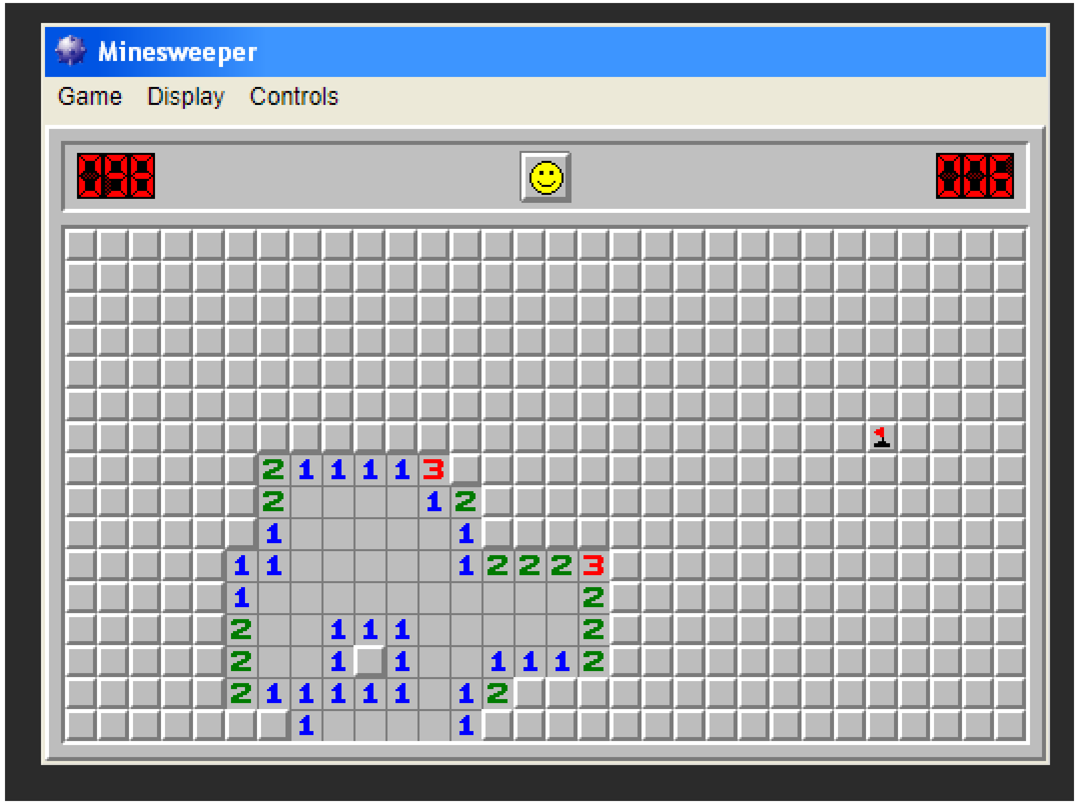
## ❖ Exploring and Exploitation

- ✓ picking a random actions
- ✓ *Exploitation*
  - *the process of taking benefits from things which we know about*
- ✓ *Exploration*
  - *to get knowledge about things which we didn’t know*



## ❖ Exploring and Exploitation Strategy

✓https://zangsi.net/minesweeper/

# Markov Decision Processes (MDP) and Bellman Equations

# Markov Decision Processes (MDPs)

- Typically we can frame all RL tasks as MDPs
- The key in MDPs is the Markov Property
  - ✓ Essentially the future depends on the present and not the past
    - More specifically, the future is independent of the past given the present
- Types of Markov Models
  - ✓ Control over state transitions and completely observable states: **MDPs**
  - ✓ Control over state transitions and partially observable states: **Partially Observable MDPs**
  - ✓ No control over state transitions and completely observable states: **Markov Chain**
  - ✓ No control over state transitions and partially observable states: **Hidden Markov Model**

- Markov chains: In the early 20th century, the Andrey Markov studied stochastic processes with no memory
  - ✓ The probability for it to evolve from a state s to a state s′ is fixed, and it depends only on the pair (s, s′), not on past states (this is why we say that the system has no memory)
- An example of a Markov chain with four states

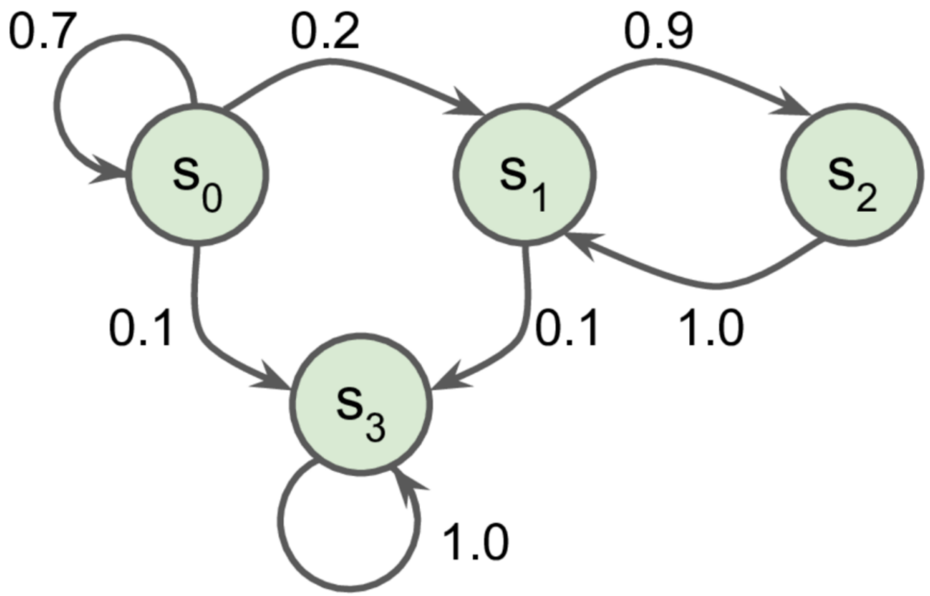



*Figure 18-7. Example of a Markov chain*

# Markov Decision Processes

- Markov Decision Processes were first described in the 1950s by Richard Bellman.
  - ✓ They resemble Markov chains but with a twist:
    - At each step, an agent can choose one of several possible actions, and the transition probabilities depend on the chosen action.
    - Moreover, some state transitions return some reward (positive or negative),
    - and the agent's goal is to find a policy that will maximize reward over time.

# State value function

- Bellman found a way to estimate the optimal state value of any state s,

$$\underbrace{V_\pi(s)}_{\text{Value of state s}} = \underbrace{\mathbf{E}_{\boxed{\pi}}[G_t}_{\text{Expected return}} | \underbrace{S_t = s}_{\text{If the agent starts at state s}}]$$

$\pi$: 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.

## ❖ The action-value function

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

Value of state-action pair s,a

Expected return

If the agent starts at state s

and chooses action a

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

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 example:

- ✓ the MDP has three states (represented by circles) and up to three possible discrete actions at each step (represented by diamonds)
  - R(s,a,s') is the reward that the agent gets when it goes from state s to state s', given that the agent chose action a.

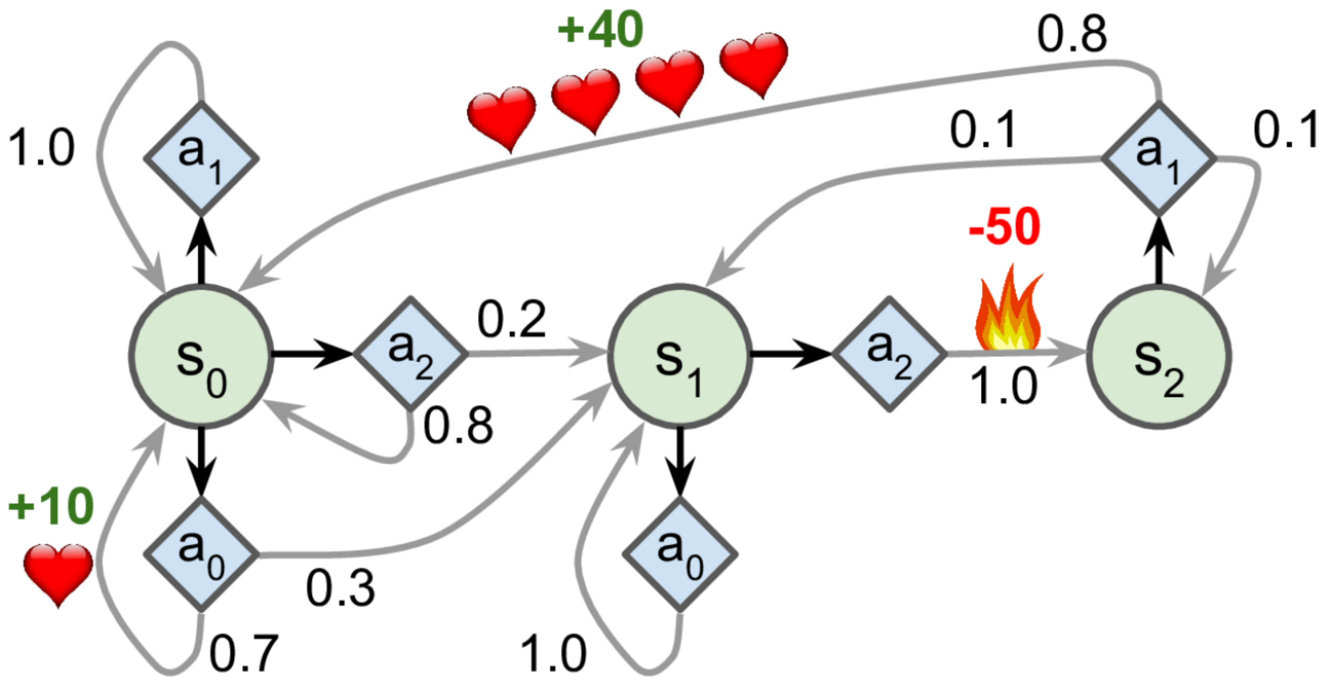


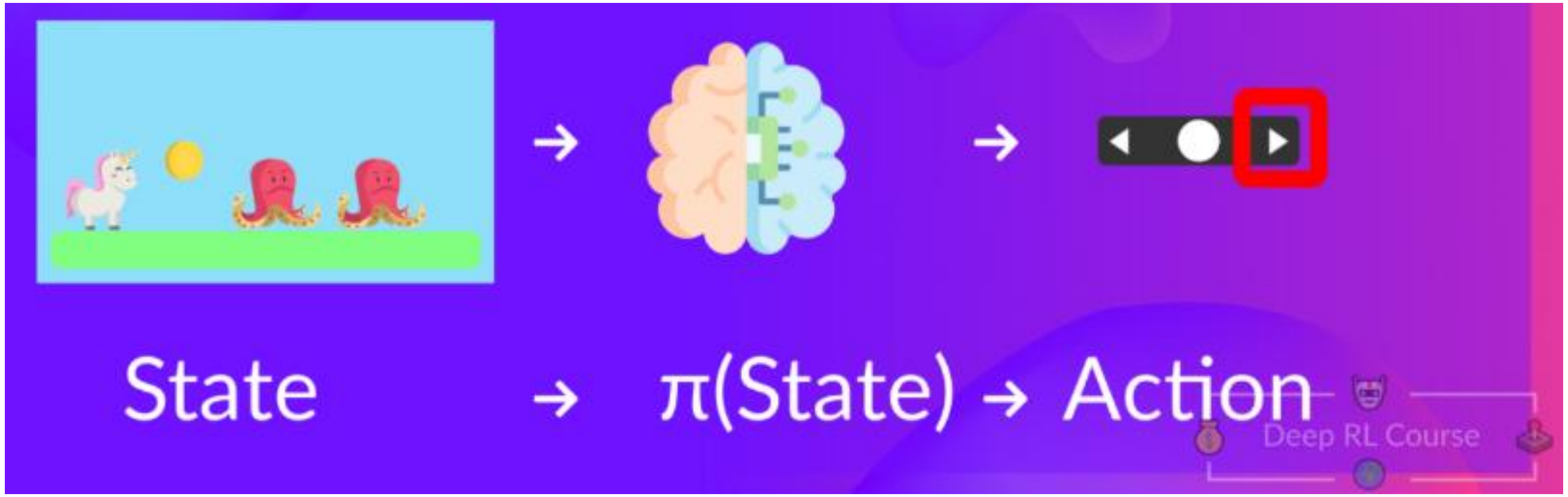

*Figure 18-8. Example of a Markov decision process*

$$T(s_2, a_1, s_0) = 0.8$$

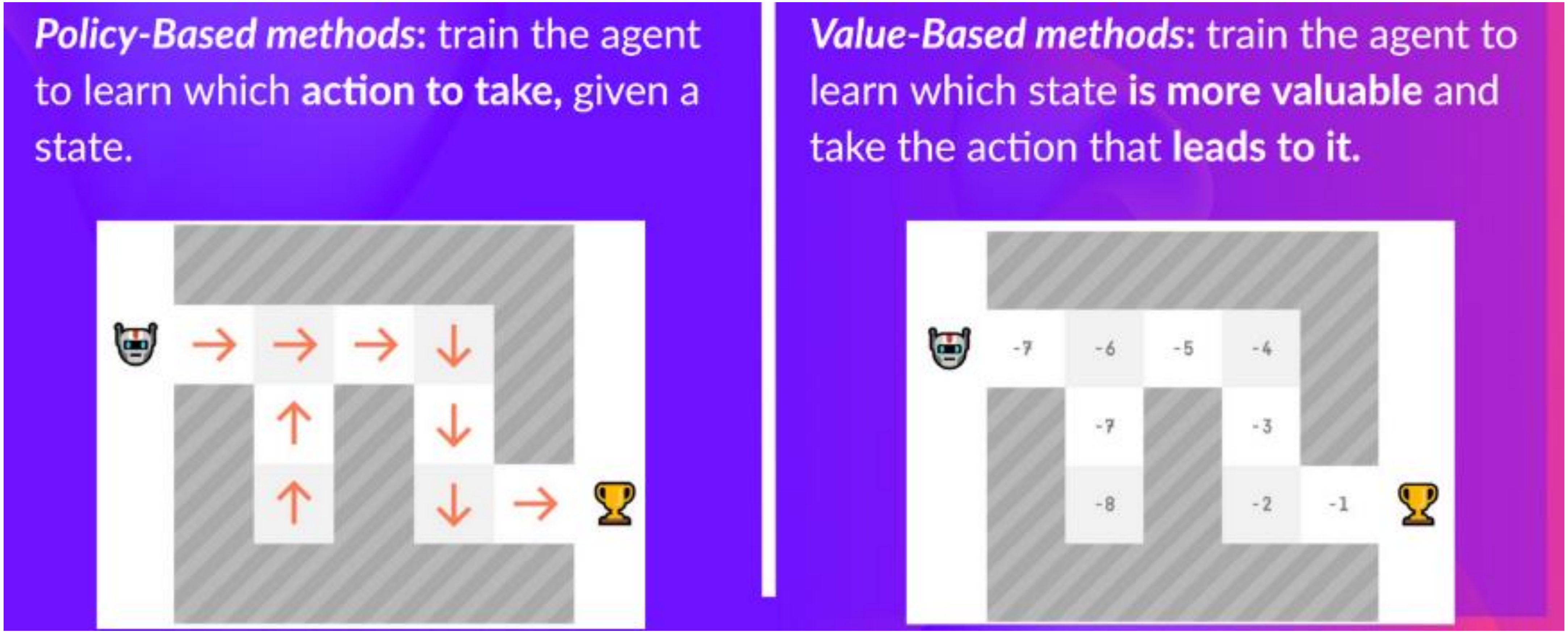
$$R(s_2, a_1, s_0) = +40$$

## The policy

- ✓ the agent's decision-making process
- ✓ The Policy π is the **brain of our Agent**
  - An agent **select the actions that maximize its expected cumulative reward**



# There are two main types of RL methods



# The link between Value and Policy:

- In policy-based: train directly the policy
  - ✓ Our policy is a Neural Network
  - ✓ No value function
- In value-based: don't train the policy
  - ✓ Our policy is a function defined by hand
  - ✓ Instead train a value-function that is a Neural Network
- Finding an optimal value fuction leads to having an optimal policy

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

KISTI SINCE 1962

## ❖ The Bellman Equation is one central to Markov Decision Processes

✓ The Bellman Equation

- “what is the maximum reward an agent can receive if they make the optimal action now and for all future decisions?”

The expected return (value) at the current state *s* is:

The expected reward for taking action *a* at state *s*…

$$\boxed{V(s)} = \boxed{\max_a}\big(\boxed{R(s,a)} + \boxed{\gamma V(s')}\big)$$

The maximum value of any possible action *a* for:

…plus the discount factor (gamma) multiplied by the value of the next state

# Bellman Optimality Equation

- Bellman found a way to estimate the optimal state value of any state s

*Equation 18-1. Bellman Optimality Equation*

$$V^*(s) = \max_a \sum_s T(s, a, s') \left[R(s, a, s') + \gamma \cdot V^*(s')\right] \quad \text{for all } s$$

$T(s_2, a_1, s_0) = 0.8$. $R(s_2, a_1, s_0) = +40$ $\gamma$ is the discount factor.

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma \cdot V_k(s')\right] \quad \text{for all } s$$

# Quality Values

- Knowing the optimal state-values can be useful, in particular to evaluate a policy, but it does not give us the optimal policy for the agent.
- Q-Values (Quality Values):
  - ✓ Bellman found a very similar algorithm to estimate the optimal state-action values
  - ✓ Q*(s, a): the optimal Q-Value of the state-action pair (s, a):
  - ✓ Defining the optimal policy, noted π*(s):

$$Q_{k+1}(s,a) \leftarrow \sum_{s\prime} T(s,a,s\prime)\left[R(s,a,s\prime) + \gamma \cdot \max_{a\prime} Q_k(s\prime,a\prime)\right] \quad \text{for all } (s\prime a)$$

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

KISTI SINCE 1962

- Monte Carlo Approach:
  - ✓ Monte Carlo uses **an entire episode of experience before learning**
  - ✓ **waits until the end of episode, then calculates return(G_t) and use it as a target for its value of policy**
- Temporal Difference (TD)
  - ✓ uses **only a step to learn**

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

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

# Temporal Difference (TD) Learning

# Temporal Difference (TD) Learning

KISTI SINCE 1962

- The TD Learning algorithm is very similar to the Value Iteration algorithm
  - ✓ the TD Learning algorithm updates the estimates of the state values based on the transitions and rewards that are actually observed
  - ✓ Goal:
    - learn $V_\pi(s)$ from episodes of experience under policy $\pi$

$$V_{k+1}(s) \leftarrow V_k(s) + \alpha(\underbrace{\underbrace{R + \gamma V_k(s')}_{\text{TD target}} - V_k(s)}_{\text{TD error}})$$

*TD target*

*TD error*

*\alpha is the learning rate (e.g., 0.001)*

$$V(s) \underset{\alpha}{\leftarrow} r + \gamma \cdot V(s\prime)$$

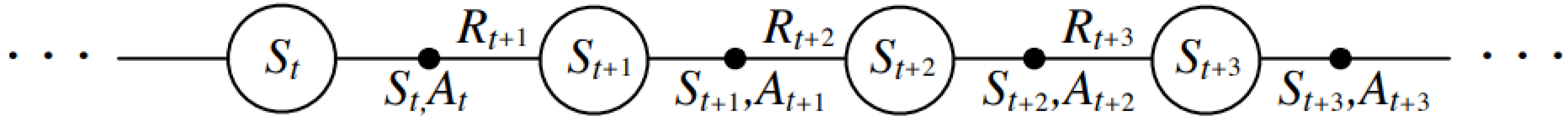
# Temporal Difference (TD) Learning

- Policy evaluation (the prediction prebole):
  - ✓ for a given policy $\pi$, compute the state-value function $V_\pi(s)$
- The simplest Temporal-Difference method TD(0):

$$V_{k+1}(s) \leftarrow V_k(s) + \alpha(\underline{R + \gamma V_k(s')} - V_k(s))$$

*TD target: an estimate of the return*

- SARSA is an on-policy algorithm
  - ✓ while learning the optimal policy it uses the current estimate of the optimal policy to generate the behavior
- Estimate optimal policy $q_\pi$ for the current policy $\pi$




  - ✓ After every transition from a nonterminal state, S_t, do this:

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

$$\text{If } S_{t+1} \text{ is terminal, then define } Q(S_{t+1}, A_{t+1}) = 0$$

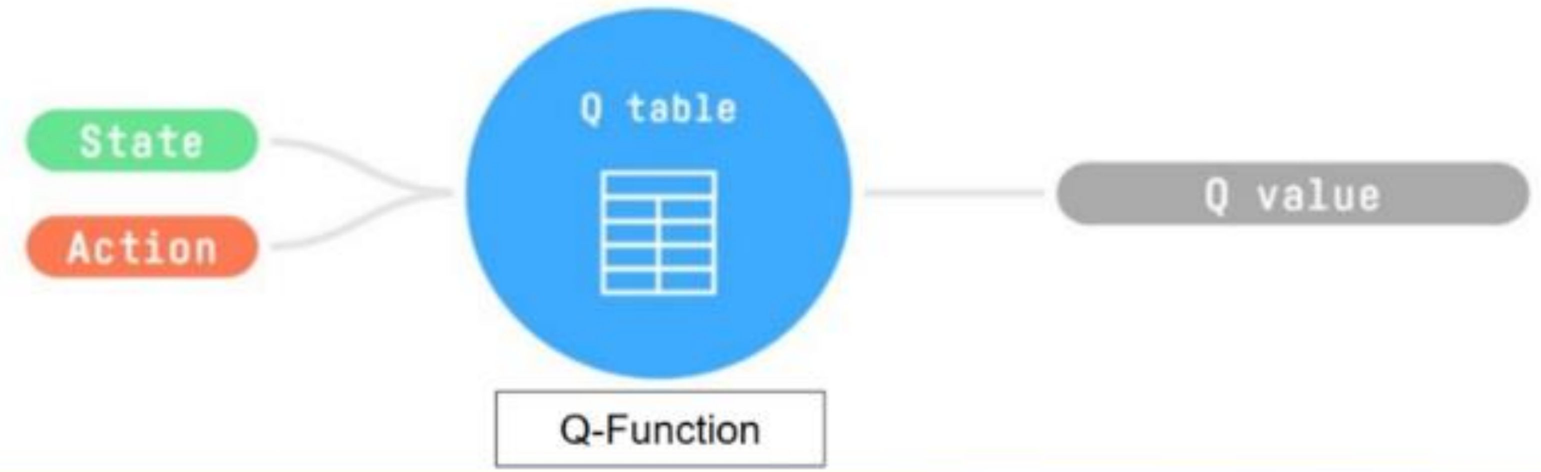
# Q-Learning

# What is Q-Learning?

- Q-Learning is an off-policy value-based method
  - ✓ that uses a Temporal Differnence(TD) approach to train its action-value function
  - ✓ Value-based method
    - finds the optimal policy indirectly by training a value or action-value function that will tell us **the value of each state or each state-action pair.**
  - ✓ Uses a TD approach:
    - **updates its action-value function at each step instead of at the end of the episode.**
  - ✓ Off-policy:
    - using **a different policy for acting and updating using epsilon-greedy policy again**

# What is Q-Learning?

- Q-Learning is the algorithm to train an action-value function
  - ✓ that determines the value of being at a particular state and taking a specific action at that state.
    - Given a state and action,  Q function outputs a state-action value (also called Q-value)




https://huggingface.co/blog/deep-rl-q-part2

# Q-learning: Off-Policy TD Control

KISTI SINCE 1962

- In Q-learning the learned action-value function, Q, directly approximates the optimal action-value function, independent of the policy being followed.
    - the transition probabilities are unknown and the rewards are initially unknown
    - Q-Learning works by watching an agent play (e.g., randomly) and gradually improving its estimates of the Q-Values

$$Q(s_t, a_t) \leftarrow (s_t, a_t) + \alpha \left( r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right)$$

$$Q(s_t, a_t) \leftarrow \boxed{Q(s_t, a_t)} + \boxed{\alpha} [\boxed{r_{t+1}} + \boxed{\lambda} \boxed{\max_{a} Q(s_{t+1}, a)} - \boxed{Q(s_t, a_t)}]$$

- Current Q-table value we are updating
- Learning rate
- Reward
- Discount
- Estimated reward from our next action

# Q-Learning is called an Off-Policy

## ❖ Off-policy: The Q-learning algorithm

✓ work by "*looking over someone's shoulder.*"

✓ the algorithm attempts to learn about policy π from experience sampled from μ.

✓ the policy being executed is completely random, while the policy being trained will always choose the actions with the highest Q-Values.

- Q-Learning is capable of learning the optimal policy by just watching an agent act randomly
- learning to play golf when your teacher is a drunk monkey

## ❖ On-policy: The Policy Gradients algorithm, and SARSA

✓ We can say that algorithms classified as **on-policy** are "*learning on the job.*"

- In other words, the algorithm attempts to learn about policy π from experience sampled from π.
- it explores the world using the policy being trained.