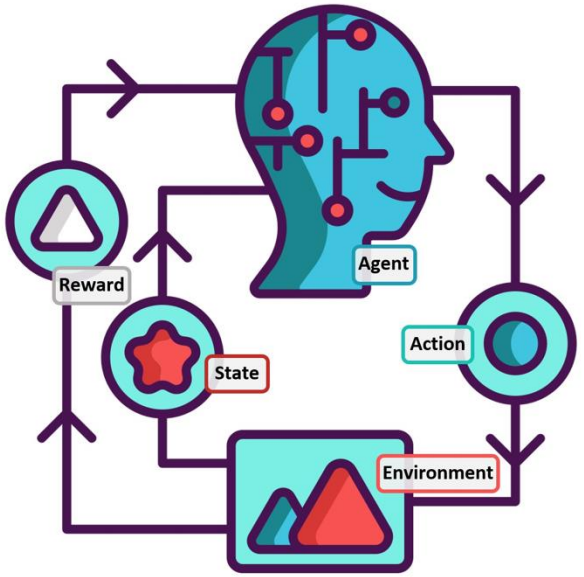
AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #02
RL Elements + Multi-armed Bandits

Gian Antonio Susto



Announcement before starting

- First partial: 7th of November
- Next labs

~~[Lab 1a] 'Deep Learning frameworks' Thursday 2 Oct (today!) 2025 8:40-10:10 Room Te~~

[Lab 1b] 'Deep Learning frameworks' Friday 3 Oct 2025 (tomorrow) 14:30-16:00 Room De (to be confirmed)

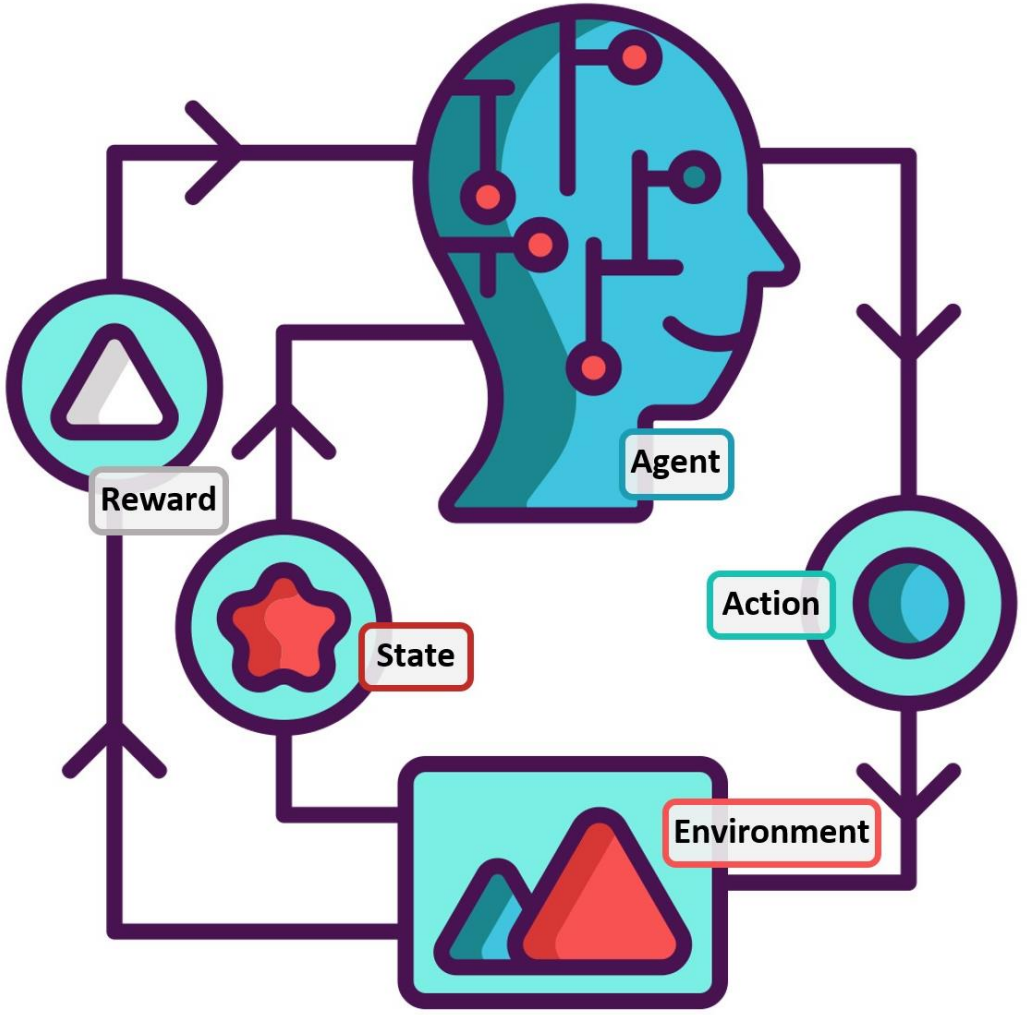
- Lab 1a and 1b can be considered as alternative (one on Tensorflow, the other on Pytorch)

The RL Elements

Reinforcement Learning

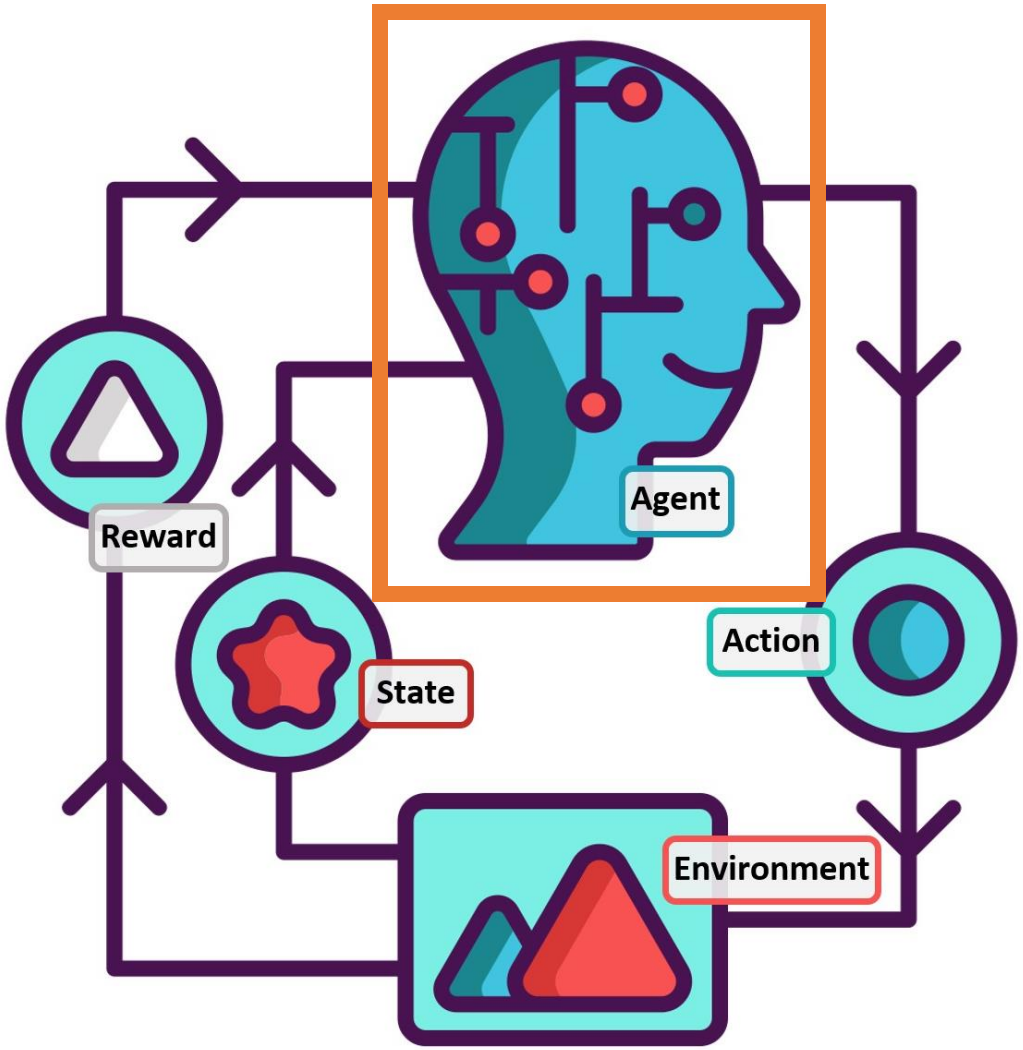
- A unique learning framework in the field of Machine Learning
- An approach that:

1. Do not rely on historical data;
2. Learn by interacting with the world;
3. Allow to combine learning and decision making;
4. Allow to achieve long-term goals
5. Exploit partial information (rewards) that we collect along the way



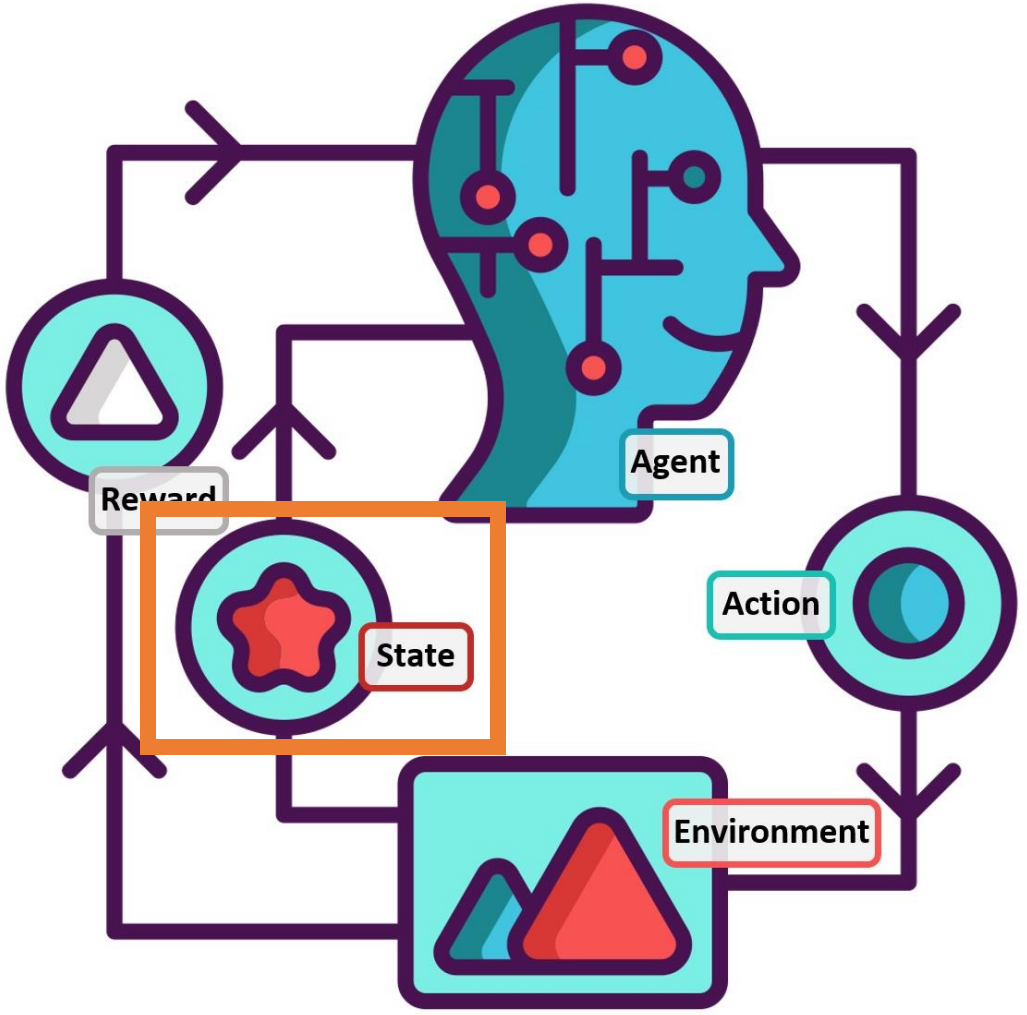
RL Formalism: The elements…

- An agent: the entity aiming at ‘solving a task’



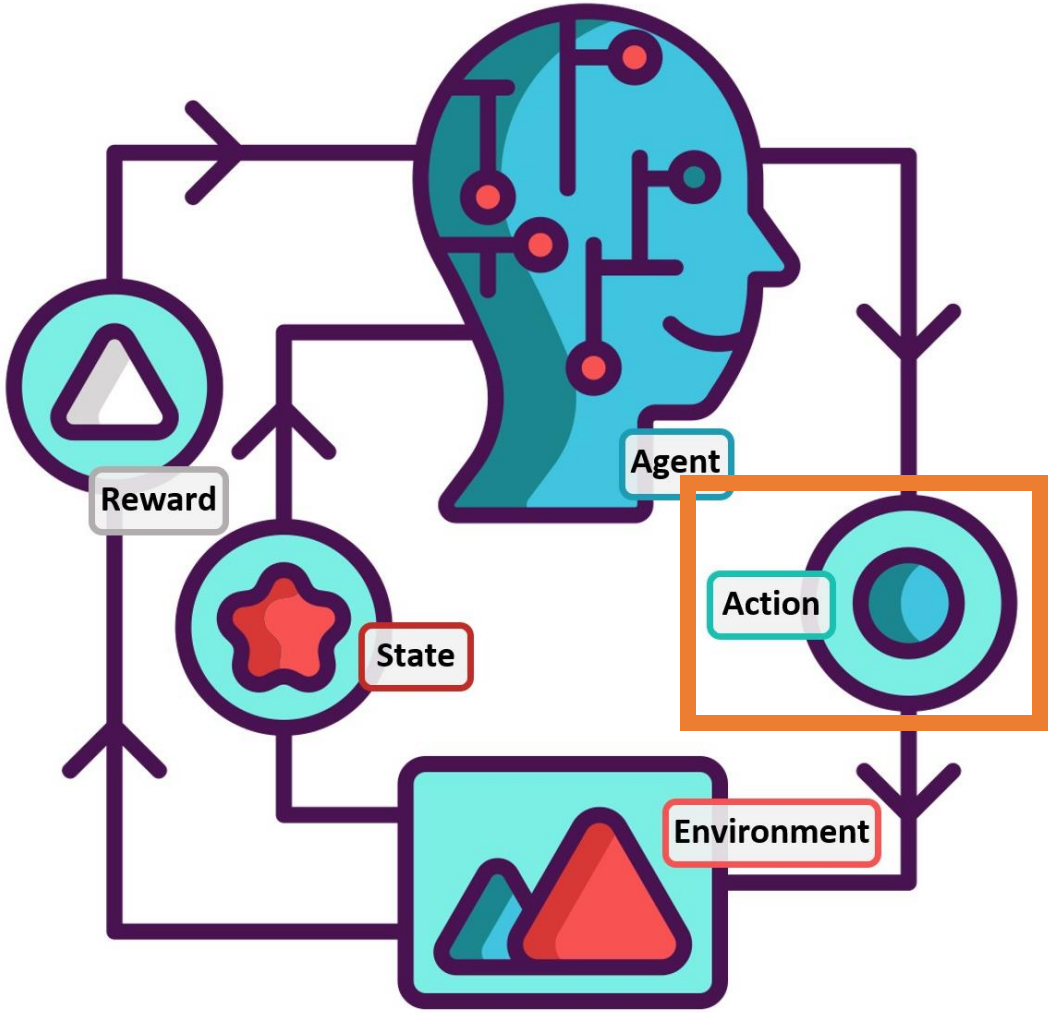
RL Formalism: The elements…

- An agent: the entity aiming at ‘solving a task’
- A set of states (exhaustive description of the system) in which the agent can be



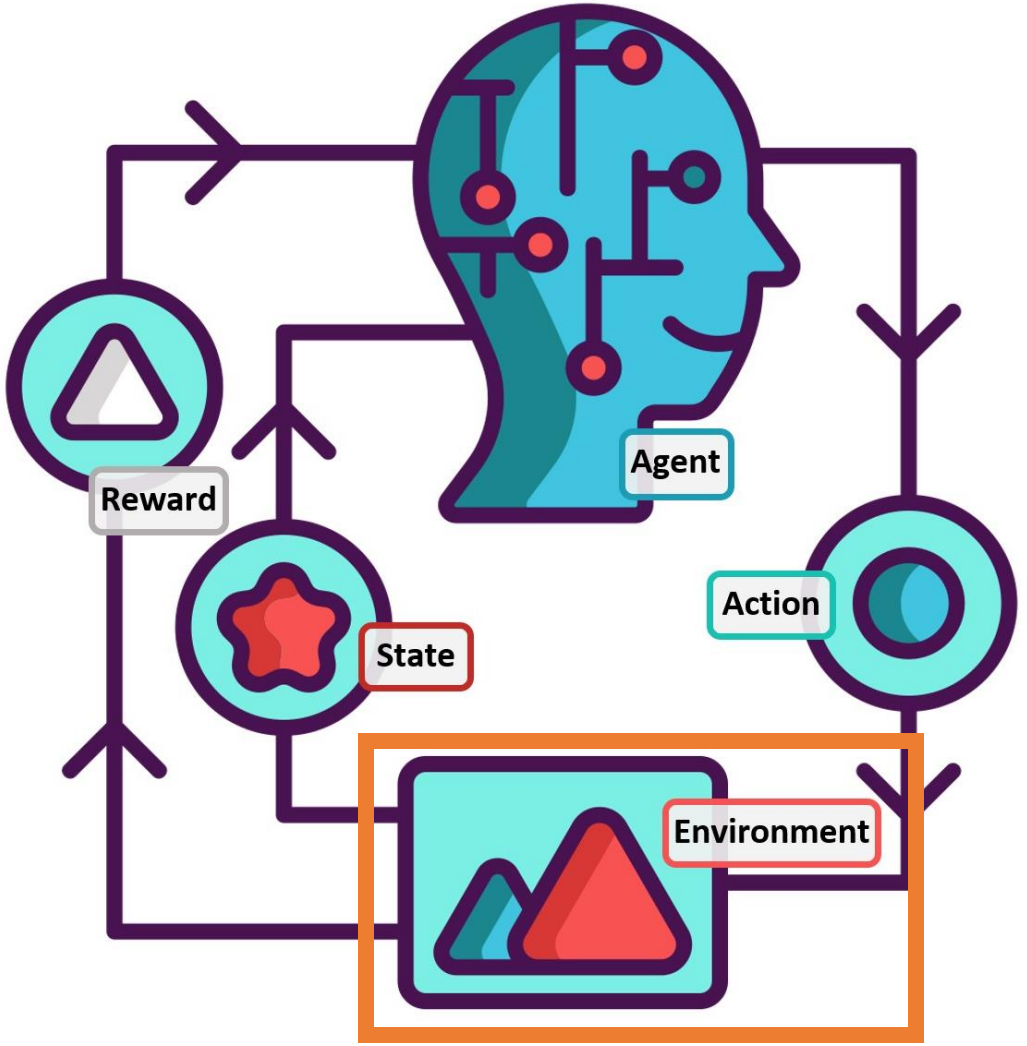
RL Formalism: The elements…

- An agent: the entity aiming at ‘solving a task’
- A set of states (exhaustive description of the system) in which the agent can be
- A set of actions (that could depend on the state) that can be taken by the agent



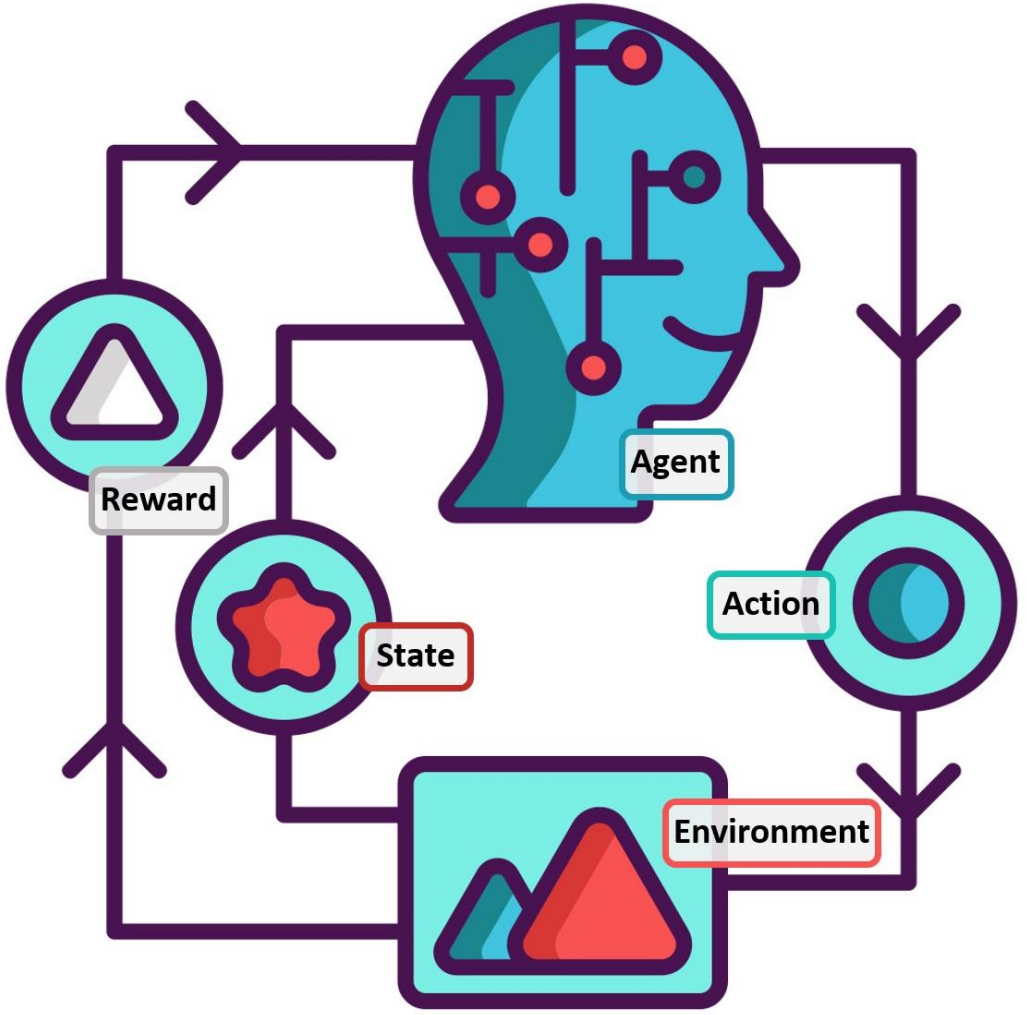
RL Formalism: The elements…

- An agent: the entity aiming at ‘solving a task’
- A set of states (exhaustive description of the system) in which the agent can be
- A set of actions (that could depend on the state) that can be taken by the agent
- An environment with which the agent interacts and that at each time *t* (i) could provide rewards R_t (ii) move the agent to a new state –depending on the state/action pair



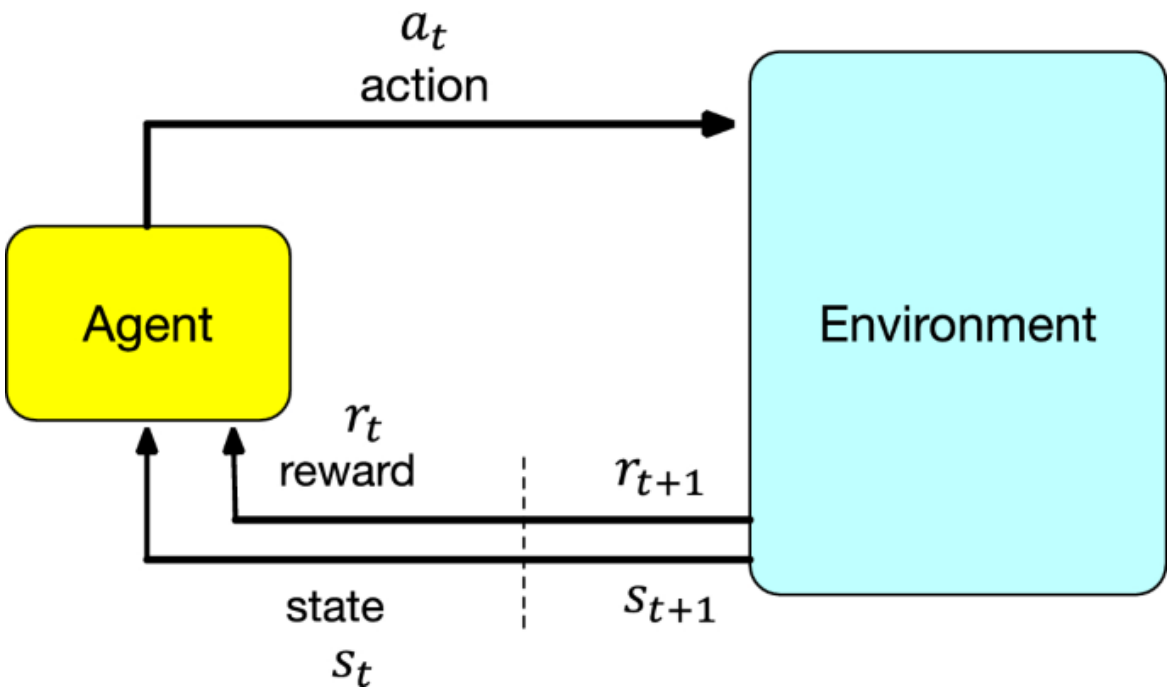
RL Formalism: The elements…

- An agent: the entity aiming at ‘solving a task’
- A set of states (exhaustive description of the system) in which the agent can be
- A set of actions (that could depend on the state) that can be taken by the agent
- An environment with which the agent interacts and that at each time *t* (i) could provide rewards R_t (ii) move the agent to a new state –depending on the state/action pair
- Agent goal is to maximize cumulative rewards over time (long-term goal)



RL Formalism: The elements…

- An agent: the entity aiming at ‘solving a task’
- A set of states (exhaustive description of the system) in which the agent can be
- A set of actions (that could depend on the state) that can be taken by the agent
- An environment with which the agent interacts and that at each time *t* (i) could provide rewards R_t (ii) move the agent to a new state –depending on the state/action pair
- Agent goal is to maximize cumulative rewards over time (long-term goal)



Rewards

- A reward R_t is a scalar feedback that indicates how well the agent is doing at step t
- With step t we indicate a discrete time-step: continuous-time treatment of RL can be done but it is outside of the scope of this course
- We consider a time step in:

1. Episodic tasks – last for a finite amount of time (ie. Games)
2. Continuous task – tasks that never ends (control of a datacenter)

Rewards

- A reward R_t is a scalar feedback that indicates how well the agent is doing at step t
- With step t we indicate a discrete time-step: continuous-time treatment of RL can be done but it is outside of the scope of this course
- We consider a time step in:

1. Episodic tasks – last for a finite amount of time (ie. Games)
2. Continuous task – tasks that never ends (control of a datacenter)

D. Silver 'Life is a continuous stream of data'

Rewards: examples

1. Autonomous agents (self-driving cars, drones, robots…)
   - +/- for/for not following desired trajectory
   - - for each time step taken for reaching a target position
   - ------ for crashing
2. Games
   - +/- win/lose
   - +/- A reward proportional to the score achieved
3. HVAC (Heating, Ventilating, Air Conditioning) energy optimization
   - - for the energy spent and/or for the user discomfort
4. Trading and Portfolio management
   - +/- proportional to € gained/lost
5. Online advertising & Recommendation systems (news, items, …)
   - +/- for ad/recommendation followed/ignored
6. Chatbot
   - +/- for conversation that the user find satisfying/not

Rewards

- The agent's job is to maximize cumulative reward (sum – or weighted sum of rewards that will be collected along the way):

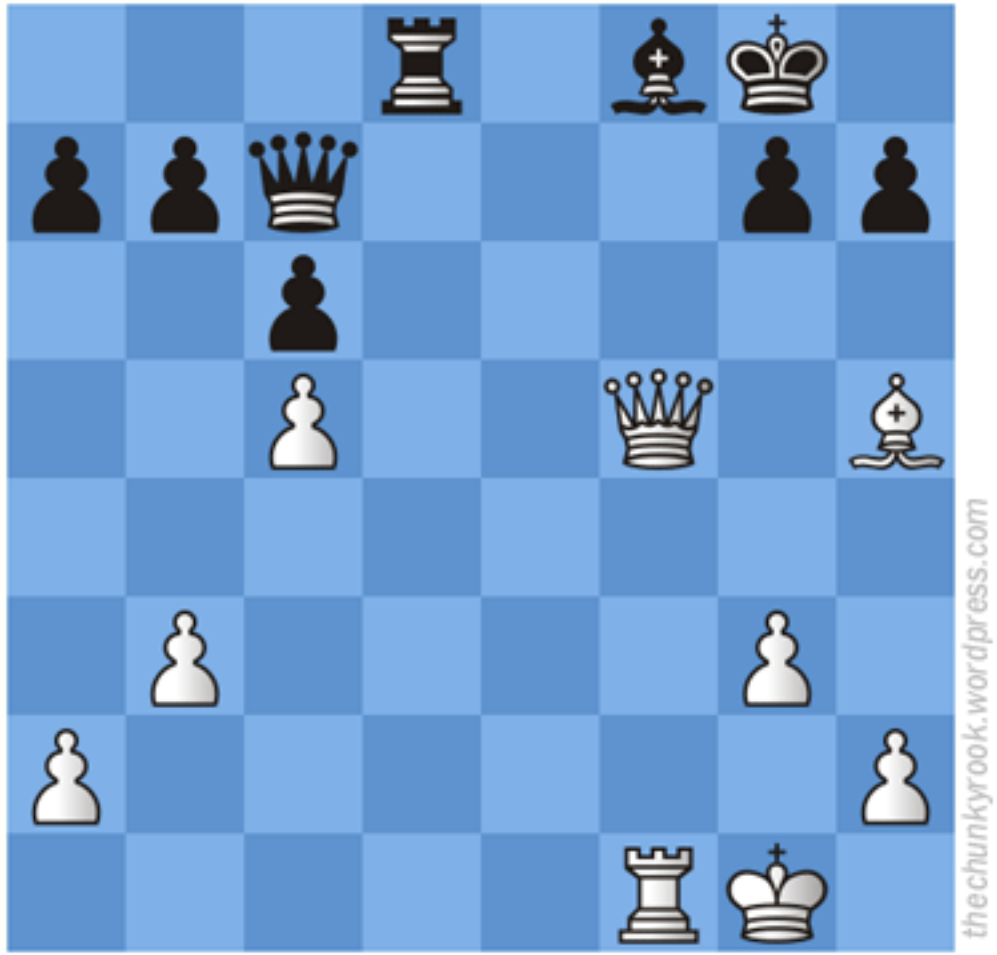
1. In episodic tasks, cumulative reward over an episode;
2. In continuous tasks, over a defined horizon (that can be rolling)

- RL is based on the reward hypothesis:

All goals can be described by the maximization of expected cumulative reward

Rewards: sequential decision making

- Agent's goal: select actions to maximize cumulative rewards/total future rewards
- Actions may have long-term consequences & rewards may be delayed
- Immediate rewards can be sacrificed to gain more long-term rewards: this is relevant both in the planning and in the 'training'



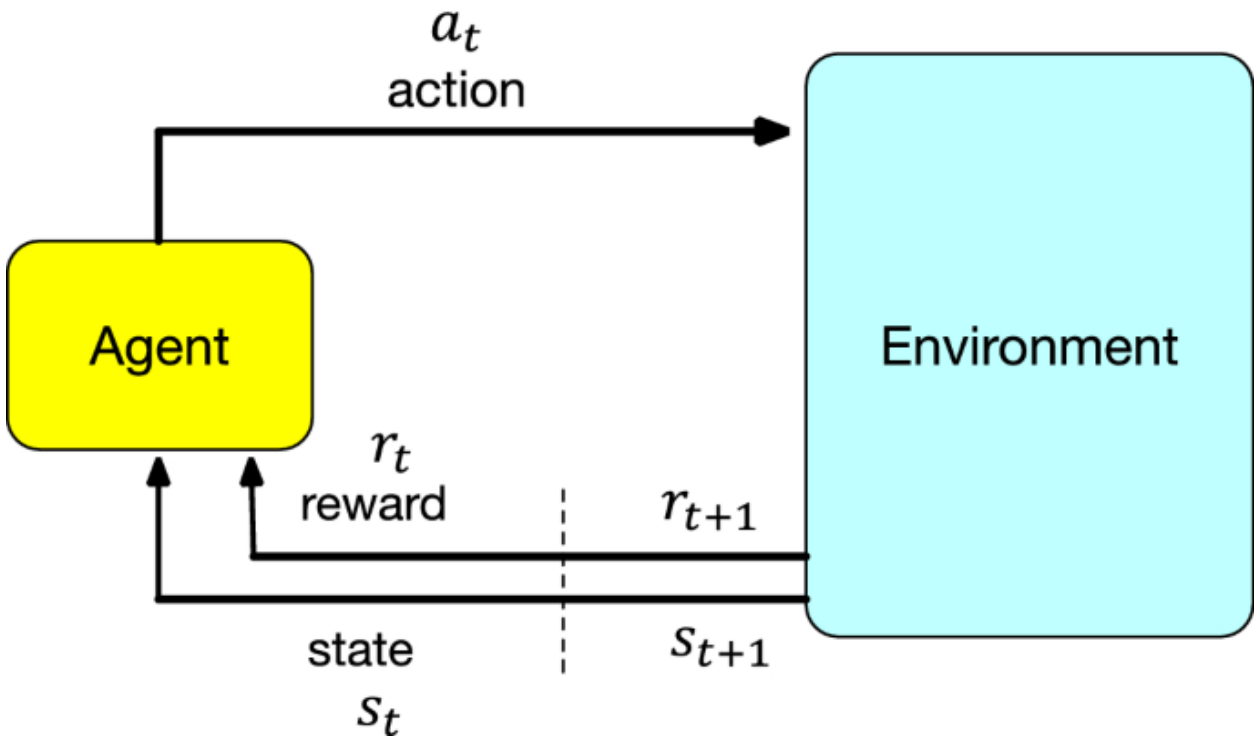
The Agent and the Environment

At each step t the agent (something that we can program/control):

- Is in the state S_t, an exhaustive description of the system (agent and environment) at time t
- Execute action A_t (that is a feasible action from state S_t)

The environment, based of the state/action pair (S_t, A_t):

- provides a reward R_{t+1}
- 'move' the agent in state S_{t+1}



RL

Episode #1:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Episode #2:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Episode #3:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

$$\vdots$$

Observations and History

Moreover, at each step the agent 'sees' an observation o_{t+1}:

- In fully observable environments $S_{t+1} = O_{t+1}$.
- In partially observable environments the state is not available, but must be inferred from the history H_t

$$H_t = O_1, R_1, A_1, \dots, A_{t-1}, O_t, R_t$$

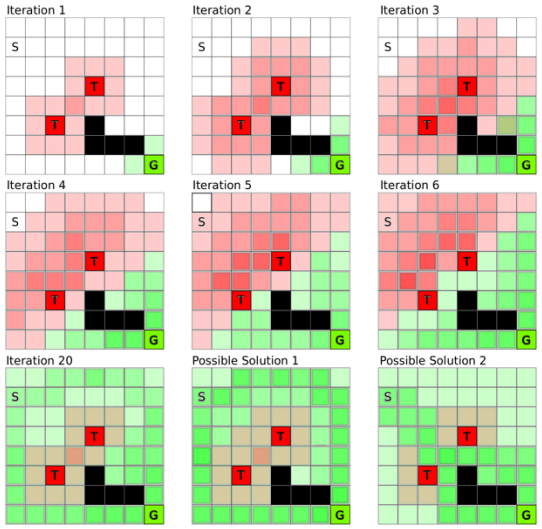
$$S_t = f(H_t)$$

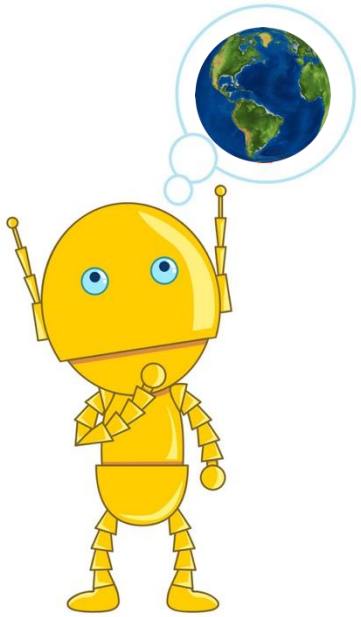
- If we can have a reliable 'state', agent's life is much easier, otherwise we must define different strategies: ex. Taking the all history as state or estimate the state with a supervised learning approximation

Components of an Agent

An agent may have the following components:

- (always) a policy, the agent's behaviour function
- (possibly) a value function, a quantitative description of how 'good' each state is
- (possibly) a q-value function, a quantitative description of how 'good' a (state, action) pair is
- (possibly) a model, the agent's representation of the environment

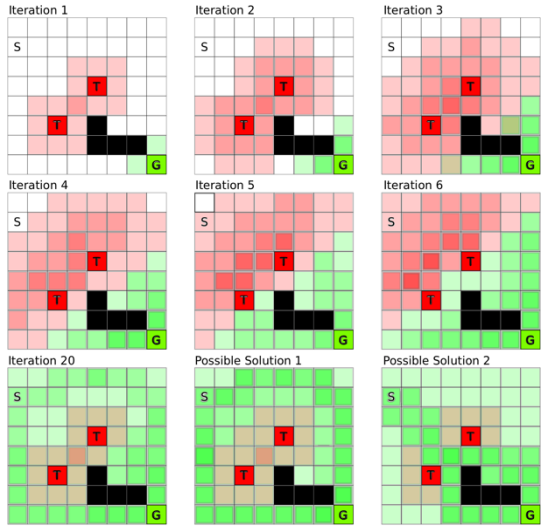


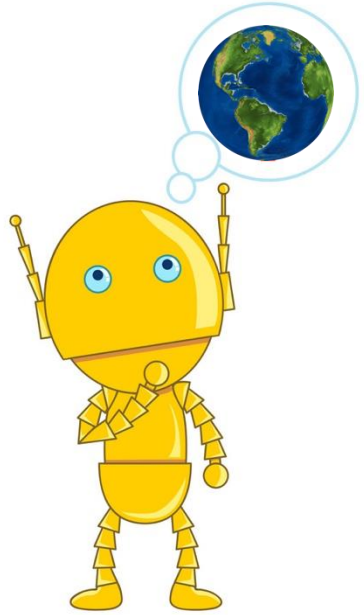


Components of an Agent

An agent may have the following components:

- (always) a policy, the agent's behaviour function
- (possibly) a value function, a quantitative description of how 'good' each state is
- (possibly) a q-value function, a quantitative description of how 'good' a (state, action) pair is
- (possibly) a model, the agent's representation of the environment





Components of an Agent: Policy

Policy: the agent's behaviour function, a map from state to action

The RL goal is to find the optimal policy (the one maximizing cumulative rewards)

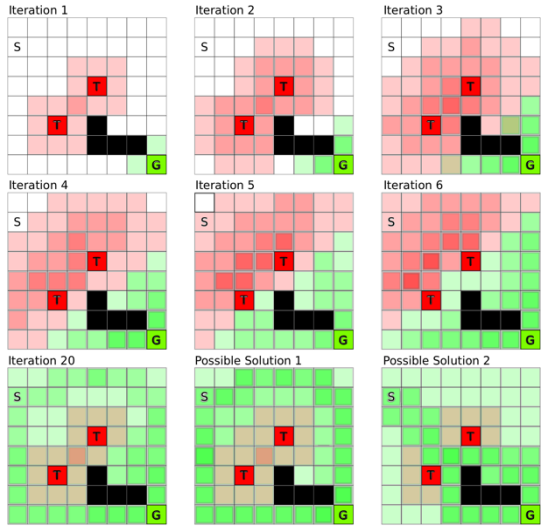
We can have:

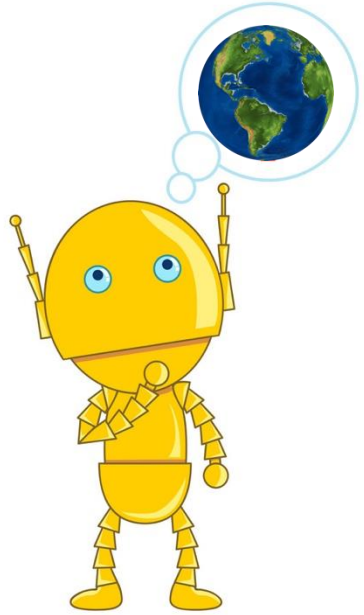
- Deterministic policies $a = \pi(s)$
- Stochastics policies $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$

Components of an Agent

An agent may have the following components:

- (always) a policy, the agent's behaviour function
- (possibly) a value function, a quantitative description of how 'good' each state is
- (possibly) a q-value function, a quantitative description of how 'good' a (state, action) pair is
- (possibly) a model, the agent's representation of the environment

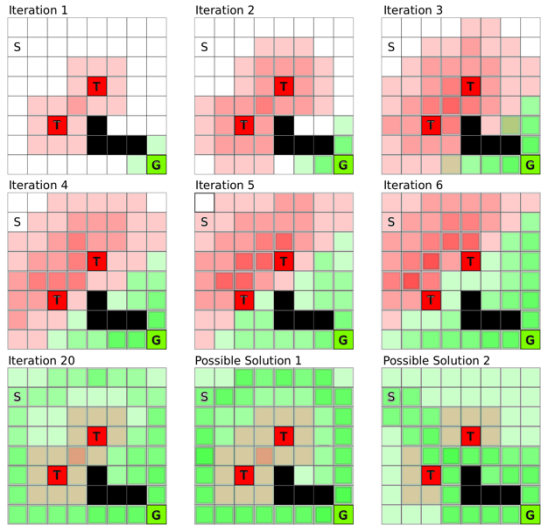


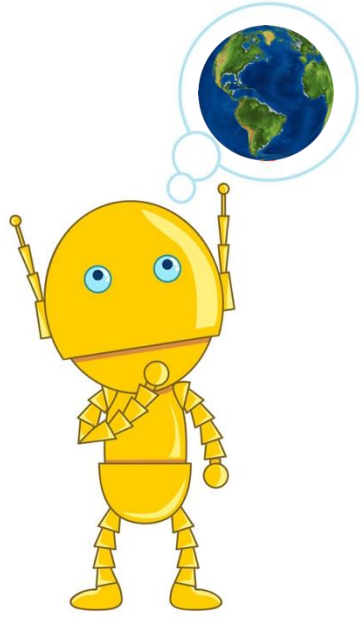


Components of an Agent

An agent may have the following components:

- (always) a policy, the agent's behaviour function
- (possibly) a value function, a quantitative description of how 'good' each state is
- (possibly) a q-value function, a quantitative description of how 'good' a (state, action) pair is
- (possibly) a model, the agent's representation of the environment



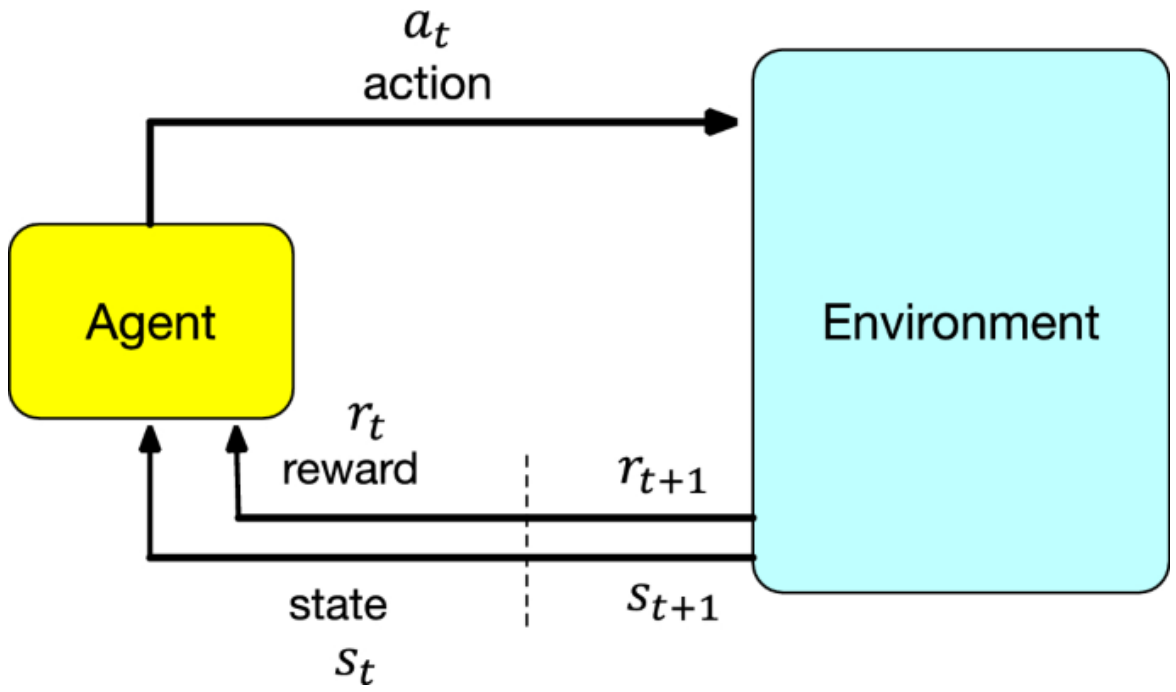


Peculiar and important quantities in RL: we will define *q* later, but they will become clearer in the next weeks

Multi-armed Bandits

The RL Elements

- An agent: the entity aiming at 'solving a task'
- A set of states S_t (exhaustive description of the system) in which the agent can be
- A set of actions A_t (that could depend on the state) that can be taken by the agent
- An environment with which the agent interacts and that at each time t (i) could provide rewards R_t (ii) move the agent to a new state –depending on the state/action pair
- Agent goal is to maximize cumulative rewards over time



Today we are considering 'simple' settings where the is only one state!
Or, we can think of it as the environment bringing back the agent to the same state $s_t = s_{t+1}$

RL vs. Multi-armed Bandits

Episode #1:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Episode #2:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

Episode #3:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$$

$$\vdots$$

Episode #1:

$$(S_0)\ A_1 R_1$$

Episode #2:

$$(S_0)\ A_2 R_2$$

Episode #3:

$$(S_0)\ A_3 R_3$$

$$\vdots$$

Episode #t:

$$(S_0)\ A_t R_t$$

Multi-armed Bandits (Chapter 2)

A simple ‘RL problem’, will allow us:

- To see some of the elements of a RL problem (rewards, time-stamps and values) in a simple setting
- To see a first example of decision making under uncertainty
- To see a first instance of the Exploration-Exploitation dilemma, a recurring topic in RL

Nevertheless, some differences to the usual RL settings are there, so be careful!

Chapter 2 will be exam material!

One-armed Bandit: aka a slot machine

‘One-armed’:

One lever, when pressed (after spending a coin) will provide money (a reward, that can be equal to zero)

‘Bandit’:

On the long-term, in the real world, the slot machine will steal money to an agent!

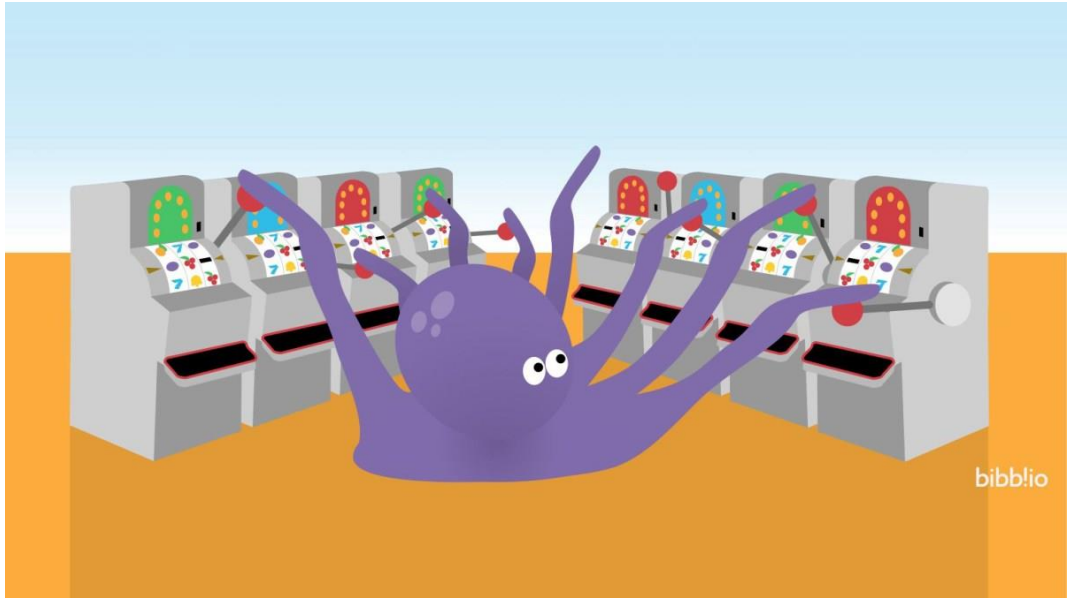
But let’s suppose that we have infinite/a lot of ‘casino coins’, and we want to change it into real money

Multi-armed Bandits

‘Multi-armed’ (or ‘k-armed’):

- Now we have many levers (k levers) associated with different machines
- Each machine is associated with a different distribution of rewards (one machine will pay more than the others)
- At each episode (we are dealing with an episodic task), the agent can pull only one lever
- What is the best strategy to

maximize cumulative (over different episodes) rewards?



Why do we care about k-armed Bandits problems?

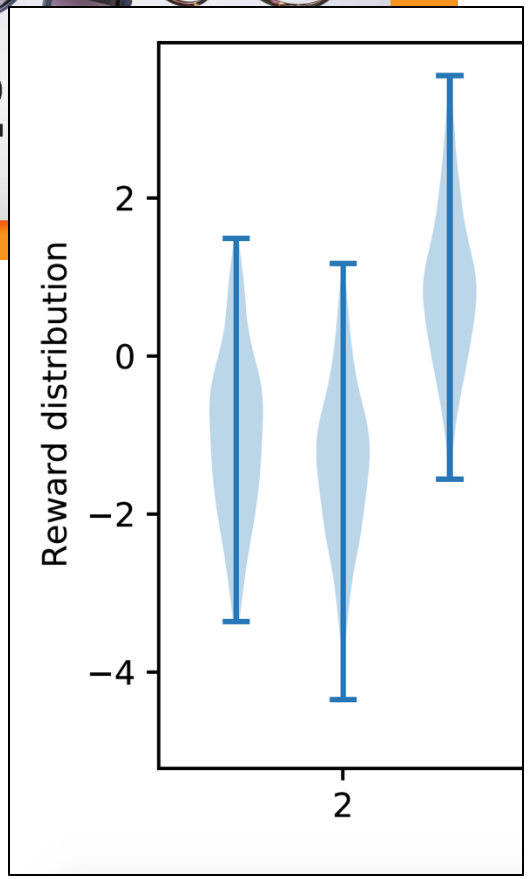
- On-line advertising/content suggestion: which ad/product/suggestion will be shown to users?
- Medical treatments: which medicine will be given to patients?
- Network server selection: which job has to be processed in one of several servers? (used in DNS server selection & cloud computing)
- …

Why do we care about k-armed Bandits problems?

- On-line advertising/content suggestion: which ad/product/suggestion will be shown to users?
- Medical treatments: which medicine will be given to patients?
- Network server selection: which job has to be processed in one of several servers? (used in DNS server selection & cloud computing)
- …

These ‘actions’ (ie. choosing an ad or another one) are associated with stochastic rewards!

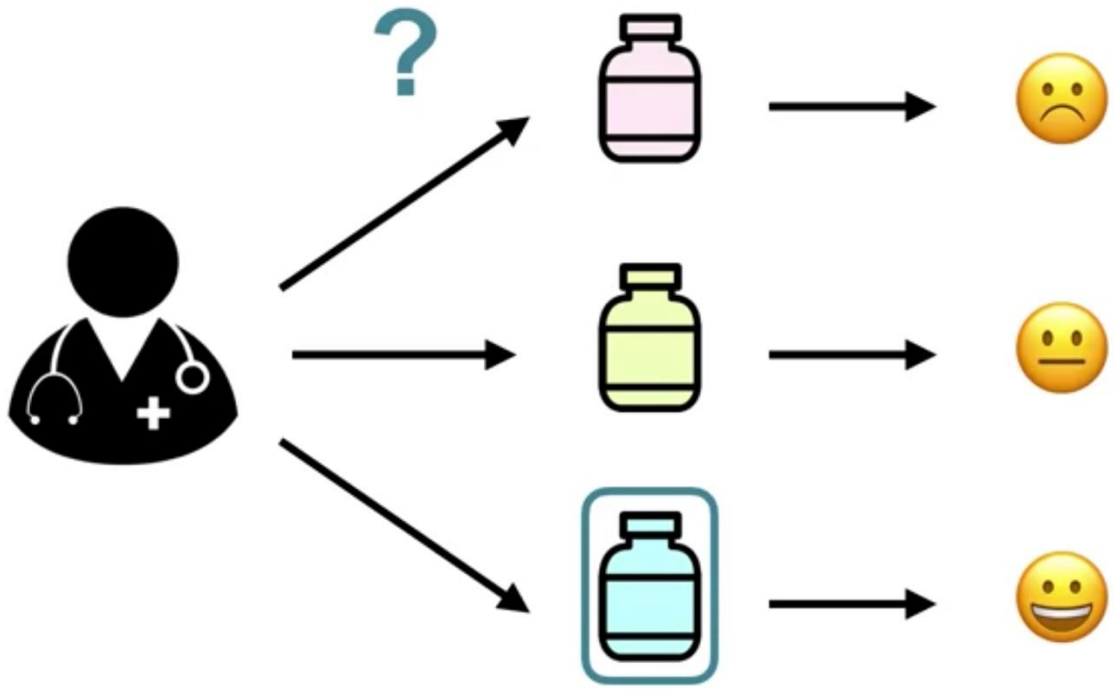
Ie. Different people will have different reactions, maybe in different moment of the day/week… there could be a distribution of possible rewards associated!



Example: clinical trials

A doctor has to find the most effective treatment.

Each patient is an episode: the doctor give a different treatment (different action) and see the effect of the treatment (reward)

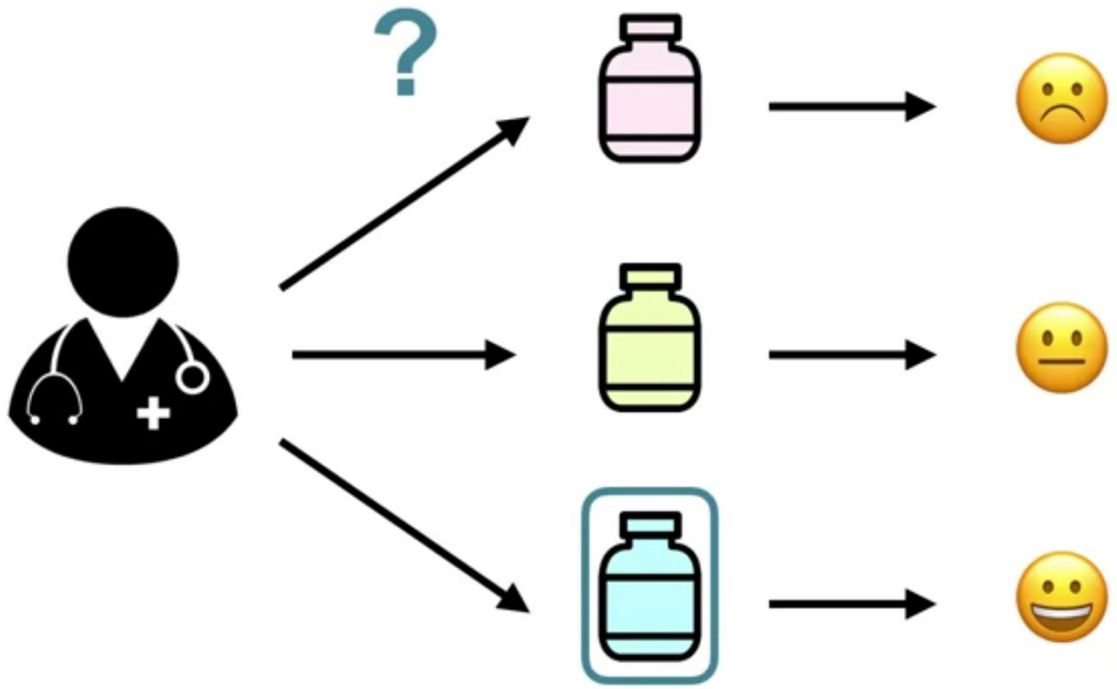


A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials

In this simple example, after 3 episodes/patients, a treatment looks more effective than another one.

What the doctor will do for the next patient (episode 4)?

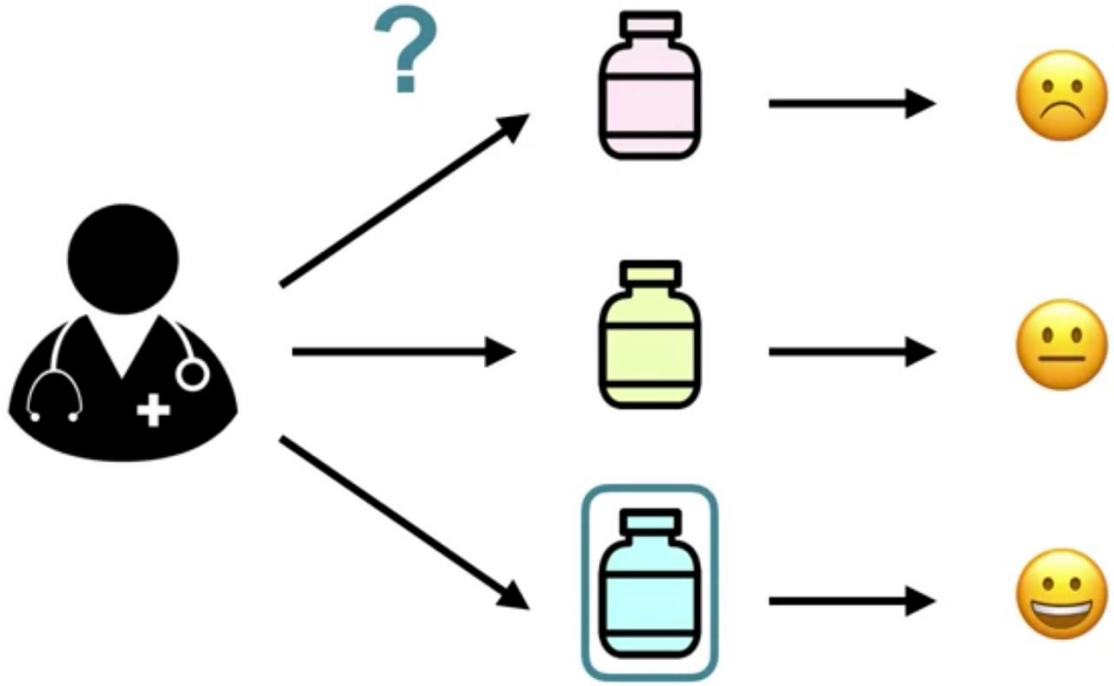


A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials

We have the first instance of the exploration-exploitation dilemma:

- We have a ‘best’ action and we want to maximize the treatments effects
- We don’t know if the action is optimal (there may be uncertainty on the effect of a treatment): there may be other treatments, but also the ones that we have already seen may provide different rewards with different patients



A. White, M. White ‘Fundamentals of Reinforcement Learning’

k-armed Bandit: Action Values

In the k-armed bandit problem:

- The agent has *k* different actions
- We have only one state, where all of the possible *k* actions can be taken
- Each action provides a stochastic reward (the reward follows a probability distribution)
- We can describe the q-value of an action:

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

- Note that here *t* indicates the episode number (that is typically indicated with *k*), while in typical RL settings *t* will indicate the discrete time-stamp within an episode

k-armed Bandit: Action Values

How to estimate this after t episodes?

In the k-armed bandit problem:

- The agent has *k* different actions
- We have only one state, where all of the possible *k* actions can be taken
- Each action provides a stochastic reward (the reward follows a probability distribution)
- We can describe the q-value of an action:

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

- Note that here *t* indicates the episode number (that is typically indicated with *k*), while in typical RL settings *t* will indicate the discrete time-stamp within an episode

k-armed Bandit: Action Values

- The q-value for an action is not know and must be estimated
- A simple way is by the Sample-Average method

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

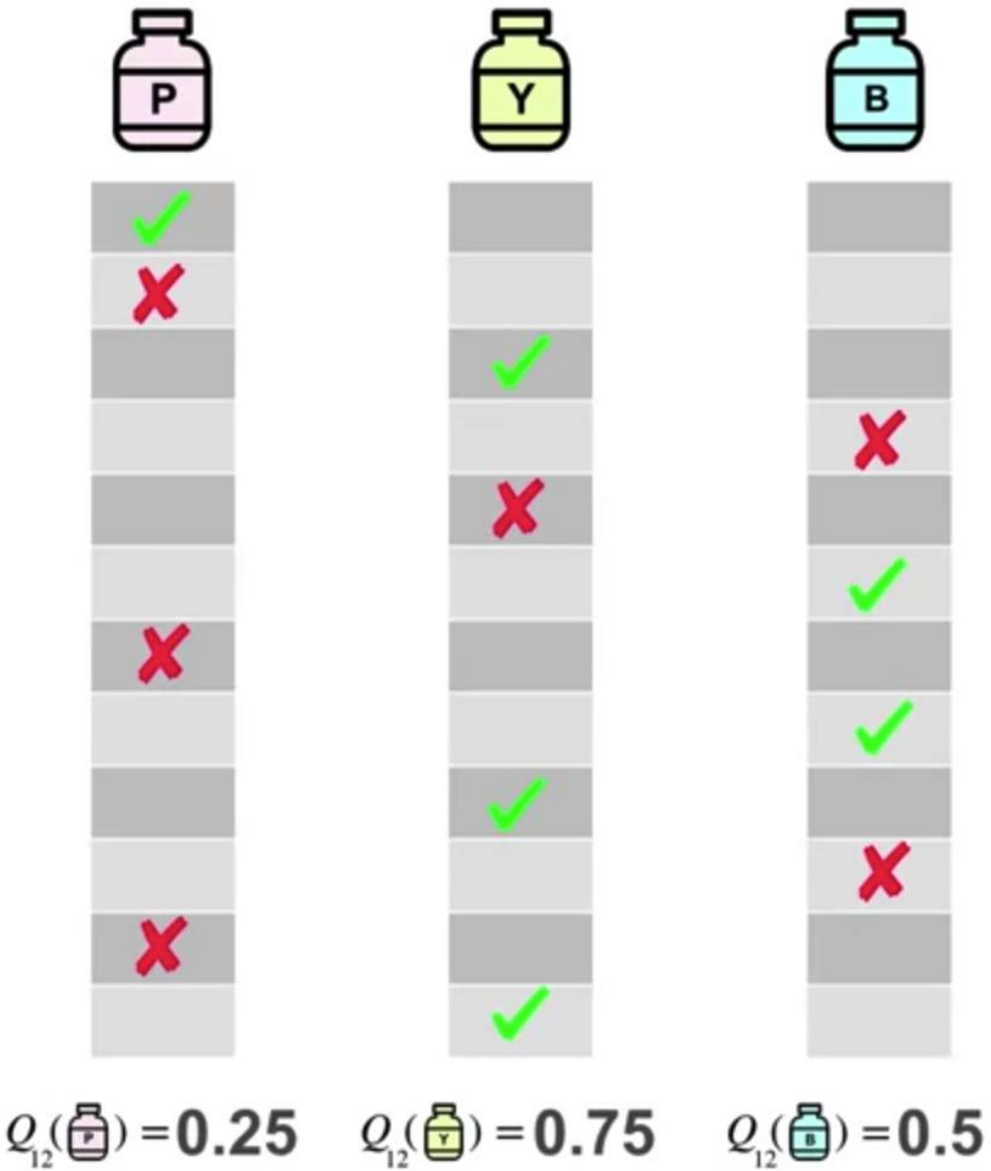
where $\mathbb{1}_{predicate}$ is equals to 1 if *predicate* is true and 0 if it is not.

- In other term is the average rewards that we have seen when applying action *a*

I can recompute this estimation every time I collect a new data

Example: clinical trials

- Let’s consider the case where a treatment provides +1 reward if it is effective (0 otherwise)
- We start with 0 as initial estimations of the action values
- A random policy is applied by the doctor and, with elevated numbers of epochs, the estimate action value will converge to the true mean value (the true reward we can expect for action *a*)



A. White, M. White ‘Fundamentals of Reinforcement Learning’

Greedy Policy

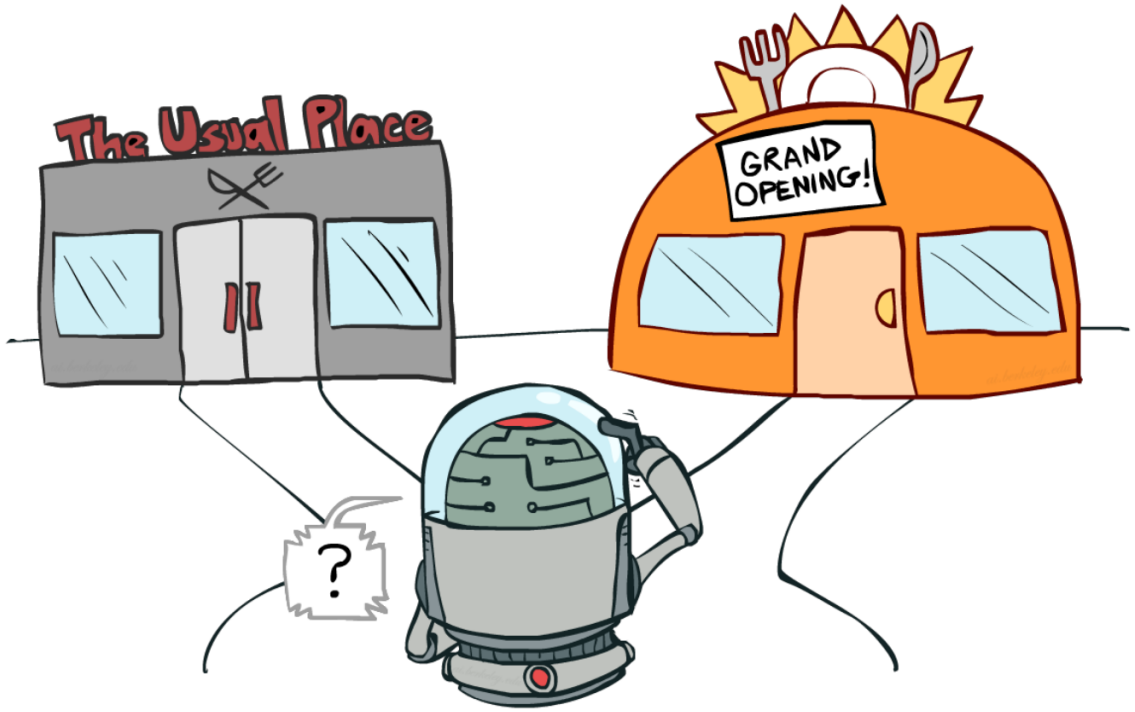
- With the policy the selected action is:

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

- This is the simplest policy, which is reasonable, but leads to and exploration problem: we may be stacked with a suboptimal action!

Exploration vs. Exploitation

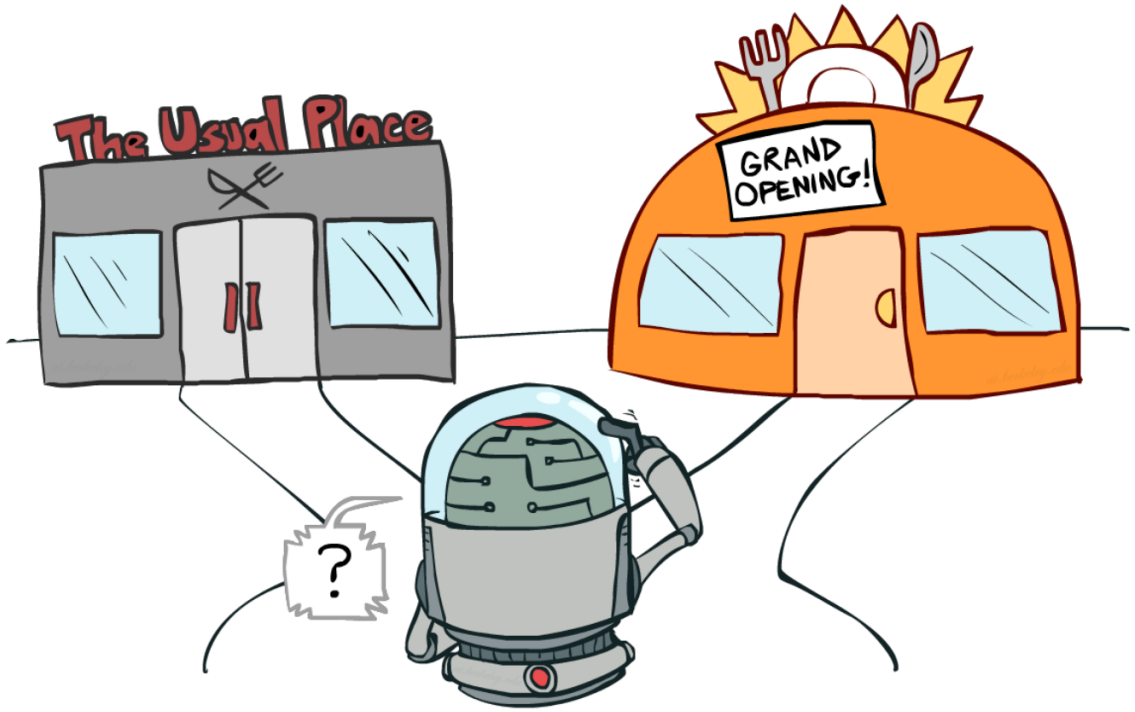
- Exploration allows us to improve knowledge for long-term benefit
- Exploitation exploit knowledge for short-term benefit
- We cannot do both exploration and exploitation at the same time: implementing trade-offs is necessary!



Exploration vs. Exploitation

Today we will see different approaches to cope with the trade-off:

1. ε-greedy
2. Greedy with optimistic inizialization
3. Upper Confidence Bound (UCB)
4. Gradient Bandit



#1 $\mathcal{E}$-Greedy Policy

- A simple approach to deal with exploration is ε-Greedy Policy, that is a stochastic policy:

$$A_t \overset{\text{def}}{=} \begin{cases} \text{greedy action with probability } 1 - \varepsilon \\ \text{non greedy action with probability } \varepsilon \end{cases}$$

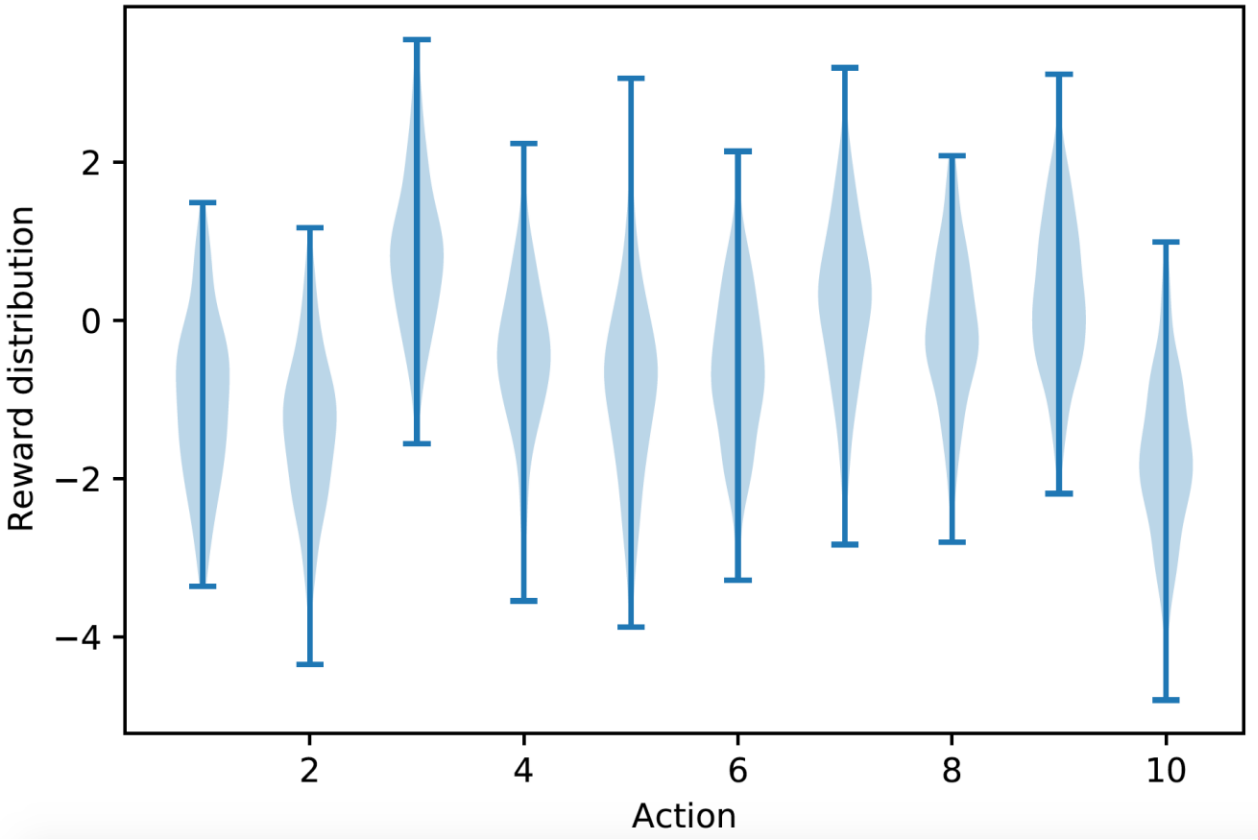
with $\varepsilon \in (0,1)$ (typically a small number)

- The non greedy actions are chosen with uniform probabilty

Example: 10-armed bandits

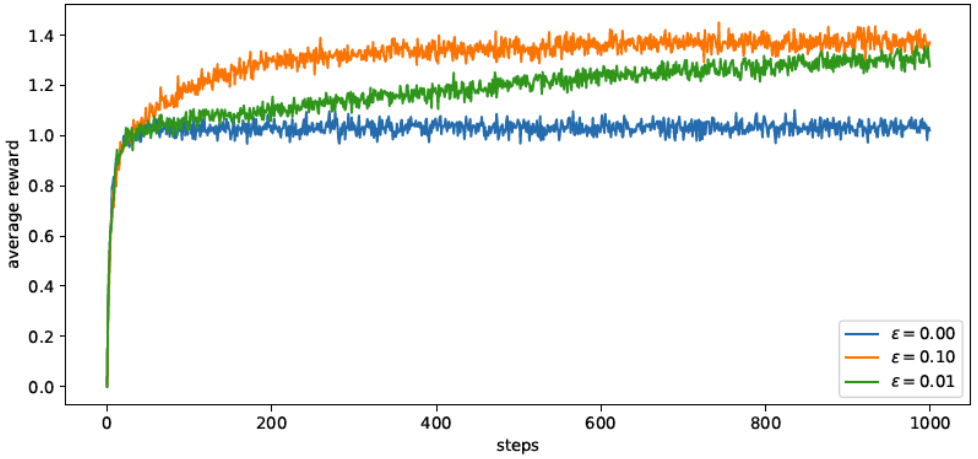
For this, and for other examples from the book, a Python implementation (by Shangtong Zhang) is available at https://github.com/ShangtongZhang/reinforcement-learning-an-introduction

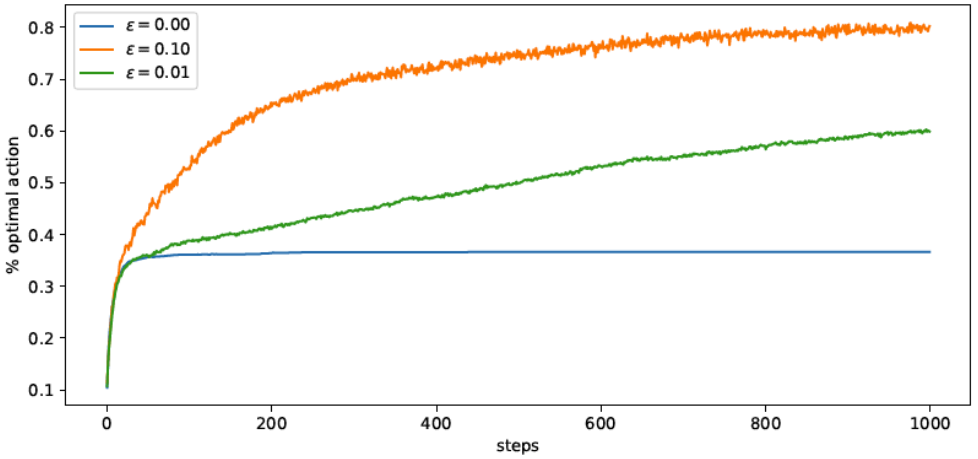
Clone/fork the repo! (also available on the Moodle page)



Example: 10-armed bandits

- 2000 randomly generated 10-armed bandit problem
- For each bandit problem, each actions values is drawn from a Gaussian distribution $\mathcal{N}(\mu_k, 1)$ with μ_k that is drawn from another distribution with $\mathcal{N}(0,1)$
- We measure the performance of different policies as an <u>average over the 2000 problems</u>

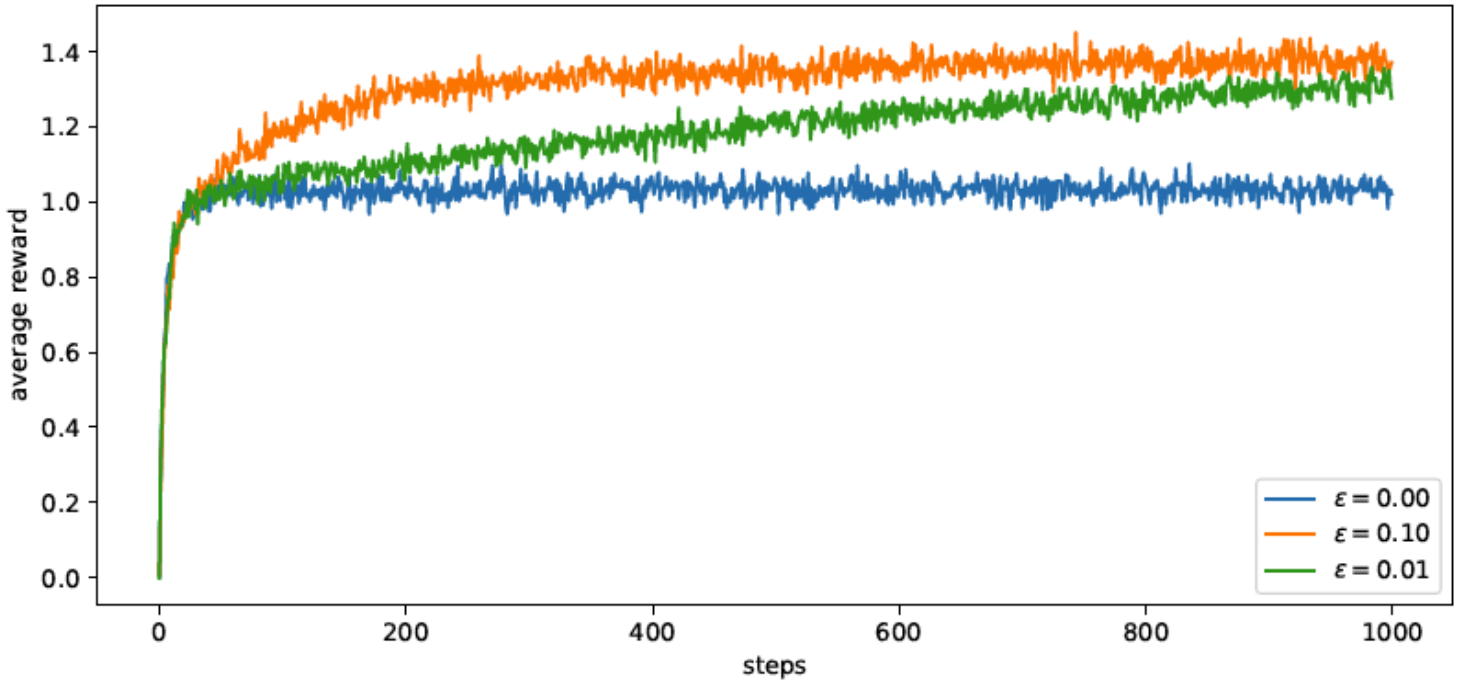


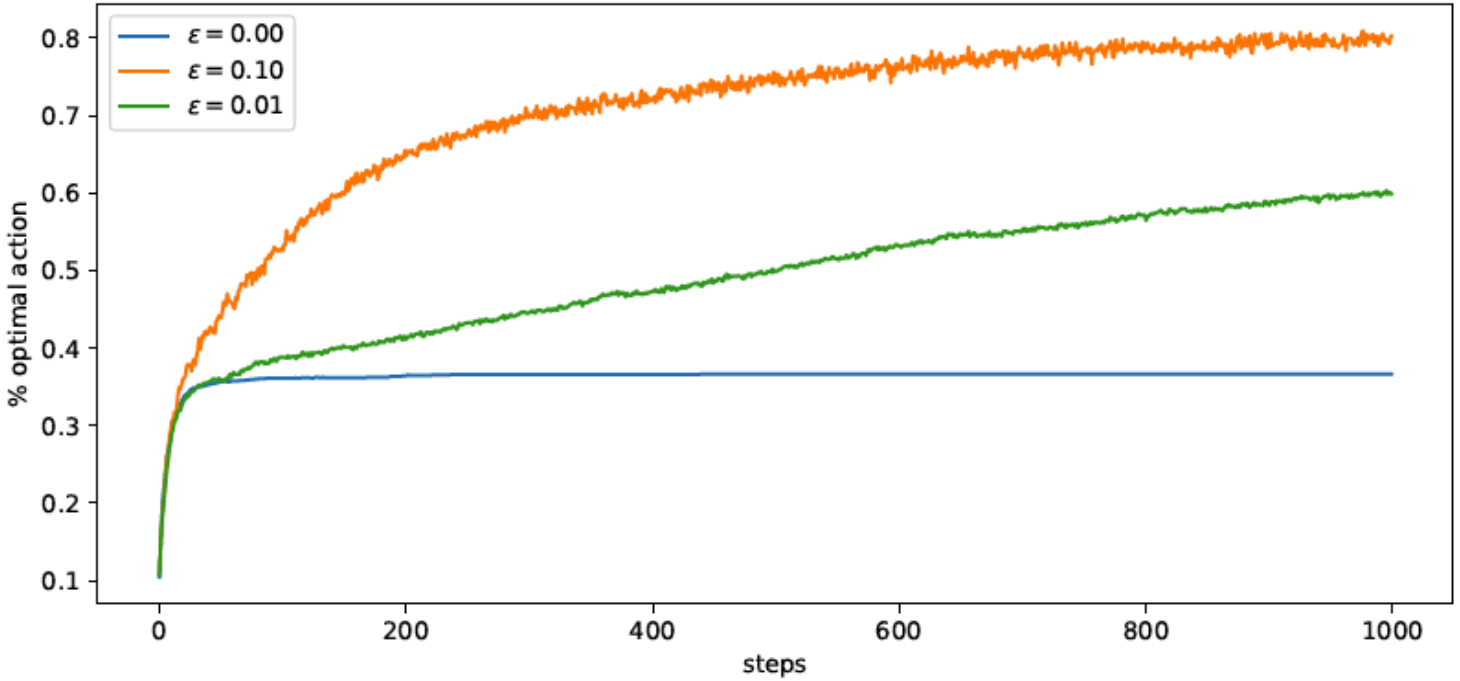


Reward distribution <u>for one of 2000</u> 10-armed bandit problem!

Example: 10-armed bandits

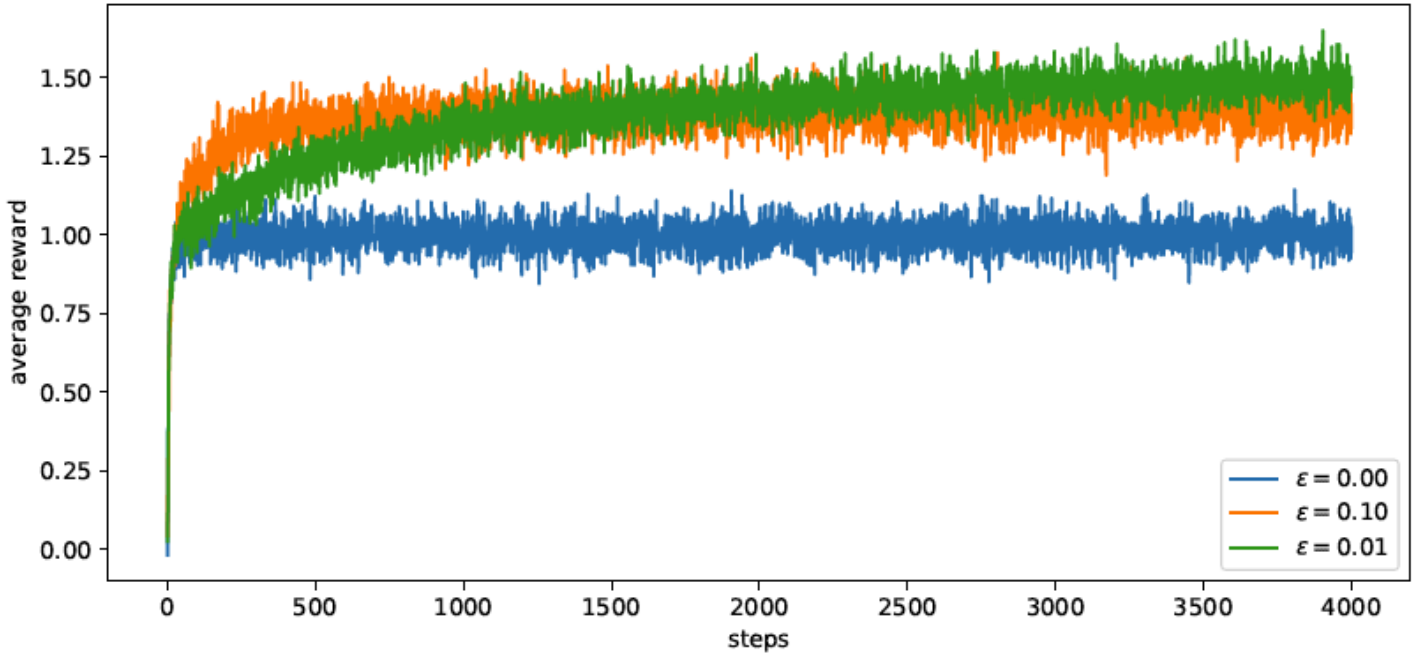
- The action value function is computed the sample-average method
- Depending on the value of Ɛ performance can vary a lot: optimal value of Ɛ depends on the problem
- For example, if variance was bigger, more exploration would have been useful

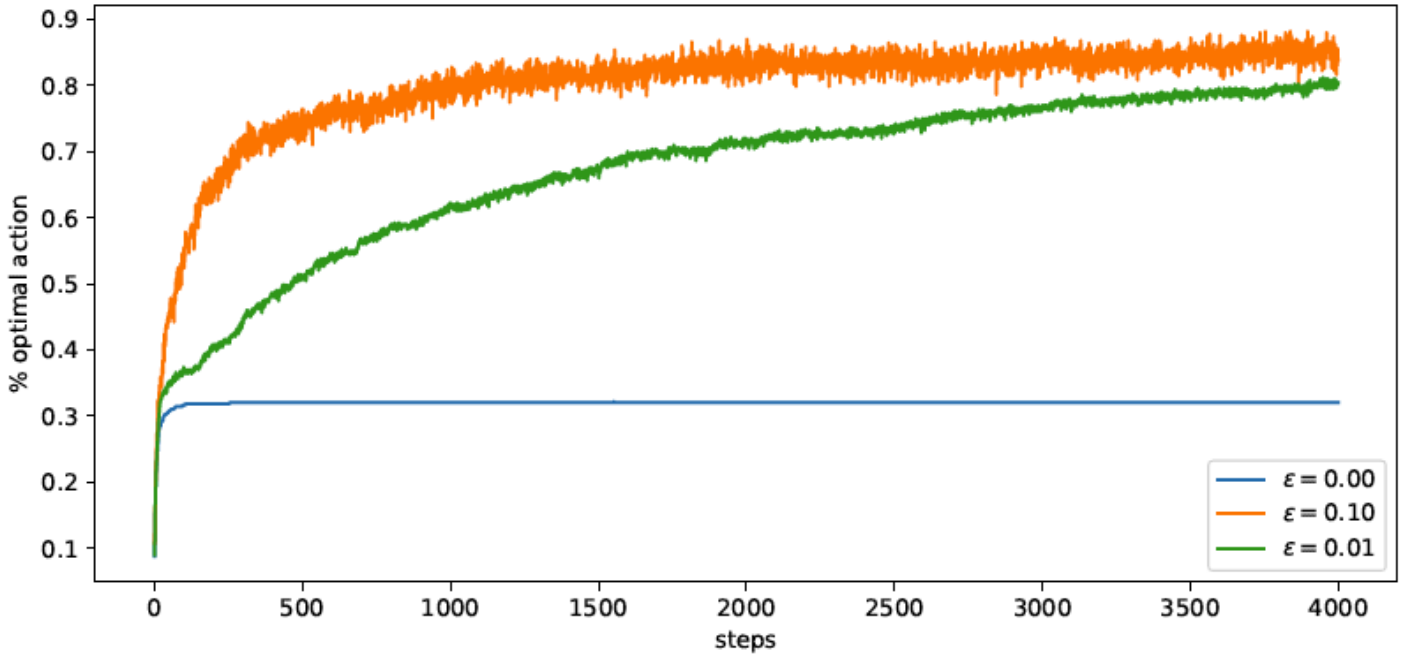




Example: 10-armed bandits

- If we run it for more time, we see that the $\mathcal{E} = 0.01$ policy, while still inferior to the one with $\mathcal{E} = 0.1$, after 2000 steps is giving higher average rewards
- <u>These plots are averaged over 500 10-armed bandits</u>





Incremental Implementation of Action Value Estimate

- Let's consider a single action *a'*
- Let R_i be the reward received after the *i*-th selection of *a'*
- After *n-1* times selections of *a'* we can estimate its action value as

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}$$

- This approach is not efficient: memory and computation would grow over time

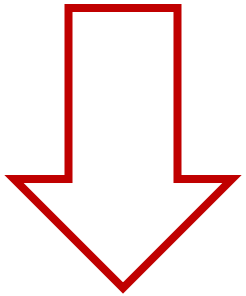
Incremental Implementation of Action Value Estimate

- This implementation requires only Q_n and n to be stored

$$
\begin{aligned}
Q_{n+1} &= \frac{1}{n}\sum_{i=1}^{n} R_i \\
&= \frac{1}{n}\left(R_n + \sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\left(R_n + (n-1)\frac{1}{n-1}\sum_{i=1}^{n-1} R_i\right) \\
&= \frac{1}{n}\Big(R_n + (n-1)Q_n\Big) \\
&= \frac{1}{n}\Big(R_n + nQ_n - Q_n\Big) \\
&= Q_n + \frac{1}{n}\Big[R_n - Q_n\Big],
\end{aligned}
$$

Incremental Estimations are commonly used in Reinforcement Learning

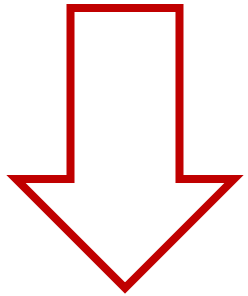
$$Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right]$$



$$NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right]$$

Incremental Estimations are commonly used in Reinforcement Learning

$$Q_{n+1} = Q_n + \frac{1}{n}\left[R_n - Q_n\right]$$



$$NewEstimate \leftarrow OldEstimate + StepSize\left[Target - OldEstimate\right]$$

Typically, we will employ fixed step-size $\alpha \in (0,1]$

Incremental Estimations are commonly used in Reinforcement Learning

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2\alpha R_{n-2} + \\
&\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i .
\end{aligned}
$$

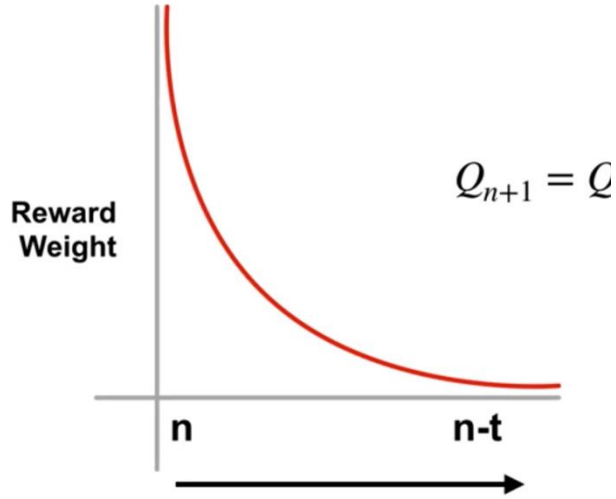
Incremental Estimations are commonly used in Reinforcement Learning

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\
&\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i .
\end{aligned}
$$

Exponentially weighted moving average (*exponential recency-weighted moving average*)

Incremental Estimations are commonly used in Reinforcement Learning

$$
\begin{aligned}
Q_{n+1} &= Q_n + \alpha\Big[R_n - Q_n\Big] \\
&= \alpha R_n + (1-\alpha)Q_n \\
&= \alpha R_n + (1-\alpha)\left[\alpha R_{n-1} + (1-\alpha)Q_{n-1}\right] \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 Q_{n-1} \\
&= \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \\
&\qquad \cdots + (1-\alpha)^{n-1}\alpha R_1 + (1-\alpha)^n Q_1 \\
&= (1-\alpha)^n Q_1 + \sum_{i=1}^{n} \alpha(1-\alpha)^{n-i} R_i .
\end{aligned}
$$

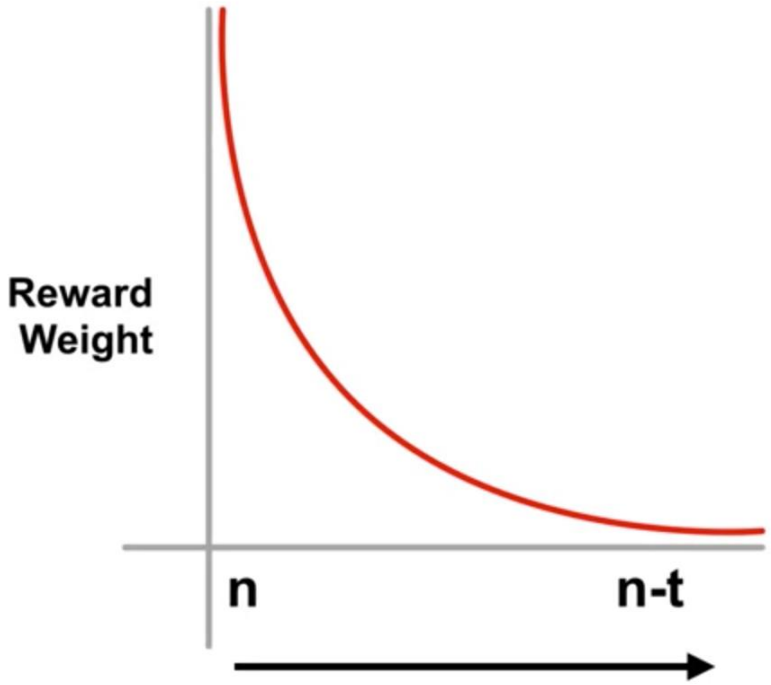


Exponentially weighted moving average (exponential recency-weighted moving average)Typically, we will employ fixed step-size $\alpha \in (0,1]$

Incremental Estimations for dealing with nonstationary problems

- This choice has the advantage of forgetting over time old rewards
- This is extremely convenient in nonstationary problems: problems were the reward distributions of the actions change over time
- Other solutions, with time-varying step size, are possible (see Section 2.5)

$$Q_{n+1} = \alpha R_n + (1-\alpha)\alpha R_{n-1} + (1-\alpha)^2 \alpha R_{n-2} + \cdots + (1-\alpha)^{n-1} \alpha R_1 + (1-\alpha)^n Q_1$$



A. White, M. White 'Fundamentals of Reinforcement Learning'

#2 - Optimistic Initial Values

There are other ways to encourage exploration…

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Loop forever:

$$A \leftarrow \begin{cases} \text{argmax}_a\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

#2 - Optimistic Initial Values

There are other ways to encourage exploration…

A simple bandit algorithm

Initialize, for $a = 1$ to k:

$Q(a) \leftarrow 0$

$N(a) \leftarrow 0$

Optimism in the face of uncertainty: turns out that choosing optimistic initial values for $Q(a)$ can help exploration!

Loop forever:

$$A \leftarrow \begin{cases} \text{argmax}_a\, Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$$

$R \leftarrow bandit(A)$

$N(A) \leftarrow N(A) + 1$

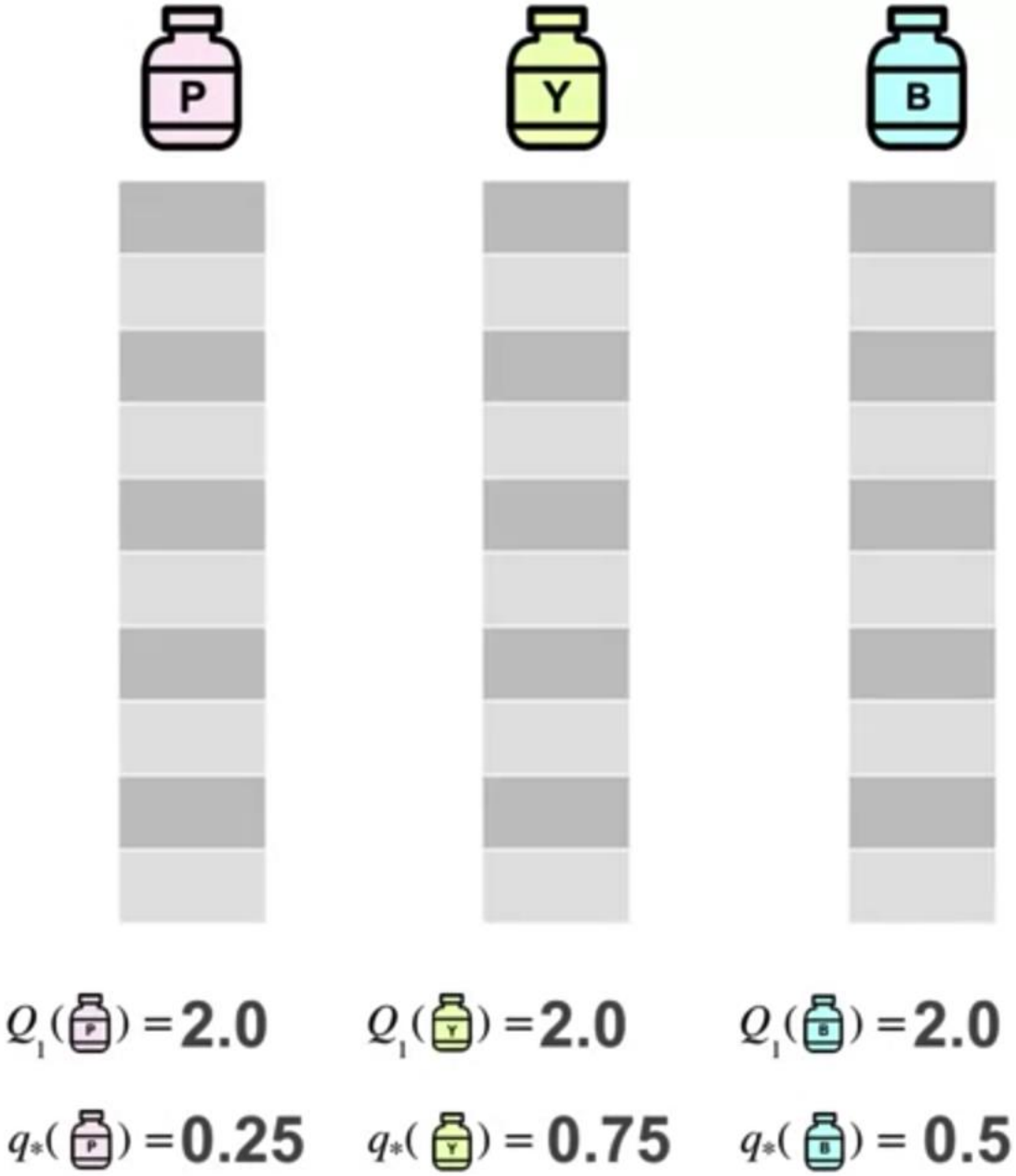
$Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$

Example: clinical trials (reprise)

- A treatment provides +1 reward if it is effective (0 otherwise)
- This time we choose a clearly optimistic choice for the action values, being equal to 2
- We choose a greedy policy with:

$$Q_{n+1} \leftarrow Q_n + \alpha\left(R_n - Q_n\right)$$

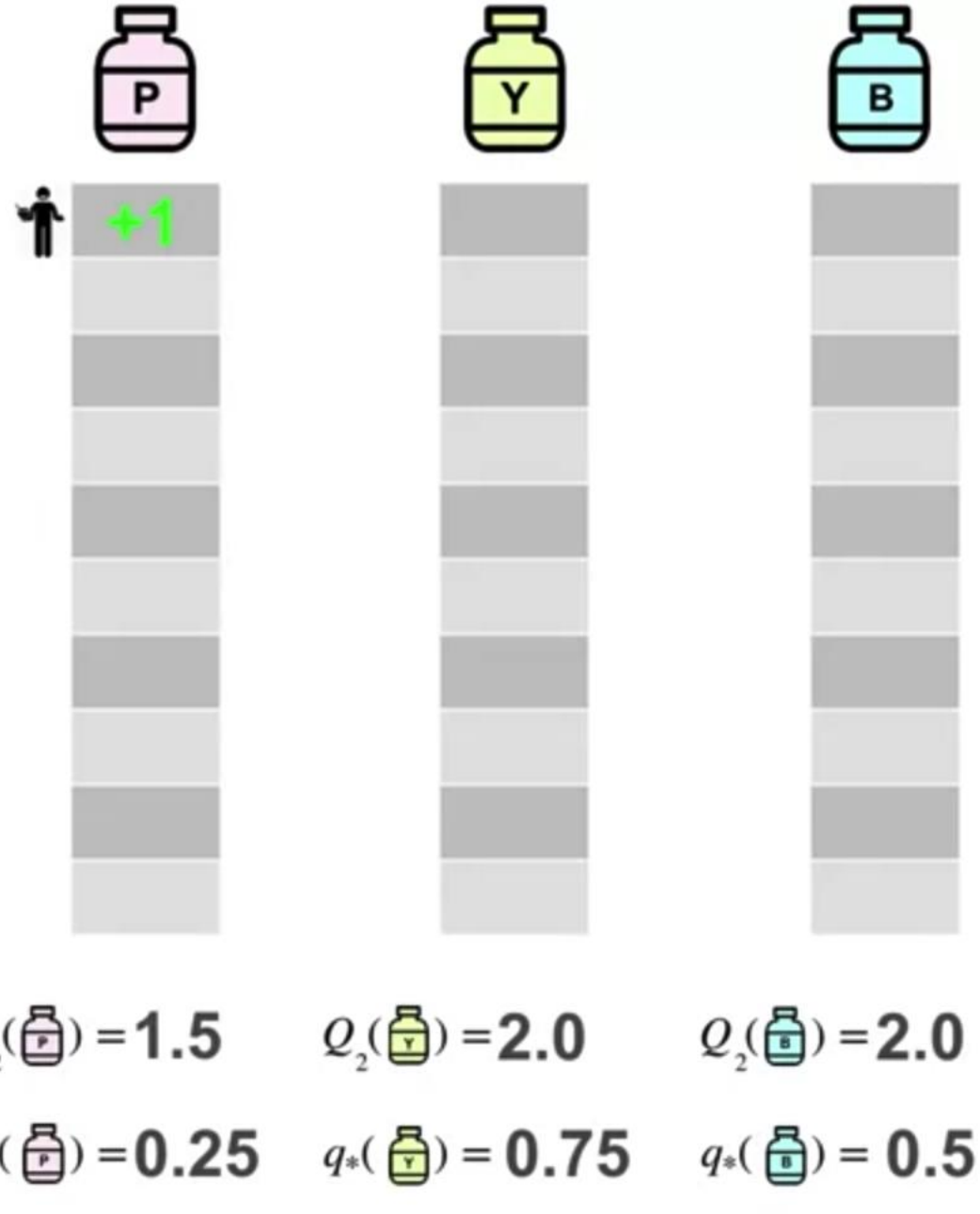
$$\textbf{Let}\ \alpha = 0.5$$



A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials (reprise)

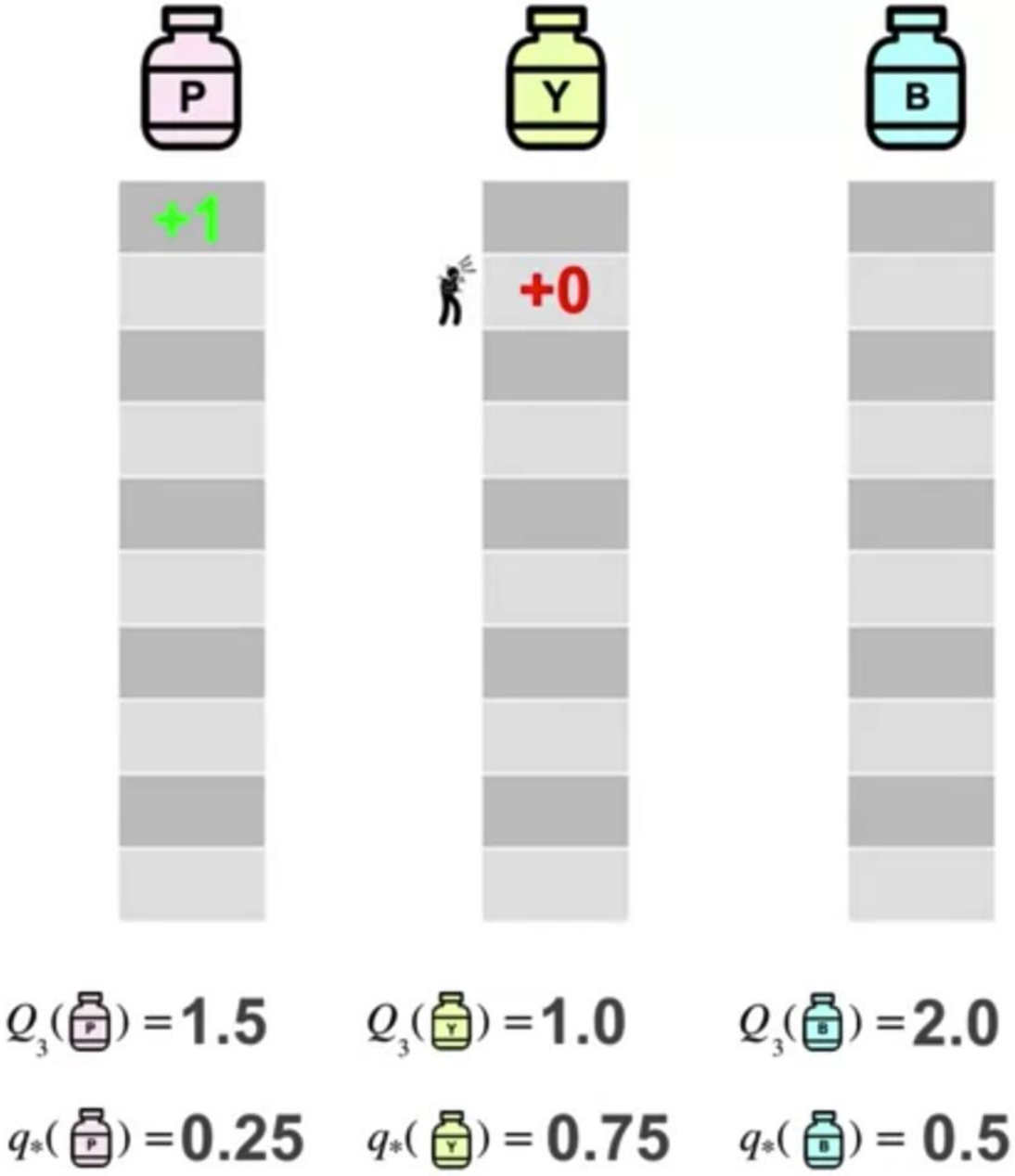
Epoch #1:



A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials (reprise)

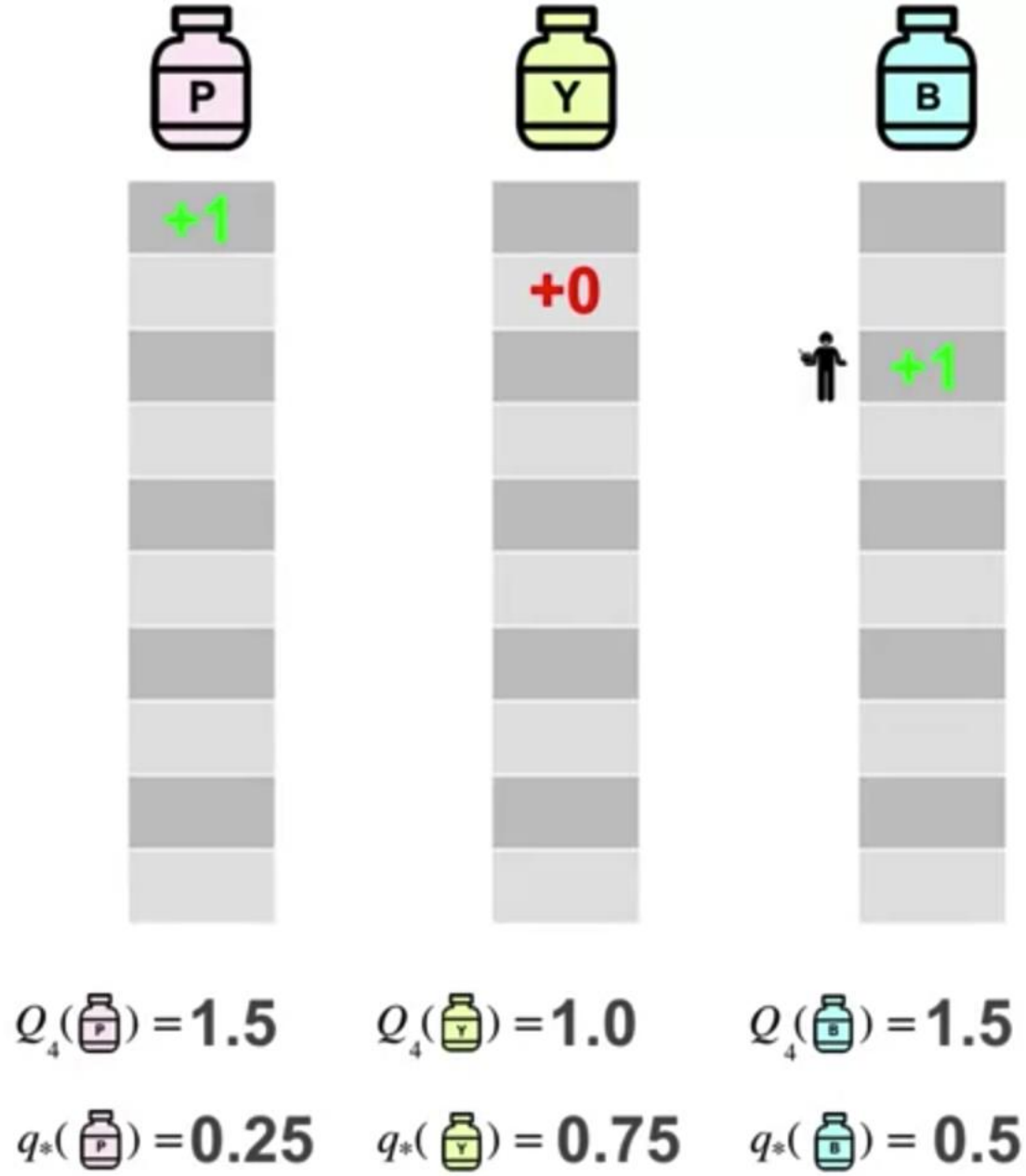
Epoch #2:



A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials (reprise)

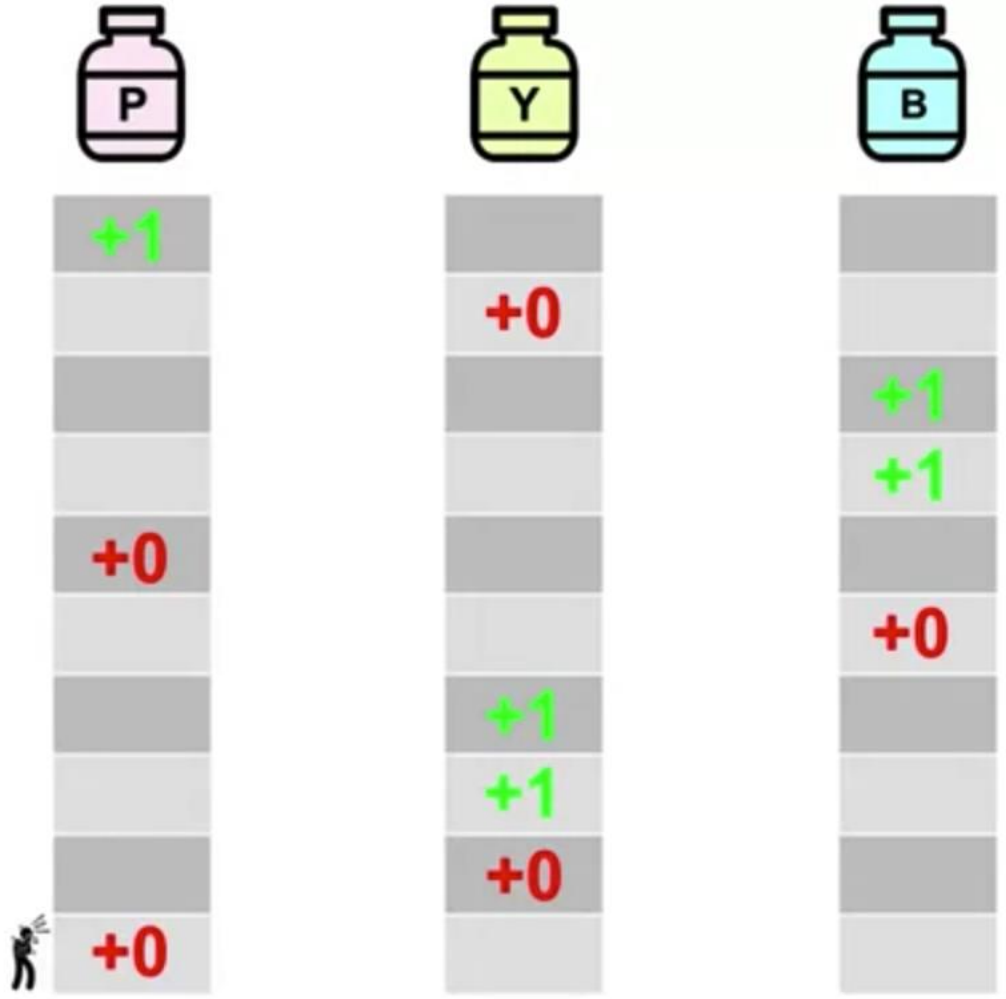
Epoch #3:



A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials (reprise)

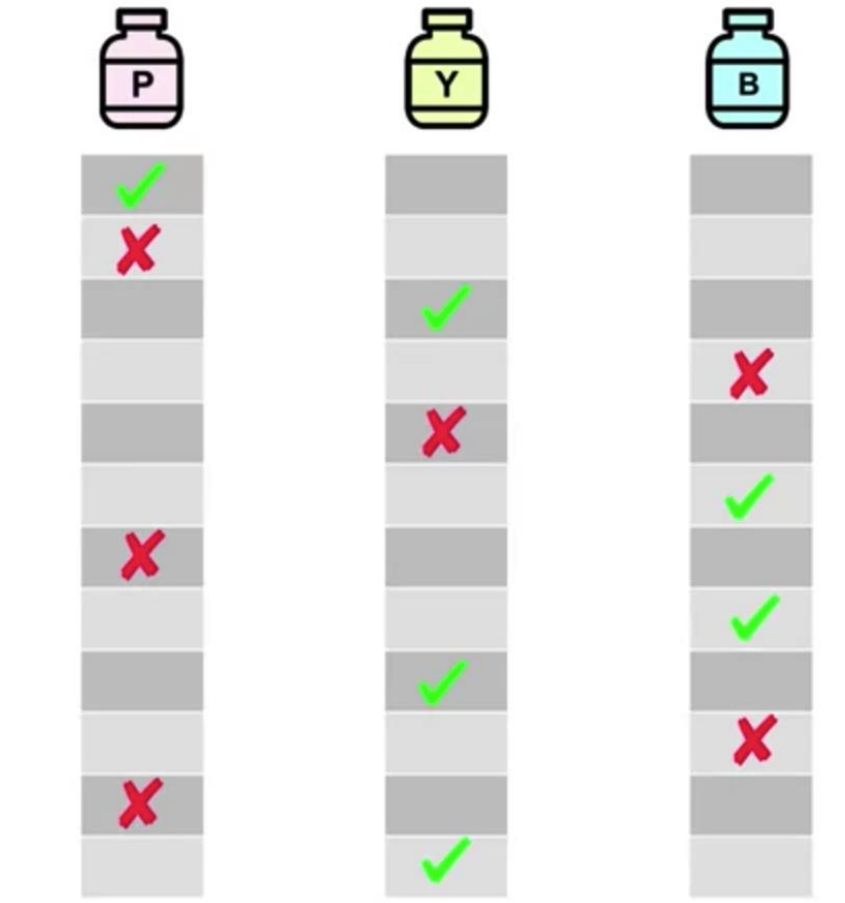
Epoch #10:



A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: clinical trials (reprise)

Initial Estimation = 0

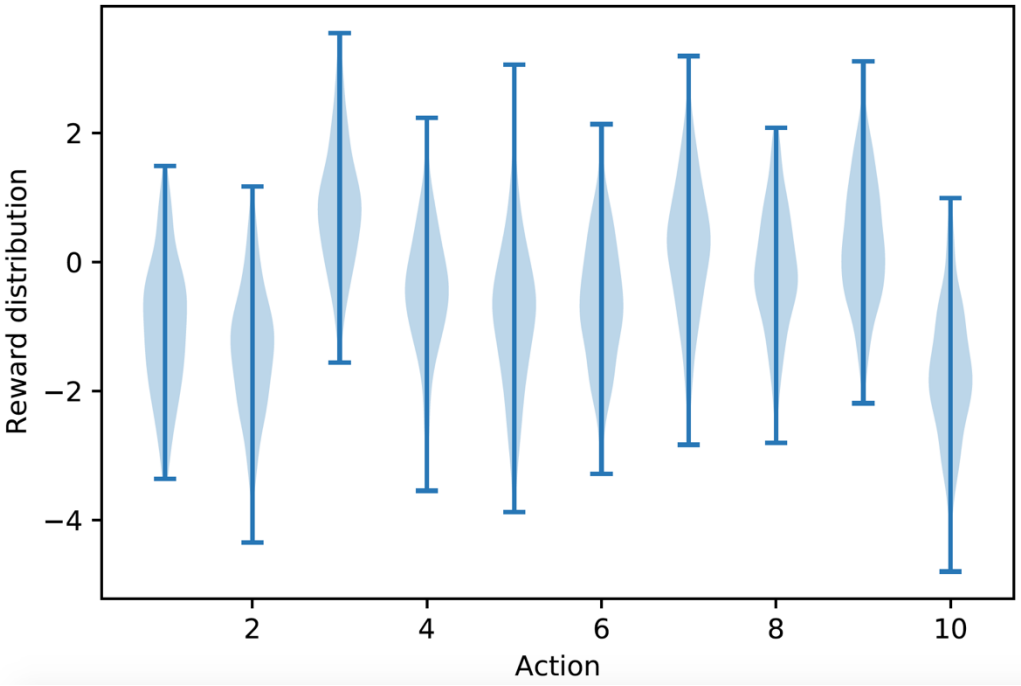


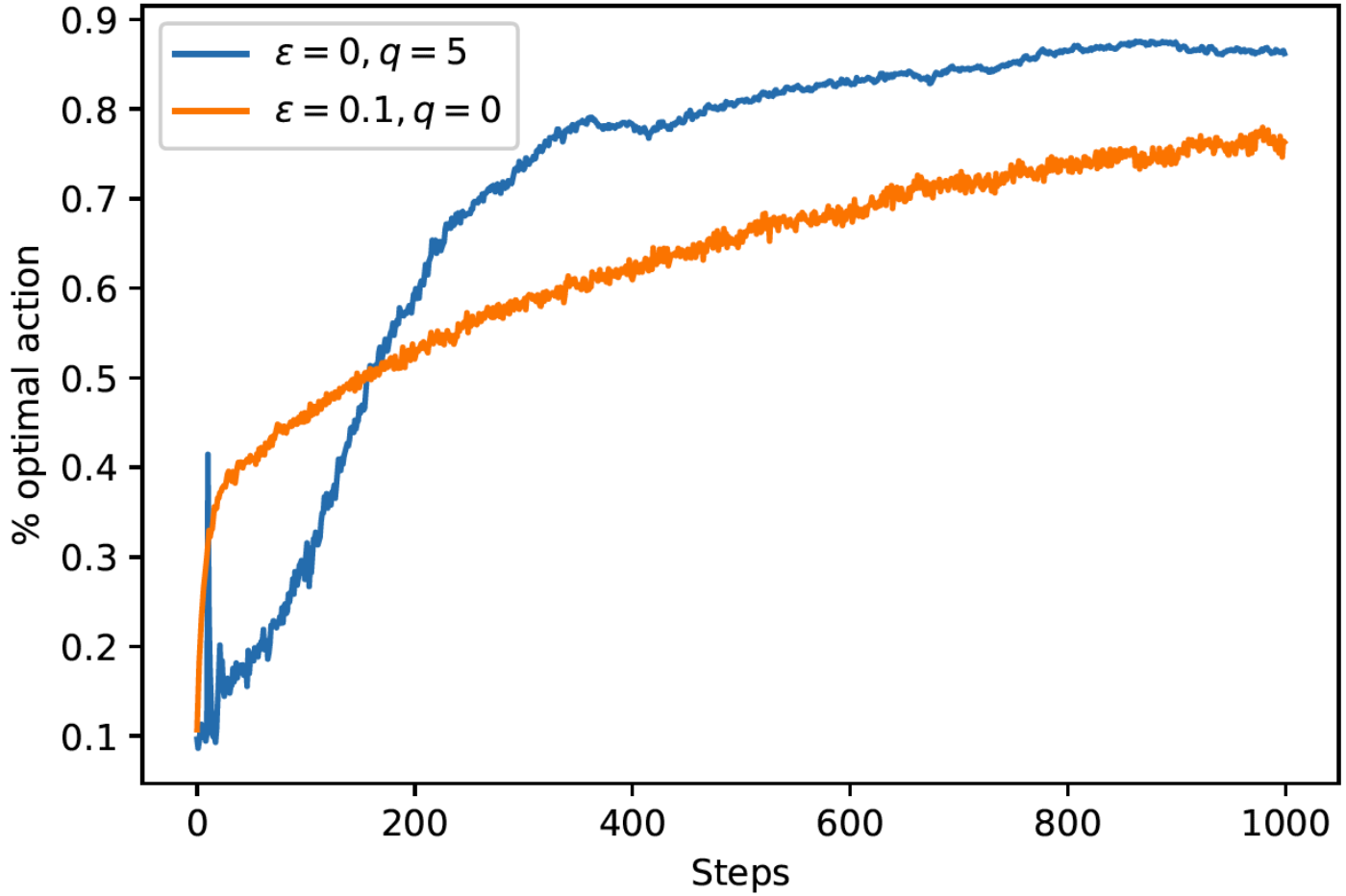
$Q_{12}(P) = 0.25$ $Q_{12}(Y) = 0.75$ $Q_{12}(B) = 0.5$

Initial Estimation = 2

$Q_{11}(P) = 0.375$ $Q_{11}(Y) = 0.5$ $Q_{11}(B) = 0.625$

Example: 10-armed bandits (reprise)





Optimistic Initial Values Limitations

Limitations

- Optimistic initial values only drive early exploration
- Such approaches are not well-suited for non-stationary problems
- In some applications we may not know what the optimistic initial value should be

However, it is a convenient heuristic and it is typically used in combination with other approaches that allowed us to encourage exploitation

#3 - Upper-Confidence-Bound (UCB) Action Selection

- Another approach for dealing with the Exploration-Exploitation dilemma is provided by UCB
- UCB uses uncertainty in the value estimates for governing the trade-off between exploration and exploitation
- Recall that in Ɛ-greedy

$$A_t \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}}\ Q_t(a) & \text{with probability } 1-\epsilon \\ a \sim Uniform(\{a_1 \ldots a_k\}) & \text{with probability } \epsilon \end{cases}$$

#3 - Upper-Confidence-Bound (UCB) Action Selection

- Another approach for dealing with t
  provided by UCB
- UCB uses uncertainty in the value es
  between exploration and exploitatic
- Recall that in Ɛ-greedy

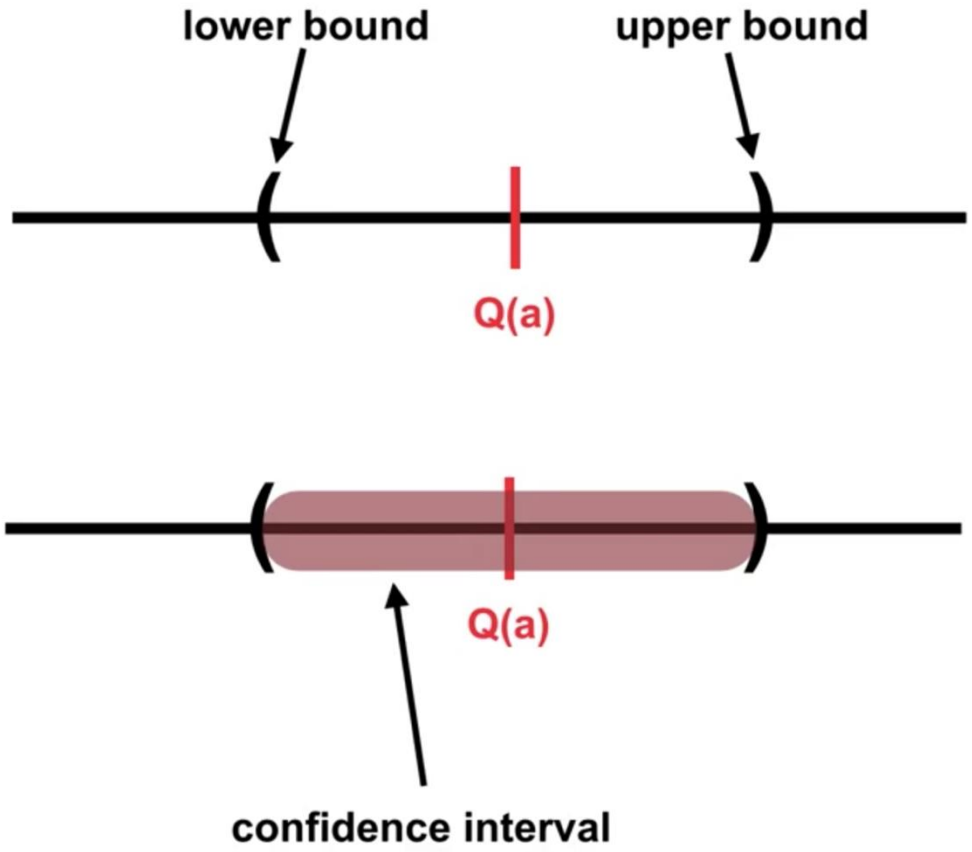
Can we do better? What if we can quantify the uncertainty in our action value estimation?

A simple idea could be to exploit how many times an actions has been taken!

$$A_t \leftarrow \begin{cases} \underset{a}{\operatorname{argmax}}\ Q_t(a) & \text{with probability } 1-\epsilon \\ a \sim Uniform(\{a_1 \dots a_k\}) & \text{with probability } \epsilon \end{cases}$$

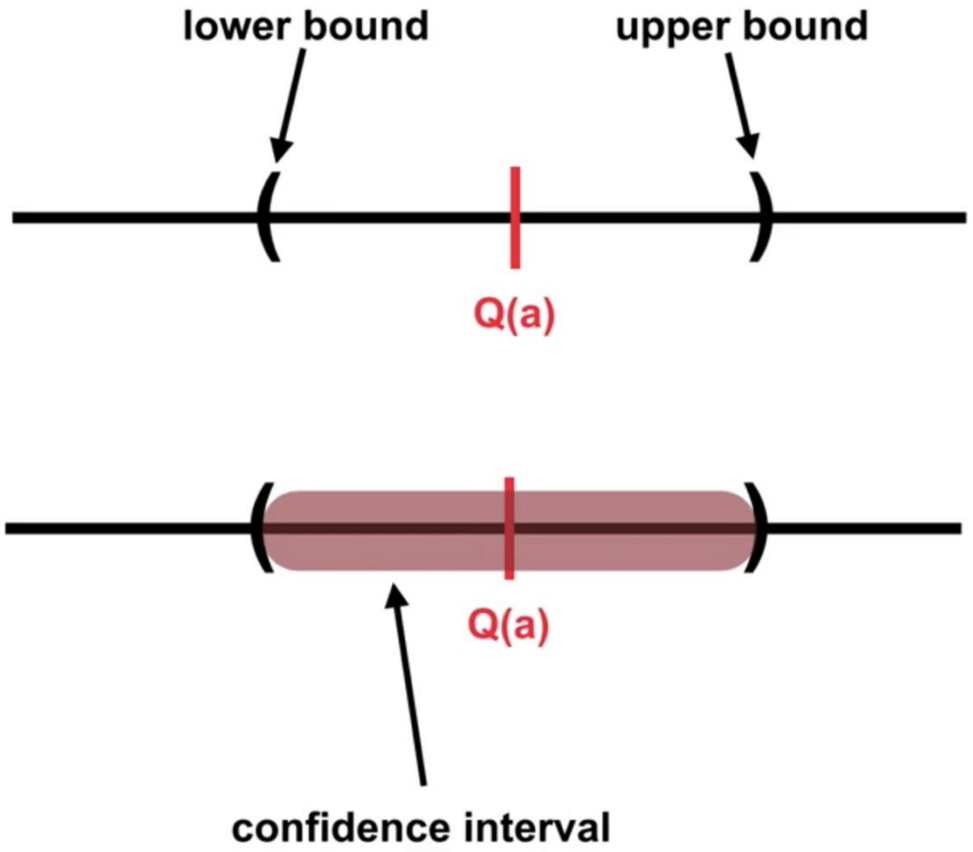
A. White, M. White 'Fundamentals of Reinforcement Learning'

UCB Action Selection



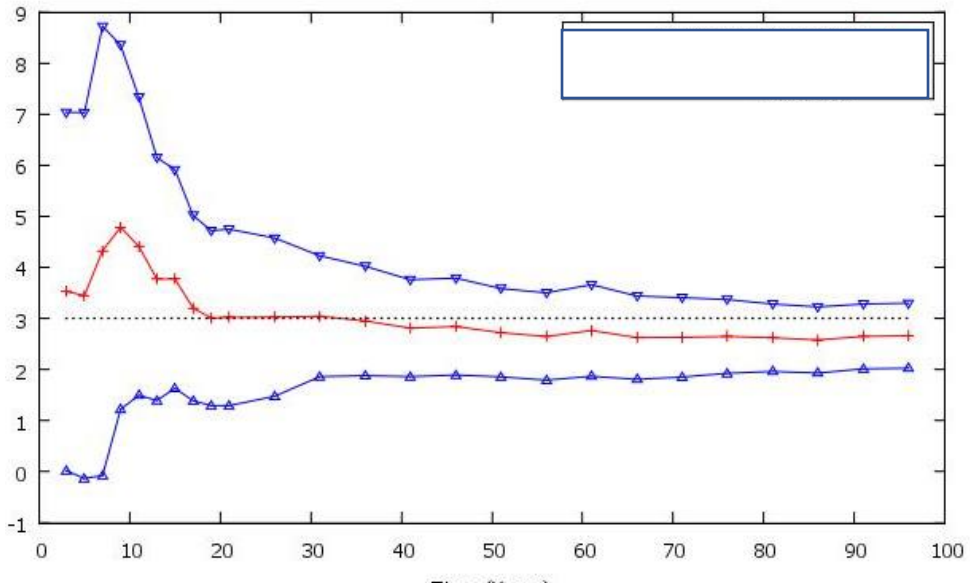
A. White, M. White 'Fundamentals of Reinforcement Learning'

UCB Action Selection



The more we take an action, the lower the confidence interval

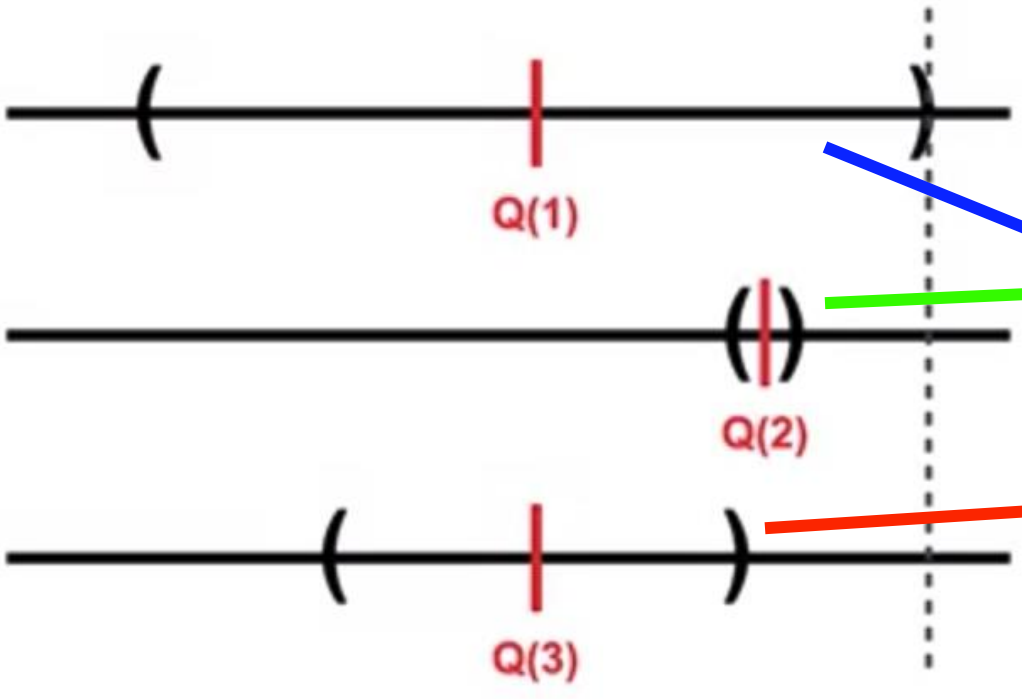
Pay attention: this is not the distribution of the award for action a, but our estimation of Q(a)!

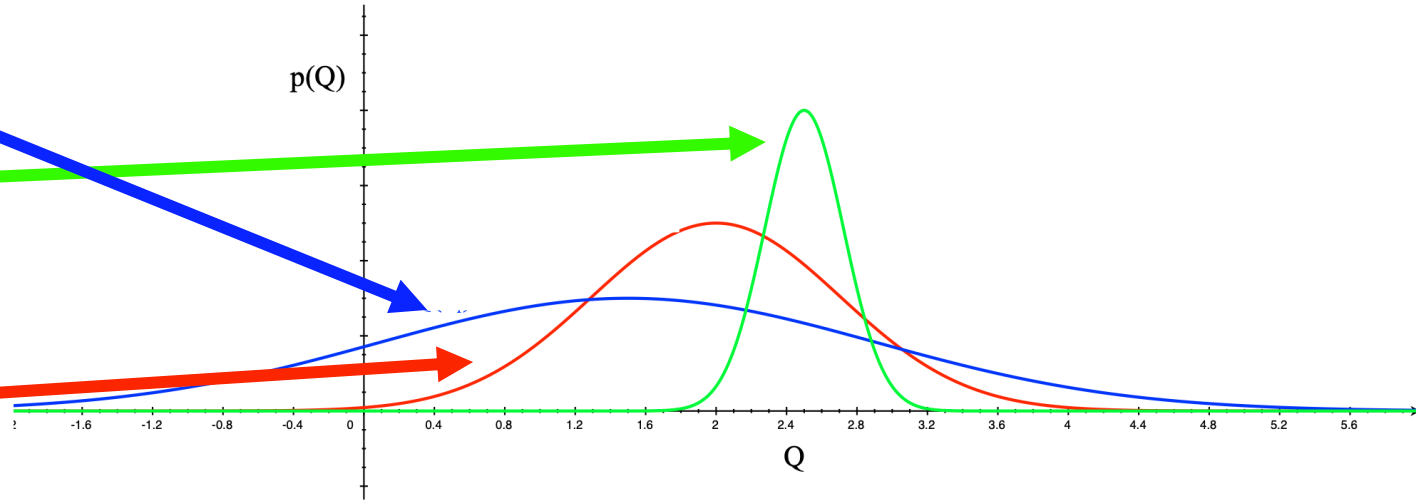


A. White, M. White 'Fundamentals of Reinforcement Learning'

UCB Action Selection

Optimism in the face of uncertainty: we take the action that 'potentially' could be the best!



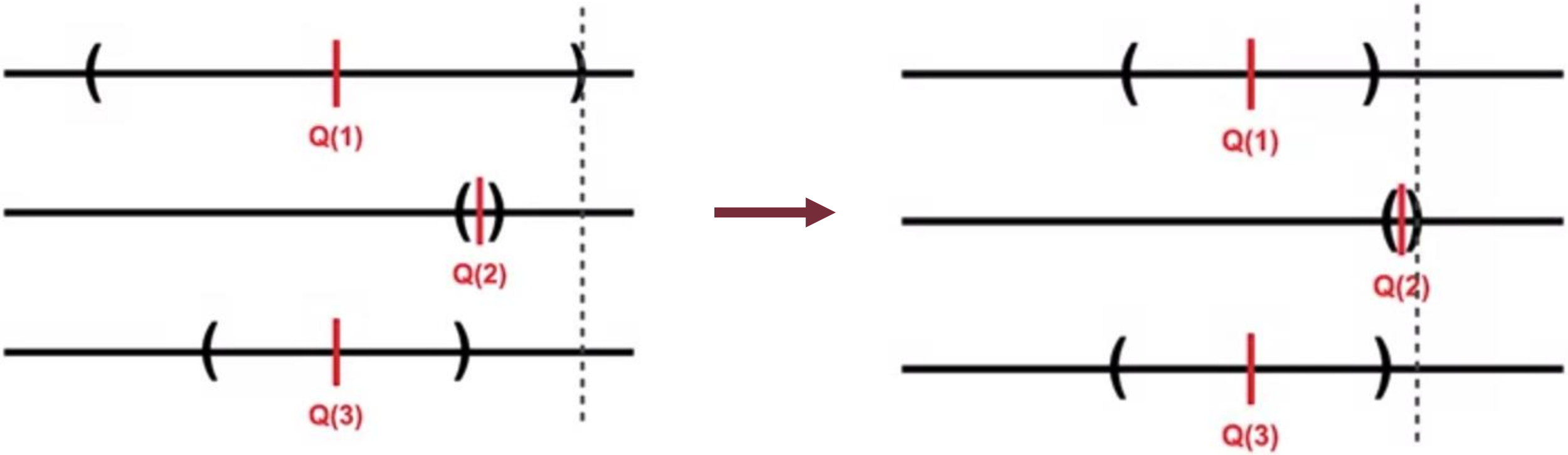


D. Silver 'Reinforcement Learning course' @ UCL

A. White, M. White 'Fundamentals of Reinforcement Learning'

UCB Action Selection

Optimism in the face of uncertainty: we take the action that 'potentially' could be the best!



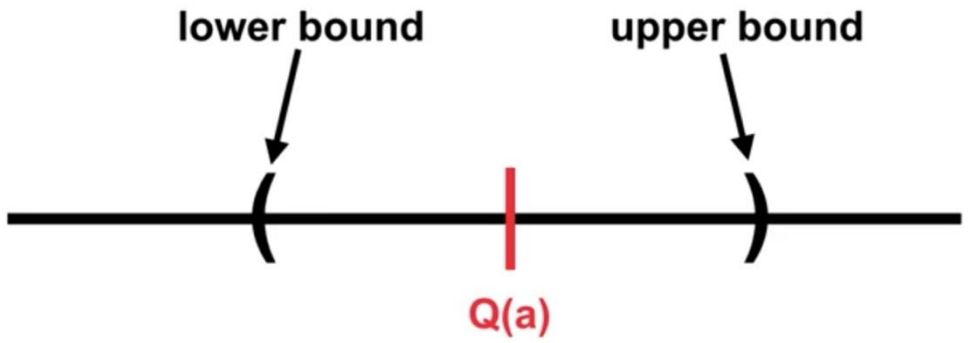
A. White, M. White 'Fundamentals of Reinforcement Learning'

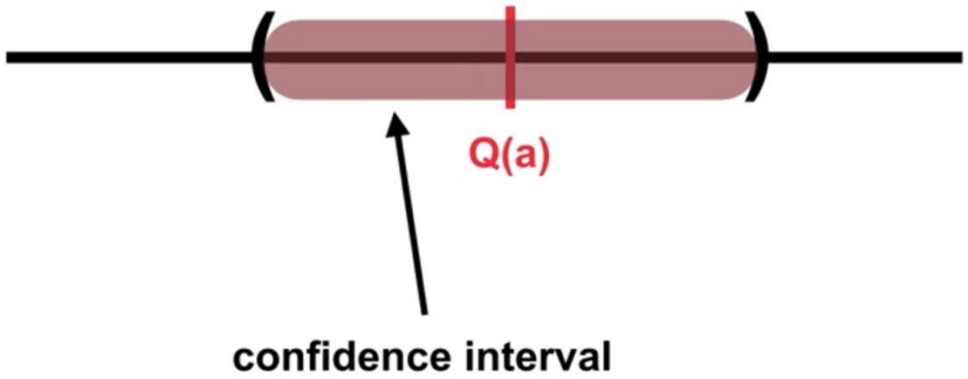
UCB Action Selection

$$A_t \doteq \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Exploration (Hoeffding's Inequality)

Exploitation





A. White, M. White 'Fundamentals of Reinforcement Learning'

UCB Action Selection

$$A_t \doteq \underset{a}{\operatorname{argmax}} \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$$

Exploration

Exploitation

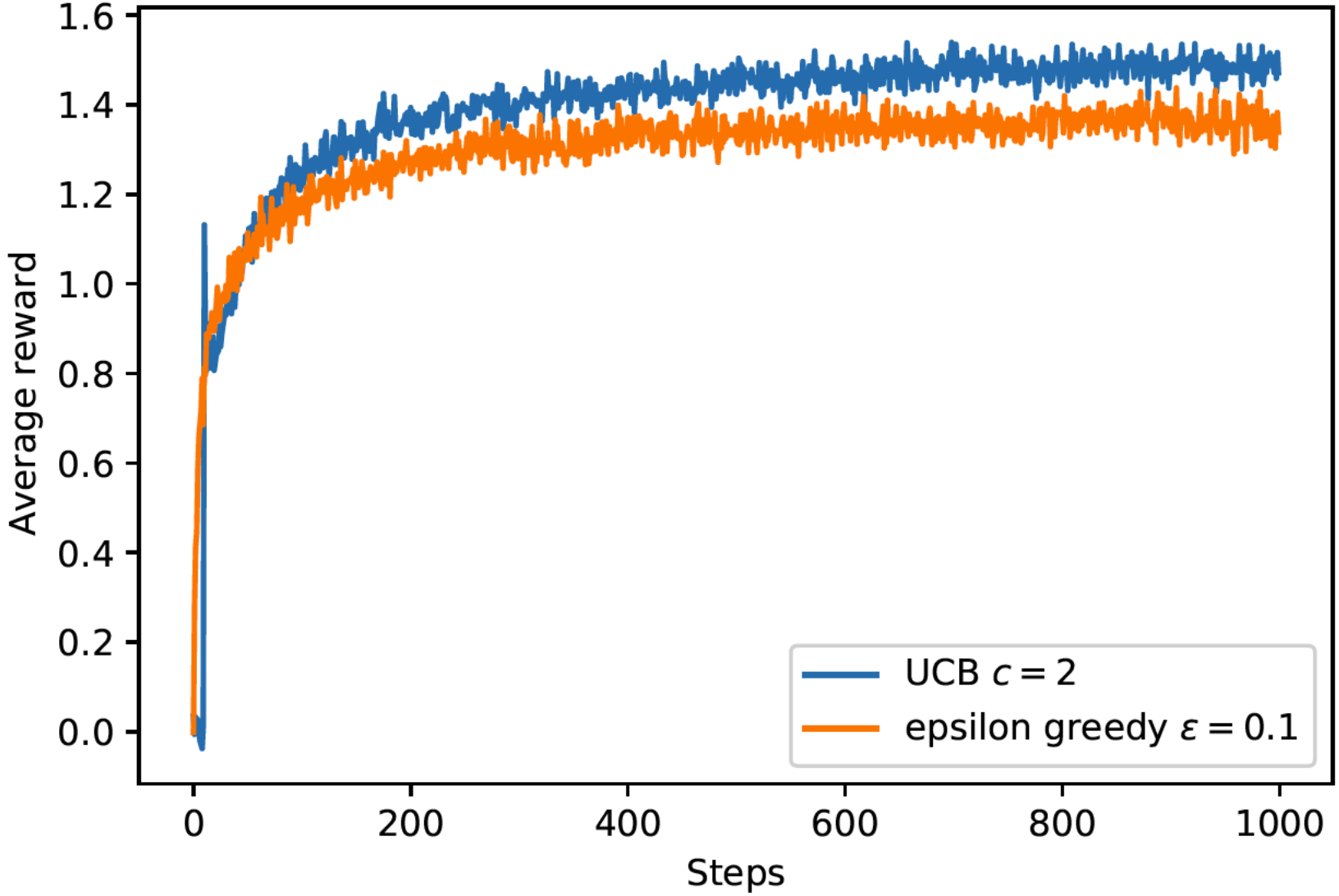
$$c\sqrt{\frac{ln\ t}{N_t(a)}} \rightarrow c\sqrt{\frac{ln\ \text{timesteps}}{\text{times action } a \text{ taken}}}$$

$$c\sqrt{\frac{ln\ 10000}{5000}} \rightarrow 0.043c$$

$$c\sqrt{\frac{ln\ 10000}{100}} \rightarrow 0.303c$$

A. White, M. White 'Fundamentals of Reinforcement Learning'

Example: 10-armed bandits (reprise)



#4 - Gradient Bandit Algorithms

- Another approach to encourage exploration without choosing randomly the action to explore is represented by Gradient Bandit Algorithms (GBA)
- In GBA a preference for each action is defined $H_t(a) \in \mathbb{R}$: the larger the preference, the more often the action is taken
- Preference has no interpretation in terms of reward: only relative preference of one action over another is important
- For preference we use the soft-max distribution*

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

$\pi_t(a)$ is the probability of taking action a at time t

* It is a probability distribution. For example, it is easy to see that the sum of all preferences (ie. over all actions) sums to 1

Gradient Bandit Algorithms

How to choose preferences?

Let's exploit gradient ascent!

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

You'll probably have seen gradient descent in Machine Learning…

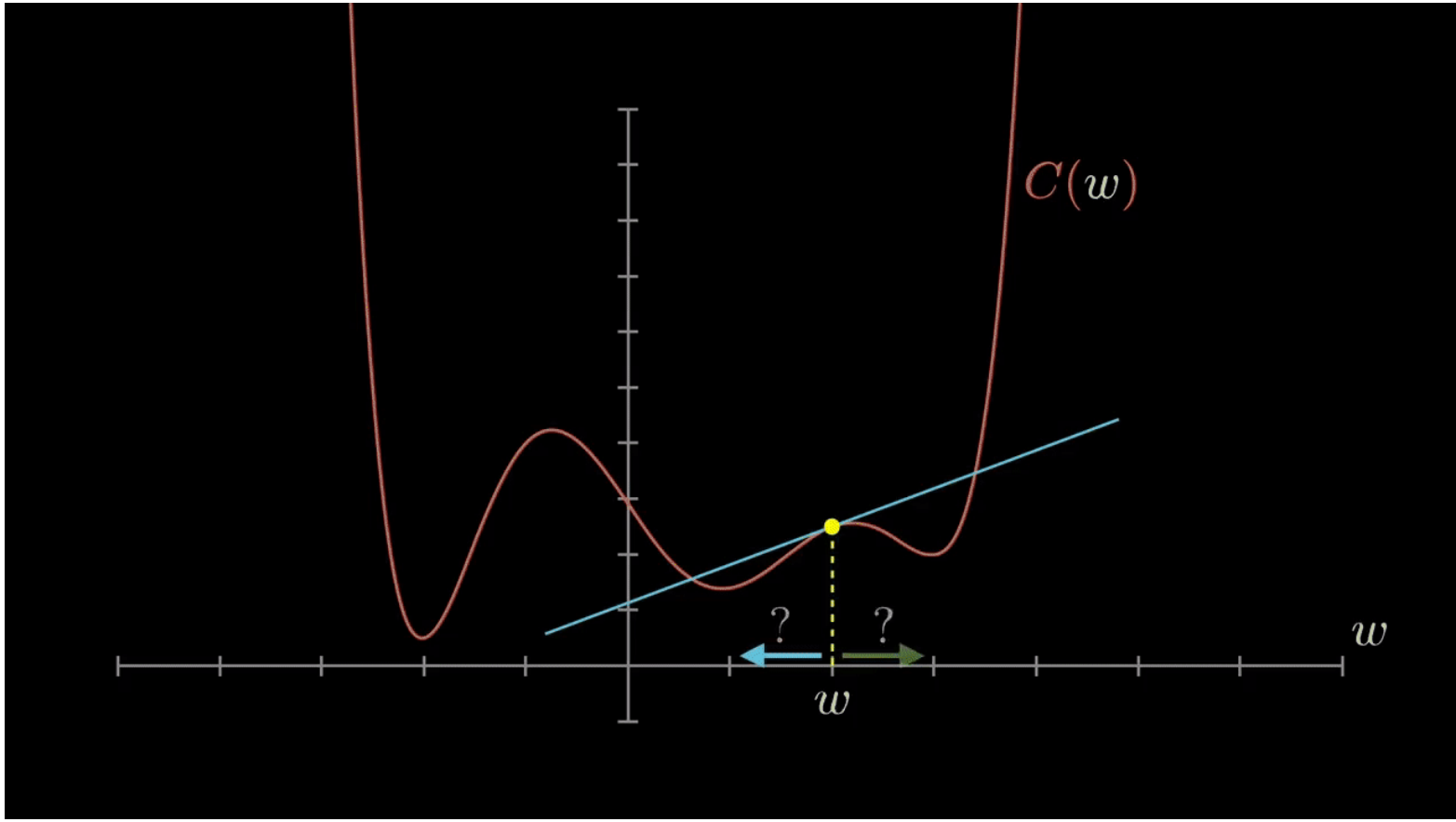
Gradient Bandit Algorithms

How to choose preferences?

Let's exploit gradient ascent!

For those of you who have not seen gradient descent in Machine Learning…

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$



(Gradient Descent)

We seek for a set of weights to minimize (or maximize) a function *J* (in ML, this happens frequently with loss functions) w.r.t. parameters *W*:

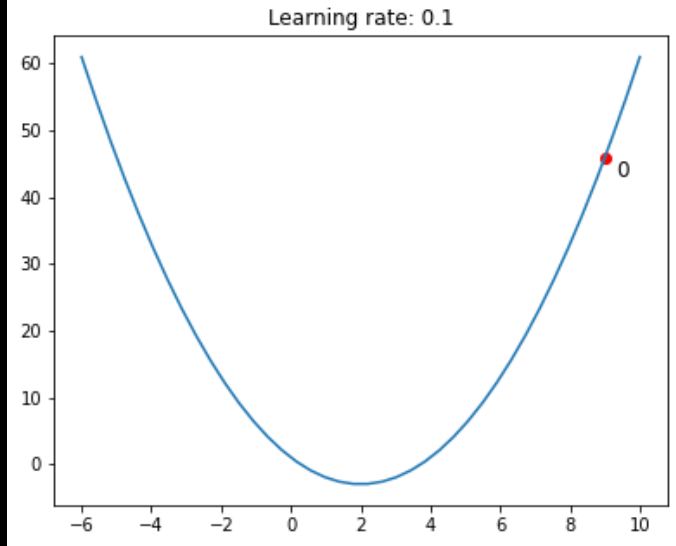
Algorithm (Gradient Descent)

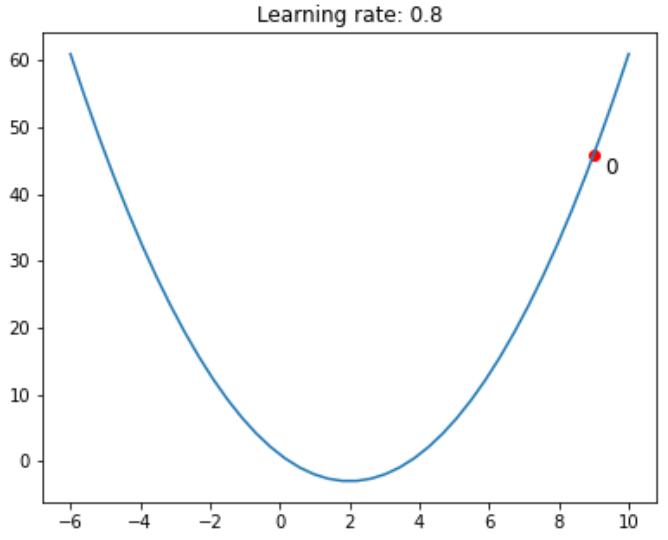
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Learning Rate

(Gradient Descent)

We seek for a set of weights to minin
frequently with loss functions) w.r.t.





Algorithm (Gradient Descent)

1. Initialize weights randomly $\sim \mathcal{N}(0,$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \alpha \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Learning Rate

Gradient Bandit Algorithms

$$\Pr\{A_t = a\} \doteq \frac{e^{H_t(a)}}{\sum_{b=1}^{k} e^{H_t(b)}} \doteq \pi_t(a)$$

- Let's start with all preferences equal to each other: (ie. $H_1(a) = 0 \; \forall \; a$)

- At each step, after selecting action A_t we receive reward R_t and we update our action preferences as follows:

$$H_{t+1}(A_t) \doteq H_t(A_t) + \alpha\big(R_t - \bar{R}_t\big)\big(1 - \pi_t(A_t)\big), \qquad \text{and}$$
$$H_{t+1}(a) \doteq H_t(a) - \alpha\big(R_t - \bar{R}_t\big)\pi_t(a), \qquad \text{for all } a \neq A_t$$

with:

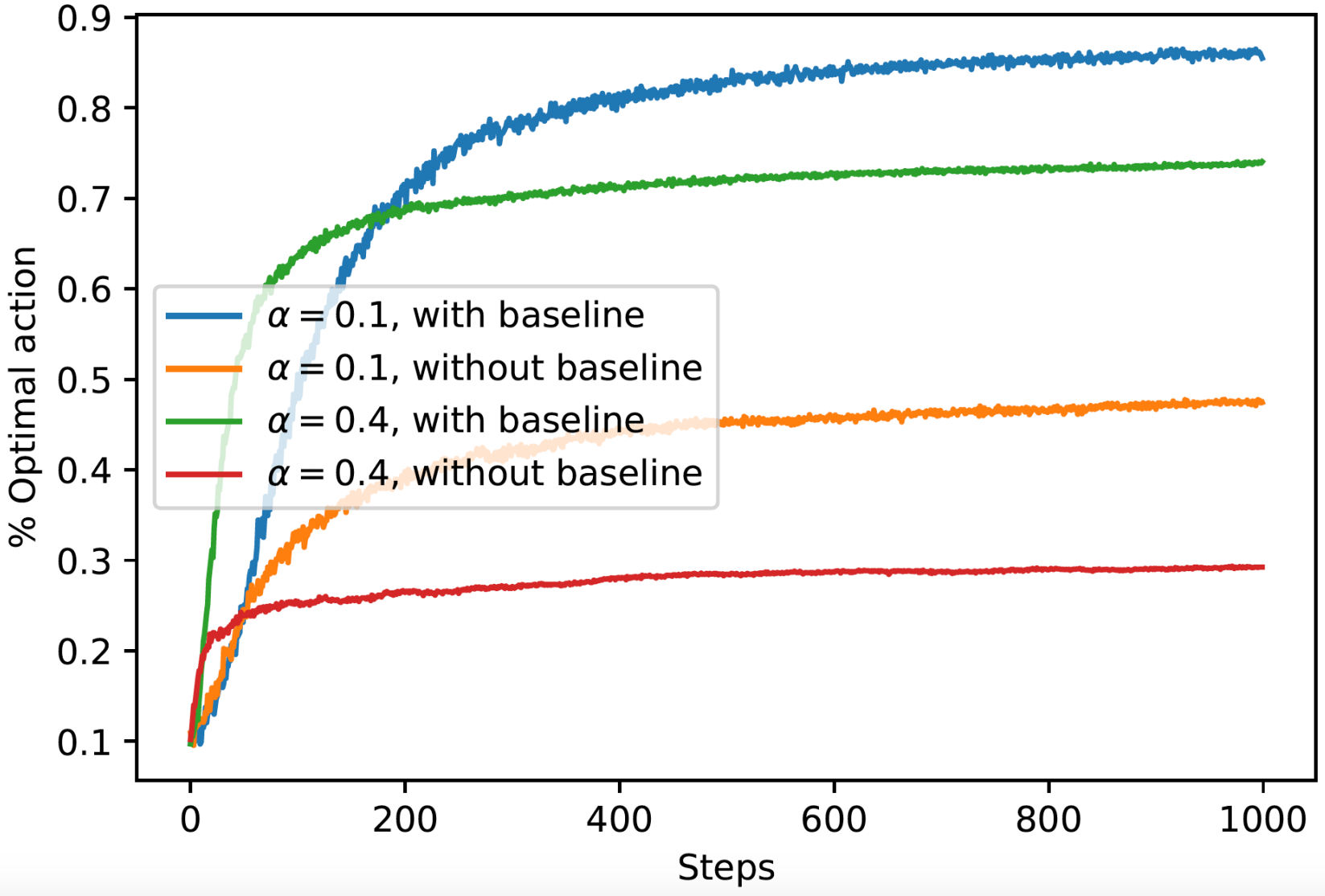
α the step-size parameter

$\bar{R}_t$ the average of the rewards up to but not including time t; this terms serves as a baseline, if the A_t provides better results than the average then we should prefer this actions over the others!

Example: 10-armed bandits (reprise)

With baseline: with $\bar{R}_t$

Without baseline:
with $\bar{R}_t = 0$

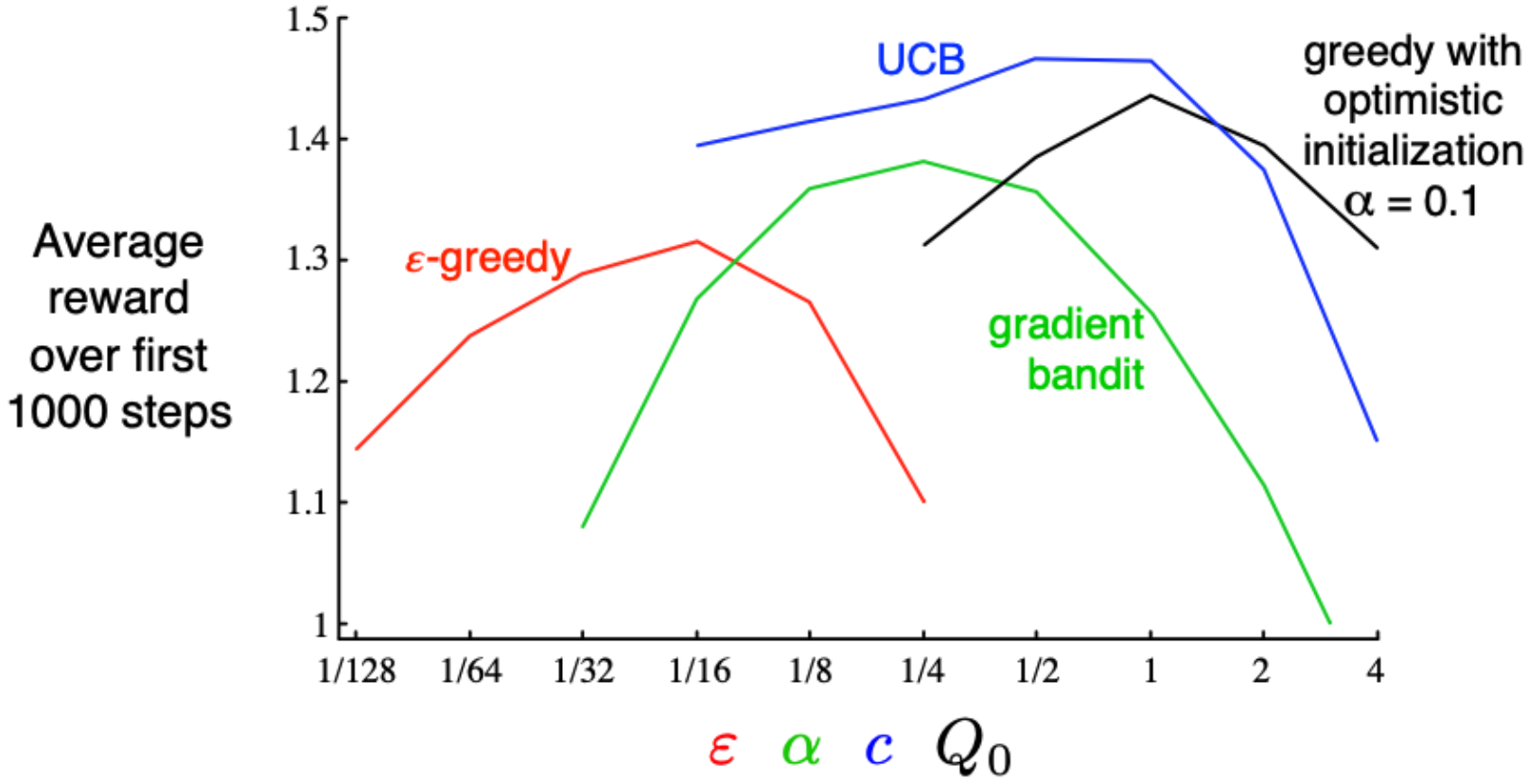


Summarizing

- We have seen the multi-armed bandits problem (ABP)
- We have seen several ways to handle the trade-off between exploration and exploitation:

1. Ɛ-greedy
2. Greedy with optimistic inizialization
3. Upper Confidence Bound (UCB)
4. Gradient Bandit

Parameter study with the 10 ABP->

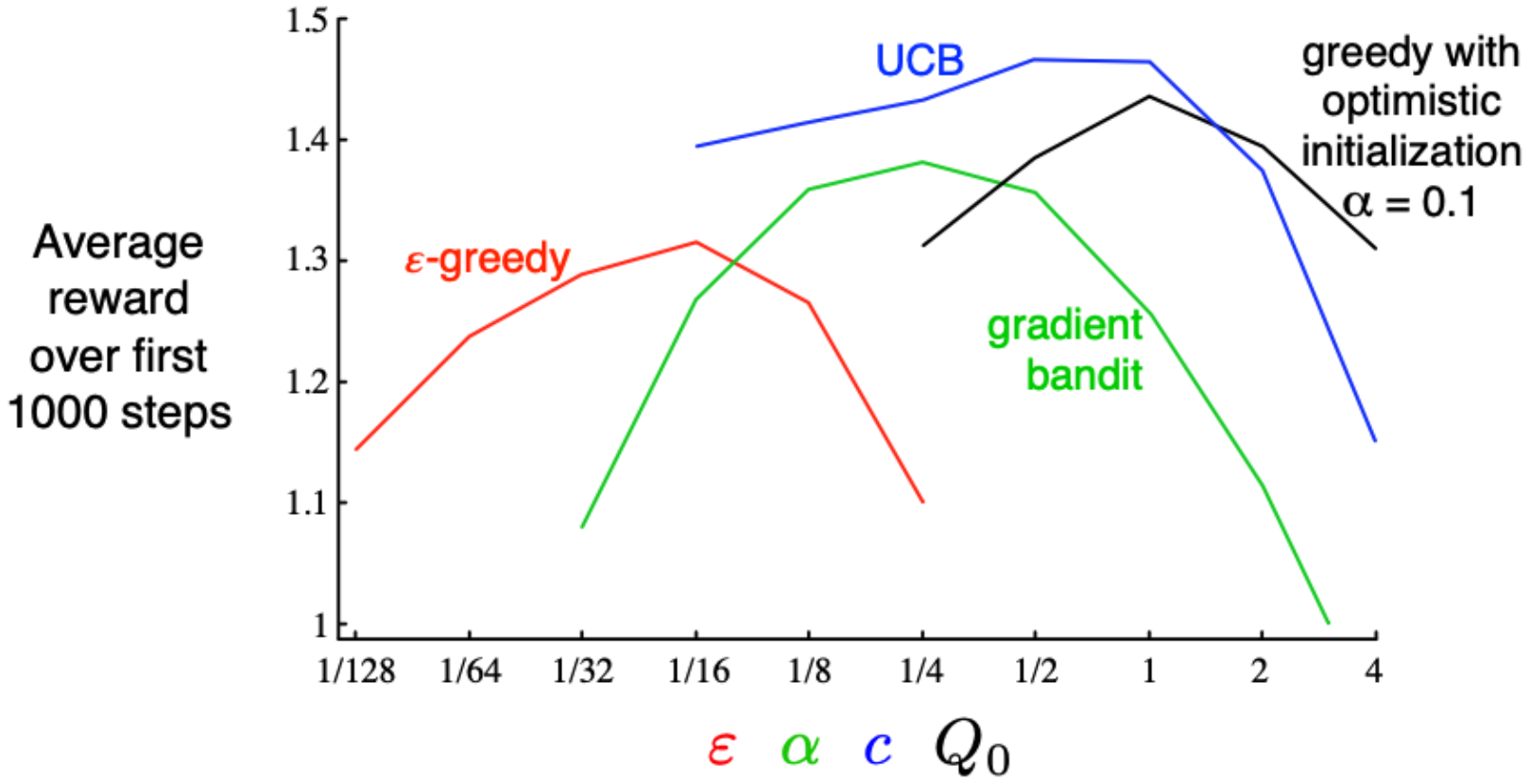


Summarizing

The parameter study showed us that:

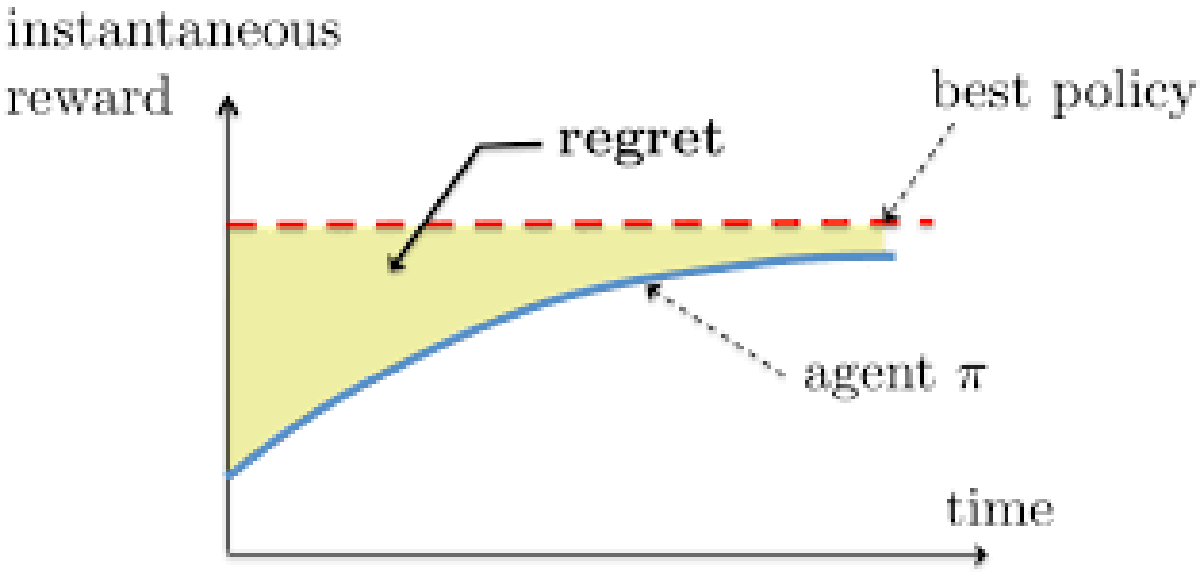
- The is always a trade-off in the choice of th parameter that can be associated with the trade-off between exploration and exploitation

- The number of episodes we have available may change a lot the efficiency of a strategy



We haven’t covered (1/2):

- The concept of regret, ie. the missed opportunity which is really helpful when studying problems taking into account the perspective of the available episodes (we will discuss this in the lab)



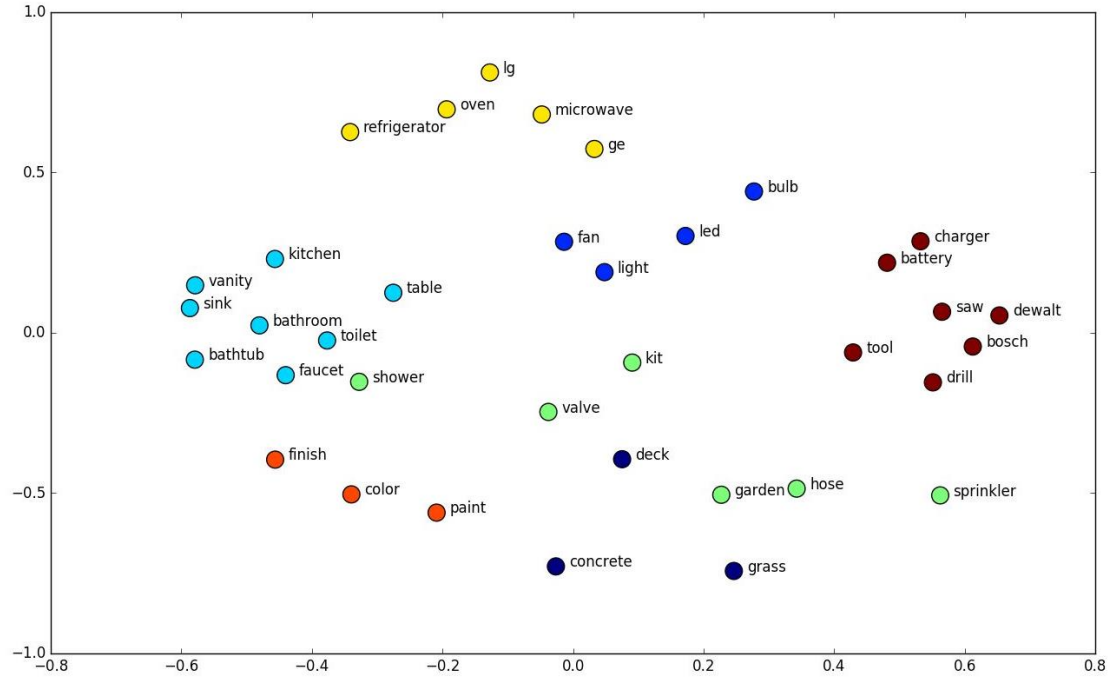
We haven’t covered (2/2):

Contextual bandits: we consider states! For example, in advertising suggestion we want to take into account information on the users actual search (or past search)

Contextual bandits can be seen as a combination of supervised learning (SL) & reinforcement learning (RL):

- RL optimize part of the suggestions based on previous decisions;

- SL optimize part of the suggestions based on the state (the current search or user)



(For handling text in SL you need tools like word-embedding…)

K-armed Bandits: Exam

- All the content of Chapter 2 of the book may be exam material
- ‘The Bandit Gradient Algorithm as Stochastic Gradient Ascent’ from page 38 is not exam material!

Credits & Additional Readings

- Several examples were taken from the Fundamental of Reinforcement Learning course by A. White and M. White
- Some contents were taken from ‘Reinforcement Learning @ UCL’ by D. Silver
- Additional Readings (inspired by D. Bacciu):

1. Seminal UCB paper https://nlp.jbnu.ac.kr/AML/papers/auer-ml-02.pdf
2. Randomized UCB algorithm for contextual bandits https://arxiv.org/pdf/1106.2369.pdf
3. Efficient learning of contextual bandit with an oracle http://proceedings.mlr.press/v32/agarwalb14.pdf
4. A Deep Learning based approach to generate exploration bonuses via model-based https://arxiv.org/pdf/1507.00814.pdf

AMCO
ARTIFICIAL INTELLIGENCE, MACHINE LEARNING AND CONTROL RESEARCH GROUP

Reinforcement Learning
2025/2026

RL2
Reinforcement Learning Research Lab @ UniPD

Lecture #02
RL Elements + Multi-armed Bandits

Gian Antonio Susto