Deep Learning

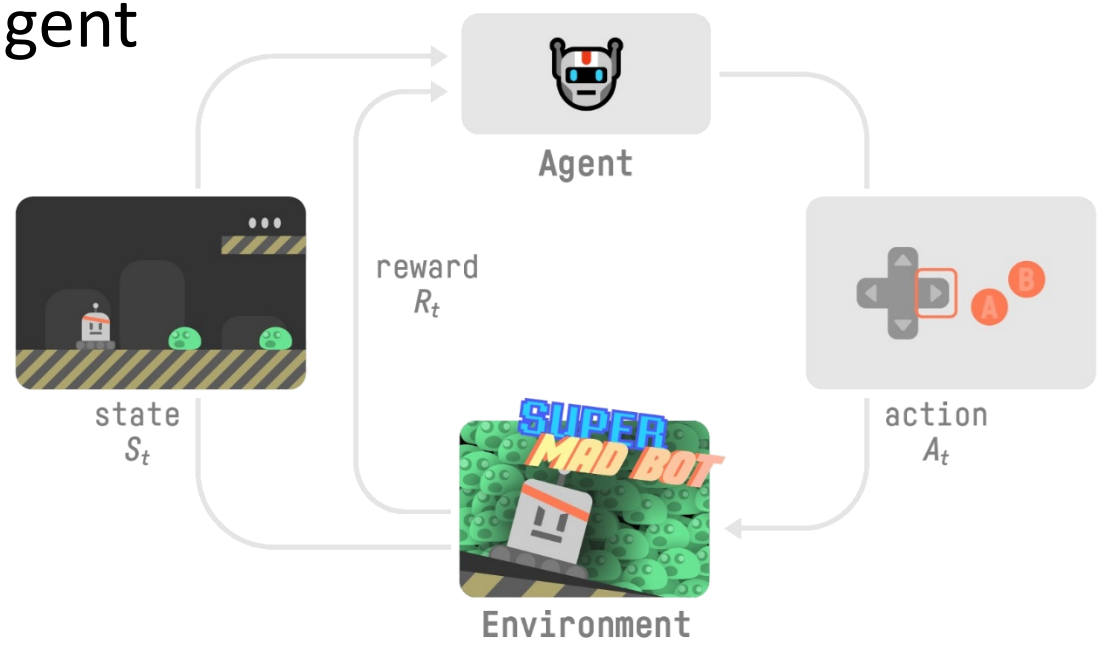
Mohammad Reza Mohammadi

2021

RL process

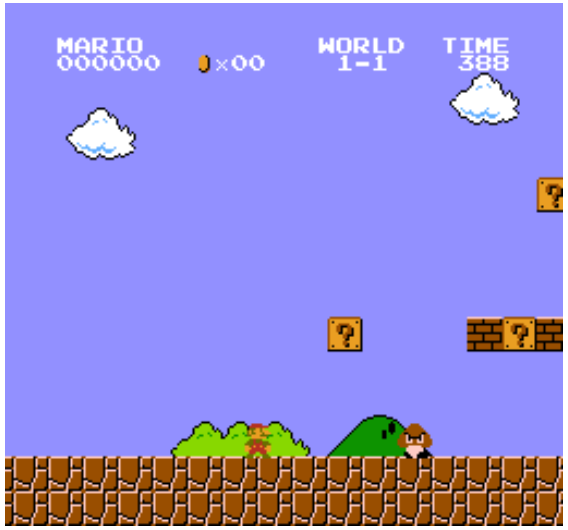
- Our agent receives state S0 from the environment
- Based on that state S0, the agent takes an action A0
- Environment transitions to a new state S1
- Environment gives some reward R1 to the agent

- The goal of the agent is to maximize its cumulative reward
  - called the expected return



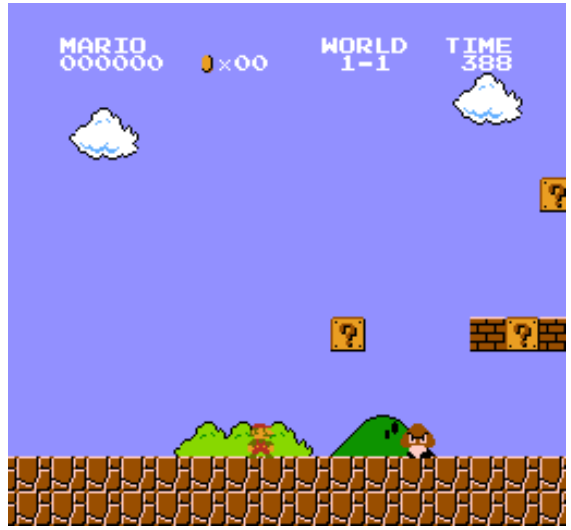
Observations/States space

- Observations/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



Action space

- The action space is the set of all possible actions in an environment
- The actions can come from a discrete or continuous space:
  - Discrete space: the number of possible actions is finite
  - Continuous space: the number of possible actions is infinite



Rewards

- The reward is fundamental in RL because it's the only feedback for the agent
- Thanks to it, our agent knows if the action taken was good or not
- The cumulative reward at each time step t can be written as:

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

- r_t: reward at time t
- τ: trajectory (sequence of states and actions)
- $R(\tau)$: return (cumulative reward)

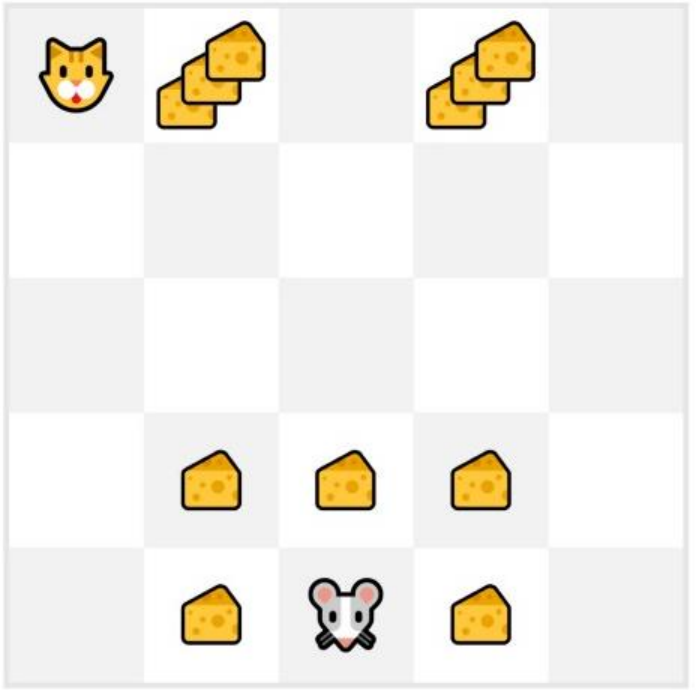
$$R(\tau) = \sum_{k=0}^{\infty} r_{t+k+1}$$

Rewards and discounting

- The rewards that come sooner are more probable to happen, since they are more predictable than the long term future reward
- To discount the rewards, we proceed like this:
  - We define a discount rate called gamma (between 0 and 1)
    - The larger the gamma, the smaller the discount
  - Each reward will be discounted by gamma to the exponent of the time step

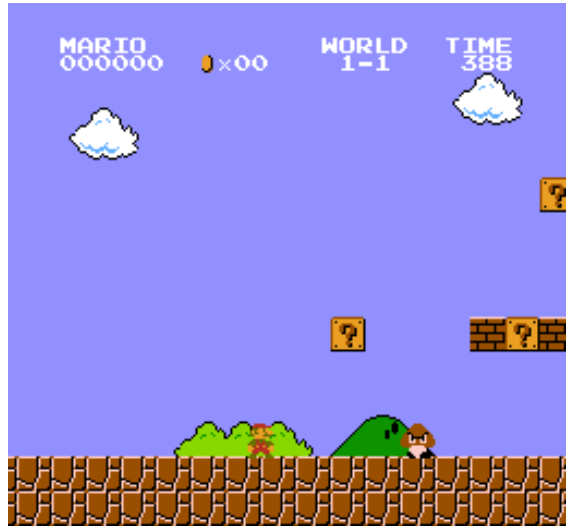
$$R(\tau) = r_{t+1} + \gamma\, r_{t+2} + \gamma^2\, r_{t+3} + \cdots$$

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



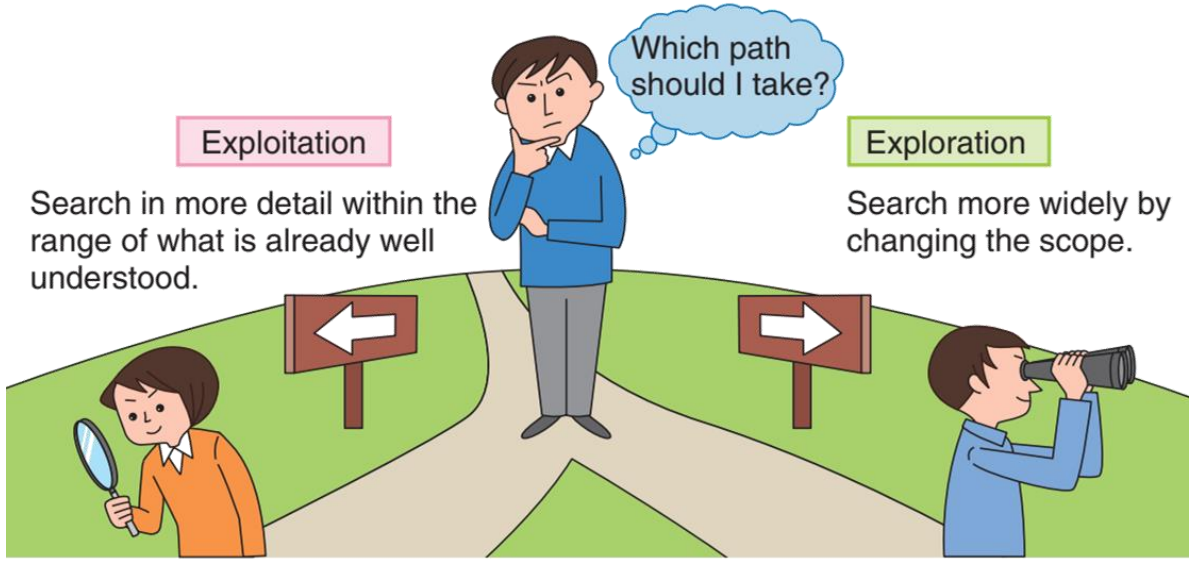
Type of tasks

- We can have two types of tasks: episodic and continuous
- Episodic task: we have a starting point and an ending point (a terminal state)
- Continuous task: these are tasks that continue forever (no terminal state)



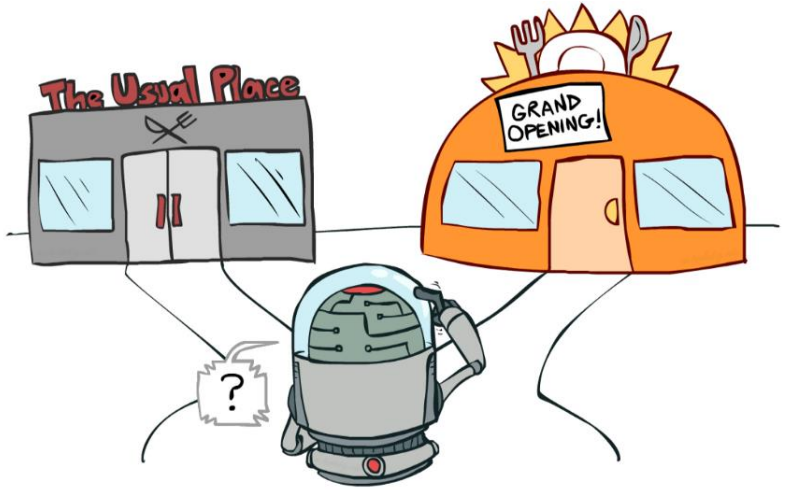
Exploration/Exploitation tradeoff

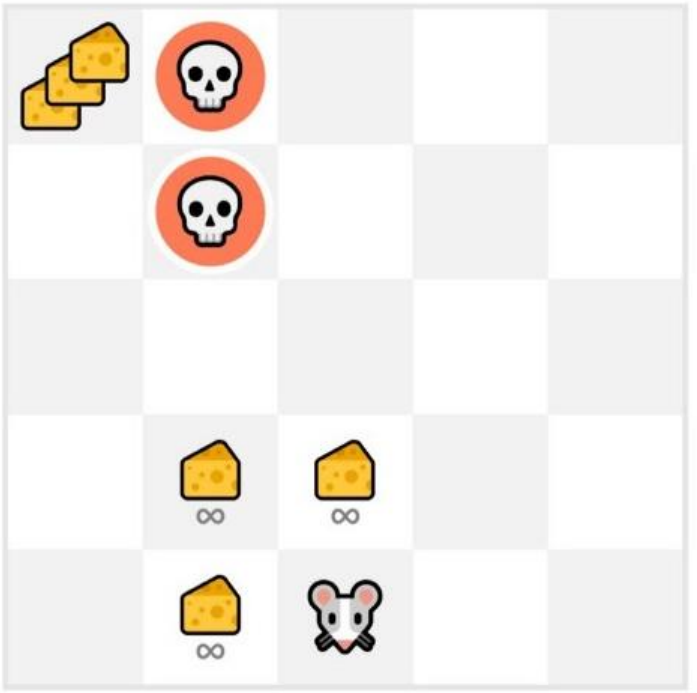
- To obtain a lot of reward, an agent must prefer actions that it has tried in the past
- But to discover such actions, it has to try actions that it has not selected before
- The agent has to exploit what it has already experienced in order to obtain reward, but it also has to explore in order to make better action selections in the future



Example

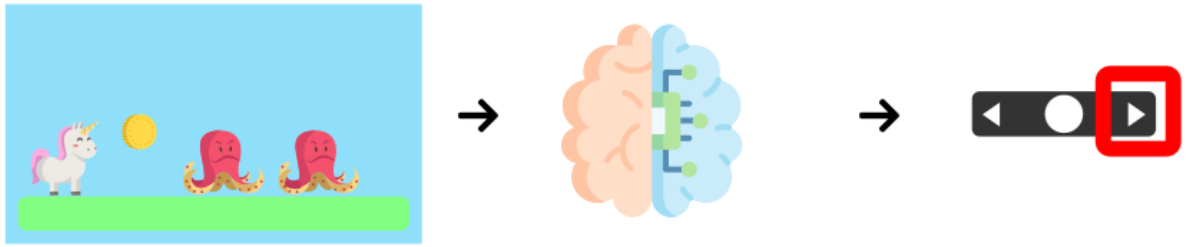
- Consider our mouse can have an infinite amount of small cheese (+1 each)
- But at the top of the maze, there is a gigantic sum of cheese (+1000)
  - If we only focus on exploitation, our agent will never reach the gigantic sum of cheese
  - If our agent does a little bit of exploration, it can discover the big reward





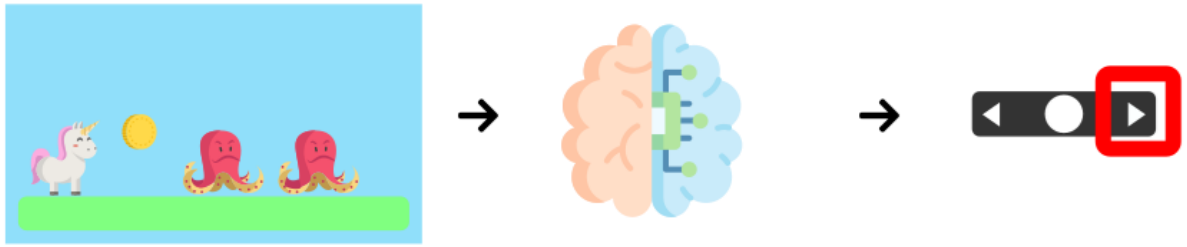
Policy

- The policy π is the brain of our agent
  - It's the function that tell us what action to take given the state we are
- This 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



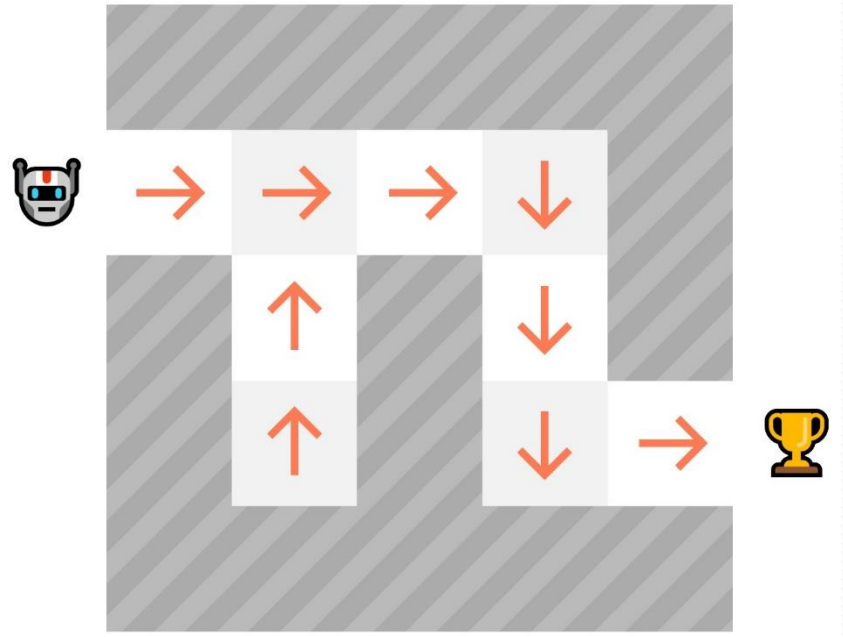
Policy

- There are two approaches to train our agent to find this optimal policy π^*:
  - Policy-based methods
    - Directly, teach the agent to learn which action to take, given the state is in
  - Value-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



Policy-based methods

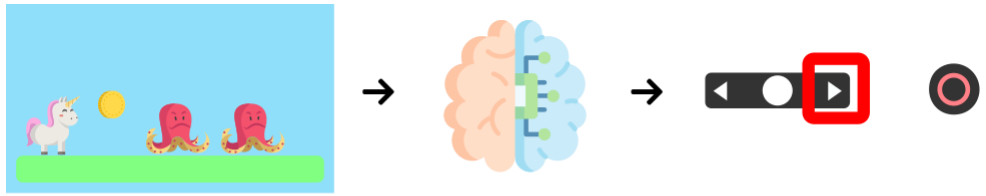
- In policy-based methods, we learn a policy function directly
- This function will map from each state to the best corresponding action
- Or a probability distribution over the set of possible actions at that state



Policy-based methods

- We have two types of policy:
  - Deterministic

$$a = \pi(s)$$



State s_0 → $\pi(s_0)$ → a_0 = Right

  - Stochastic

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

State s_0 → $\pi(A|s_0)$ → [Left: 0.1, Right: 0.7, Jump: 0.2]

Value-based methods

- Instead of training a policy function, we train 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 act according to our policy
  - "Act according to our policy" just means that our policy is "going to the state with the highest value"

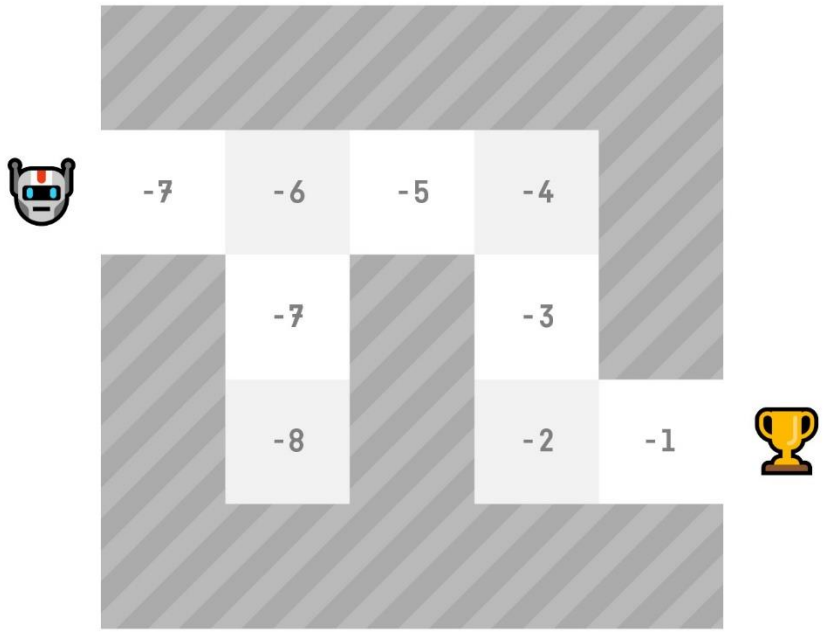
$$\underbrace{v_\pi(s)}_{\text{Value function}} = \underbrace{\mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots\right]}_{\text{Expected discounted return}} \Big| \underbrace{S_t = s}_{\text{Starting at state s}}$$

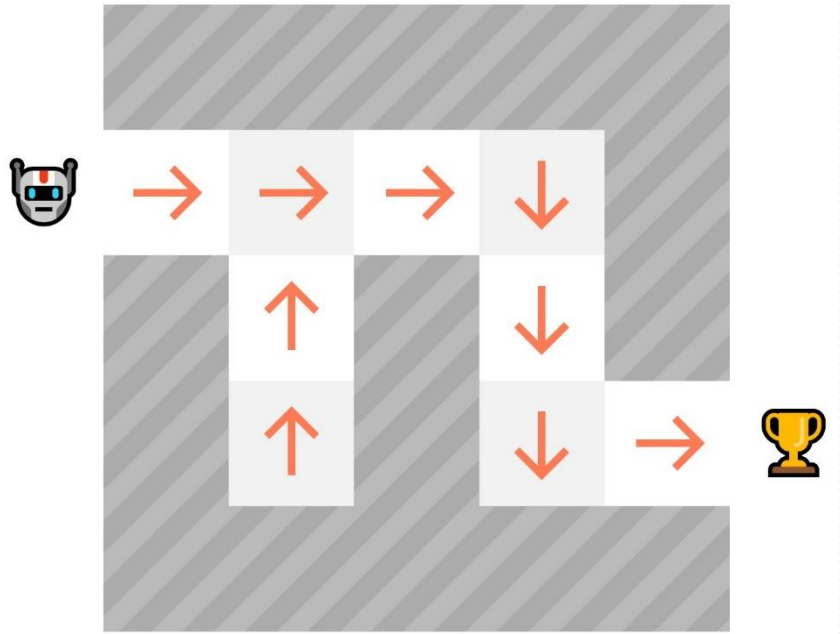
Value-based methods

- At each step, our policy will select the state with the biggest value defined by the value function

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

Value function — Expected discounted return — Starting at state s





Value-based methods

- Because we didn't train our policy, we need to specify its behavior
  - For instance, if we want a policy that given the value function will take actions that always lead to the biggest value, we'll create a Greedy Policy

$$\pi(s) = \arg\max_a Q_\pi(s, a)$$

- So the difference is:
  - In policy-based the optimal policy is found by training the policy directly
  - In value-based, finding an optimal value-function leads to having an optimal policy

Tabular solution methods

- The state and action spaces are small enough for the approximate value functions to be represented as arrays, or tables
- In this case, the methods can often find exact solutions
  - exactly the optimal value function and the optimal policy
- This contrasts with the approximate methods
  - which only find approximate solutions
  - can be applied effectively to much larger problems

Initialized

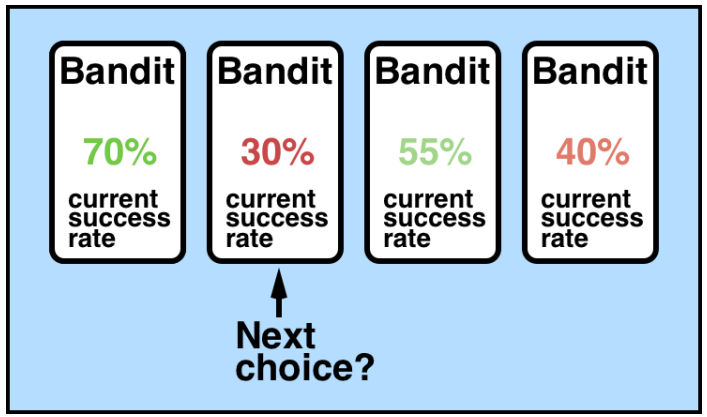
Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

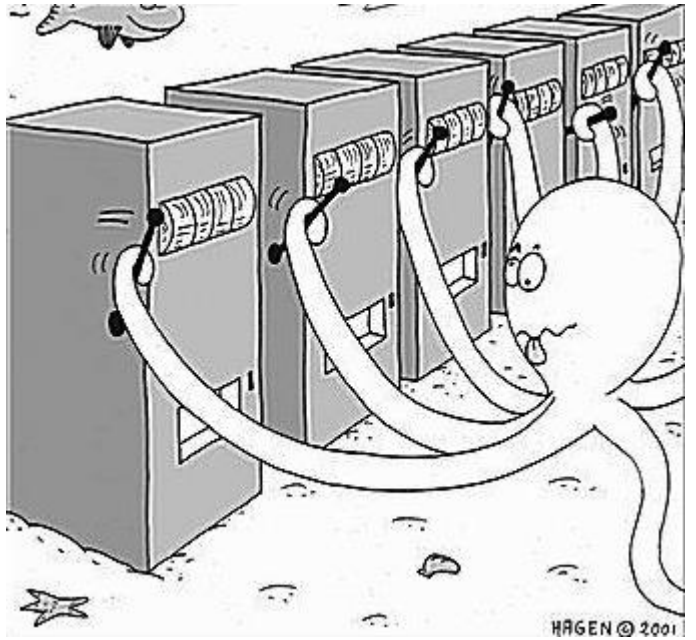
Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

A k-armed bandit problem

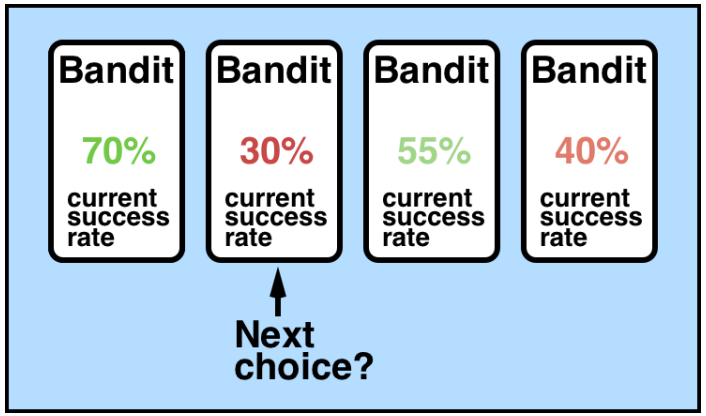
- You are faced repeatedly with a choice among k different options, or actions
- After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected
- Your objective is to maximize the expected total reward over some time period, for example, over 1000 action selections, or time steps

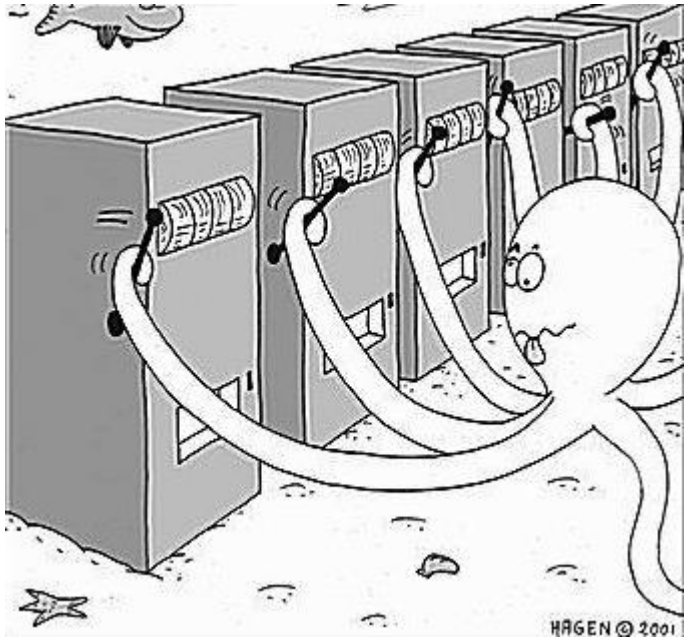




A k-armed bandit problem

- Each action selection is like a play of one of the slot machine's levers, and the rewards are the payoffs for hitting the jackpot
- Through repeated action selections you are to maximize your winnings by concentrating your actions on the best levers





A k-armed bandit problem

- In our k-armed bandit problem, each of the k actions has an expected or mean reward given that action is selected
- Let us call this the value of that action
- We denote the action selected on time step t as A_t, and the corresponding reward as R_t
- The value then of an arbitrary action a, denoted by $q_*(a)$

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

- If you knew the value of each action, then it would be trivial to solve the problem: you would always select the action with highest value

A k-armed bandit problem

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

- We denote the estimated value of action a at time step t as $Q_t(a)$
- If you maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest
- We call these the greedy actions (exploiting your current knowledge)
- If instead you select one of the nongreedy actions, then we say you are exploring, because this enables you to improve your estimate of the nongreedy action's value
- Exploitation is the right thing to do to maximize the expected reward on the one step, but exploration may produce the greater total reward in the long run

A k-armed bandit problem

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

- If you have many time steps ahead on which to make action selections, then it may be better to explore the nongreedy actions and discover which of them are better than the greedy action
- These methods estimate the values of actions and use the estimates to make action selection decisions
- One natural way to estimate the value is by averaging the rewards actually received

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

A k-armed bandit problem

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

- We call this the sample-average method for estimating action values because each estimate is an average of the sample of relevant rewards
- Greedy action selection:

$$A_t \doteq \arg\max_a Q_t(a)$$

A k-armed bandit problem

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a]$$

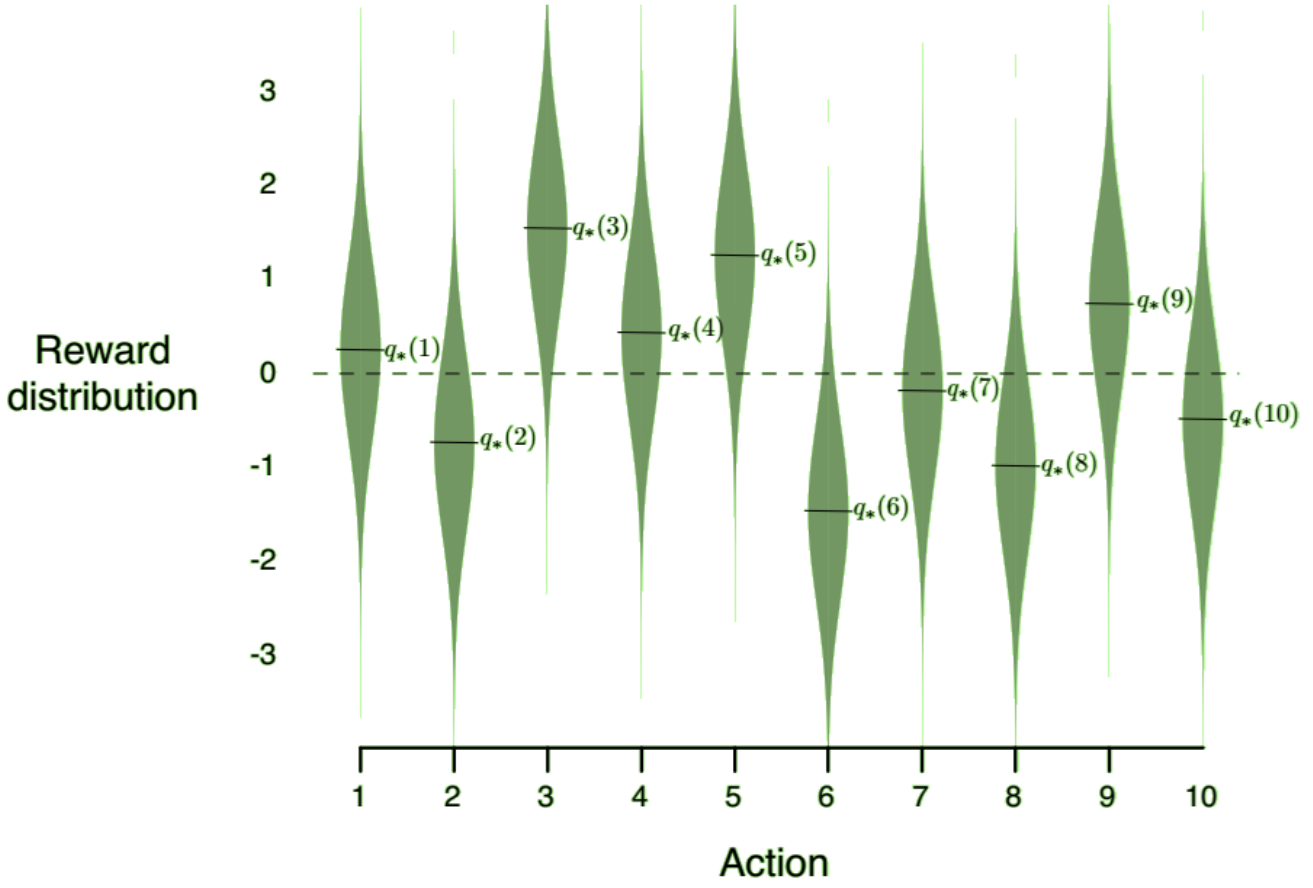
$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot 1_{A_i=a}}{\sum_{i=1}^{t-1} 1_{A_i=a}}$$

$$A_t \doteq \arg\max_a Q_t(a)$$

- A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability ε, instead select randomly from among all the actions with equal probability, independently of the action-value estimates
  - We call these methods ε-greedy methods

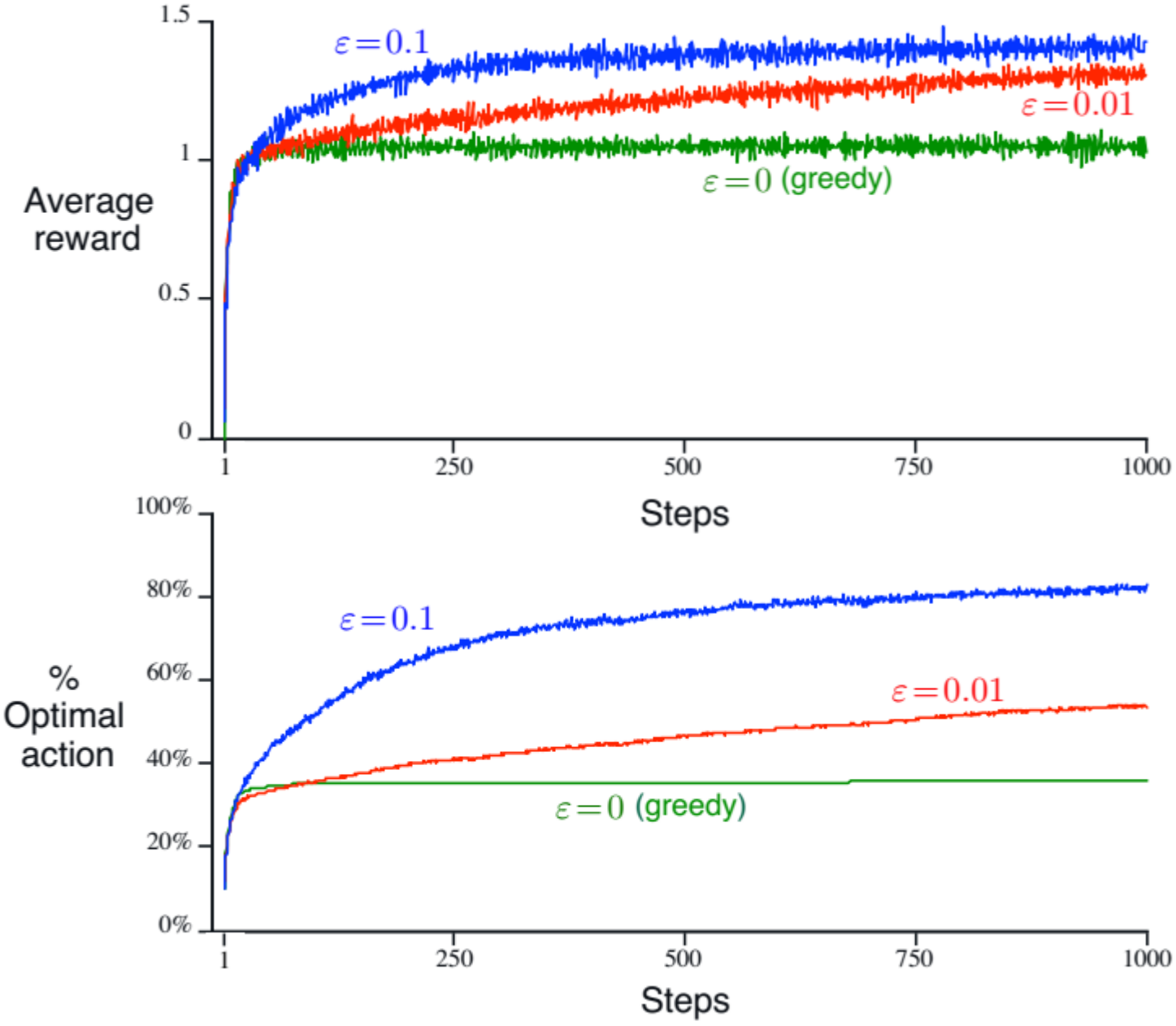
The 10-armed testbed

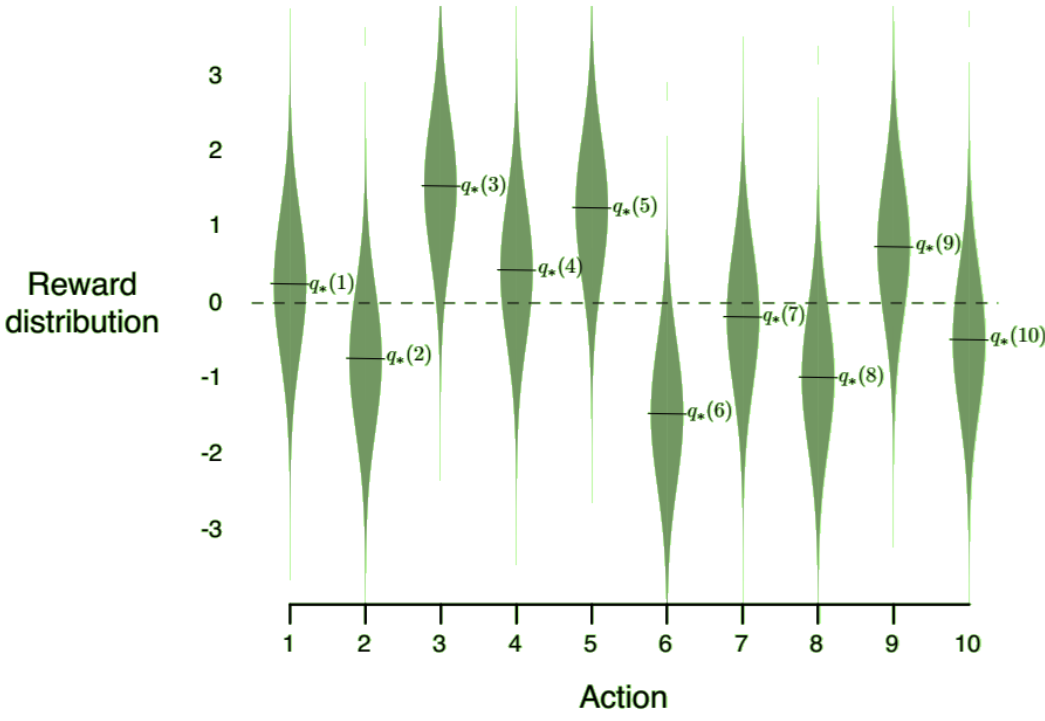
- We generate a set of 2000 randomly generated k-armed bandit problems with k = 10
- For each problem, the action values, $q_*(a), a = 1, \ldots, 10$, were selected according to a normal (Gaussian) distribution with mean 0 and variance 1.



The 10-armed testbed

- When a learning method applied to that problem selected action A_t at time step t, the actual reward, R_t, was selected from a normal distribution with mean $q_*(A_t)$, and variance 1
- For any learning method, we can measure its performance and behavior as it improves with experience over 1000 time steps
- Repeating this for 2000 independent runs, each with a different bandit problem, we obtained measures of the learning algorithm's average behavior


Average reward
1.5
1
0.5
0
$\varepsilon = 0.1$
$\varepsilon = 0.01$
$\varepsilon = 0$ (greedy)
1
250
500
750
1000
Steps
% Optimal action
100%
80%
60%
40%
20%
0%
$\varepsilon = 0.1$
$\varepsilon = 0.01$
$\varepsilon = 0$ (greedy)
1
250
500
750
1000
Steps


Reward distribution
3
2
1
0
-1
-2
-3
$q_*(1)$
$q_*(2)$
$q_*(3)$
$q_*(4)$
$q_*(5)$
$q_*(6)$
$q_*(7)$
$q_*(8)$
$q_*(9)$
$q_*(10)$
1
2
3
4
5
6
7
8
9
10
Action