## Part II:
## Reinforcement learning

## Challenges of reinforcement learning (recap from slides01)

Reinforcement learning is the subfield of machine learning that deals with learning from interaction with an unknown environment

Unlike supervised learning:

- the agent only gets feedback on the *chosen* action;
- the feedback is delayed: it applies to sequences of actions;
- by choosing actions, the agent is in control of what data is seen.

## Exploration–Exploitation Dilemma

At any point, the agent will have limited information, and can choose actions to:

- exploit that information: pick the action believed to be best
- explore: try out another action, maybe getting new information

In the long term, the best performance will be obtained if there is a good balance between exploring and exploiting

## Multi-armed bandits: introduction

Bandit problems are a simplified type of reinforcement problem:

- The environment has only one *state*
- but the exploration–exploitation dilemma is there (all the more clearly)

The simplification allows these problems to be thoroughly analyses mathematically, though many important questions are still unanswered

- The part of chapter 2 we will skip (starting with section 2.5) goes into some of this theory

## Multi-armed bandits: definition

(Named after slot machines: "one-armed bandits")

Each time step $t$, the agent chooses an action $A_t$ and receives reward $R_t$

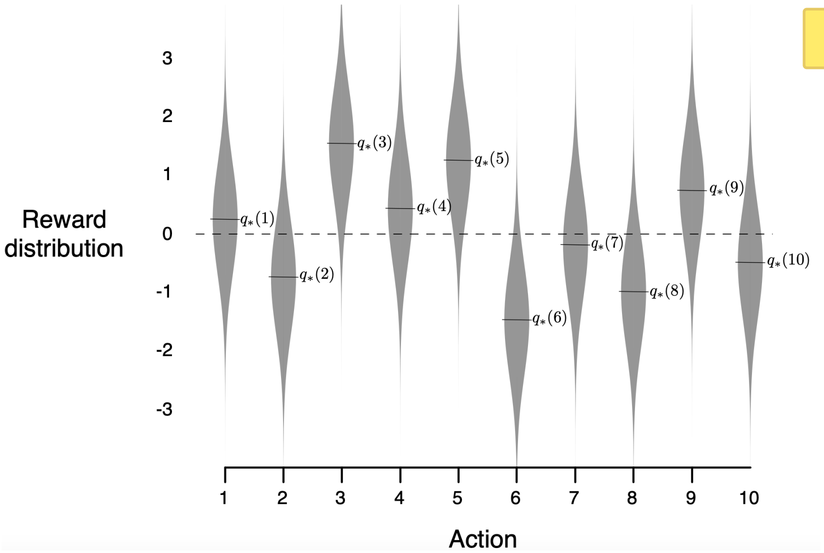
- $k$ available actions
- $q_*(a) := \mathbb{E}[R_t \mid A_t = a]$: expected reward of action $a$ (also called action value)
  *($*$'s will denote 'true' or 'optimal')*

Taking the action with the largest $q_*(a)$ would give the optimal reward... but we don't know those values

- $Q_t(a)$: our estimate of $q_*(a)$ at time $t$ (before taking an action)
  - Obvious choice: the average of the $R_i$'s at all $i < t$ where $A_i = a$

Taking the action with the largest $Q_t(a)$ is called the greedy action

# Example


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

## Adding exploration

By taking the greedy action, we are doing only exploitation, no exploration

Straightforward way to ensure we do some exploration as well: for each action, with some small probability $\epsilon$, select an action uniformly at random

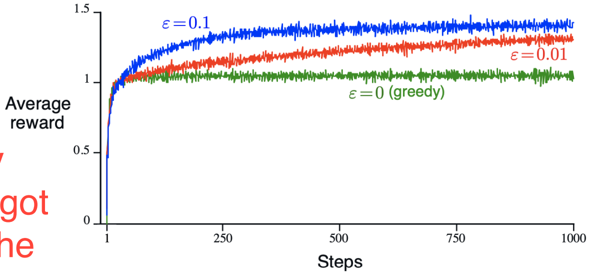
- Note that this will still end up selecting the greedy action, with overall probability $\epsilon/k$

This is called an $\epsilon$-greedy strategy

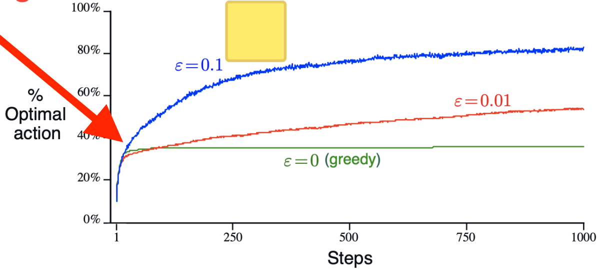
Why does this work?

- As $t$ goes to infinity, so does the number of times each action is tried. So our estimates $Q_t(a)$ will converge to $q_*(a)$.

## Example results (over 2000 different problems)



Greedy algorithm got lucky in the beginning



## Observations

- $\epsilon$ too small (e.g. 0): likely to get stuck with suboptimal action
- $\epsilon$ too large: in the long run, keeps making wrong moves even if it has already found the optimal action

Good value depends very much on the problem. E.g. if $R_t$ has small variance, it will be easier to identify the optimal action, so a small $\epsilon$ is good enough

Improvements:

- We could let $\epsilon$ get smaller over time
- Many more ways to improve over this simple idea (e.g. by taking into account how confident we are about each estimate, or how likely some action is to turn out optimal)

## Incrementally tracking value estimates

Suppose $R_1, \ldots, R_{n-1}$ are previous rewards obtained after selecting action $a$ *(note the simplified indexing on this slide)*. Then

$$Q_n := \frac{R_1 + \ldots + R_{n-1}}{n-1}$$

No need to keep track of all numbers $R_1, \ldots, R_{n-1}$. Instead, update $Q_n$ to $Q_{n+1}$ when new reward $R_n$ comes in:

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

## Incremental formula

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

- Cheap in terms of memory and computation
- Example of a general form we will see very often:

$$\text{NewEstimate} = \text{OldEstimate} + \text{StepSize}(\text{Target} - \text{OldEstimate})$$

## Simple bandit algorithm

**A simple bandit algorithm**

```
Initialize, for $a = 1$ to $k$:
    $Q(a) \leftarrow 0$
    $N(a) \leftarrow 0$

Loop forever:
    $A \leftarrow \begin{cases} \arg\max_a Q(a) & \text{with probability } 1-\varepsilon \quad \text{(breaking ties randomly)} \\ \text{a random action} & \text{with probability } \varepsilon \end{cases}$
    $R \leftarrow bandit(A)$
    $N(A) \leftarrow N(A) + 1$
    $Q(A) \leftarrow Q(A) + \frac{1}{N(A)}\left[R - Q(A)\right]$
```

## Markov Decision Process

The general reinforcement learning problem can be formalized by a Markov decision process (MDP)

Compared to bandit problems: the environment can now be in one of multiple states $s$

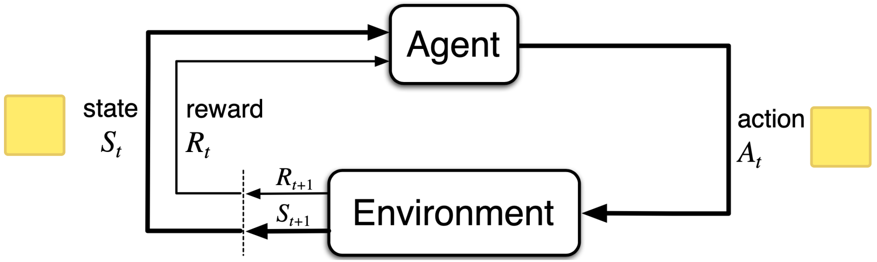
- Could reflect raw sensor readings
- Could also reflect other information the agent has about the environment

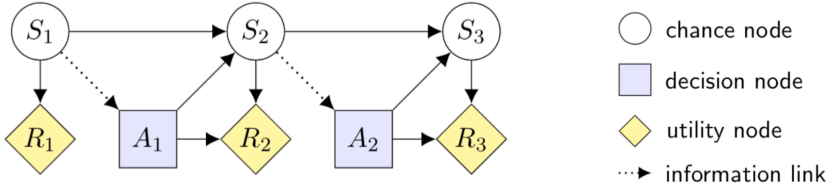
Has been studied extensively, e.g. in mathematics and engineering (where it also called *control*

## MDPs: additional challenge

- The agent's action may influence the state of the environment
- So we may need to choose between large immediate reward but going to a less desirable state, or small immediate reward but a better chance of going to a better state in the future

## Reward, state, action





Note that the index $t$ increases *after* the action

## Notation

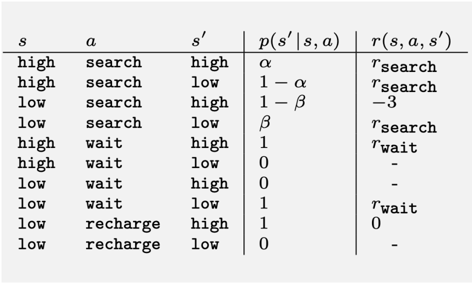
- $\mathcal{A}(s)$: set of actions available in state $s$
    - Even if 'the same' action is available in different states, it might be good in one but bad in another state. So we can think of them as being different actions that we need to learn about separately.
- Dynamics function (suggestively written with a $\mid$):

$$\underline{p(s', r \mid s, a) := P(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a)}$$

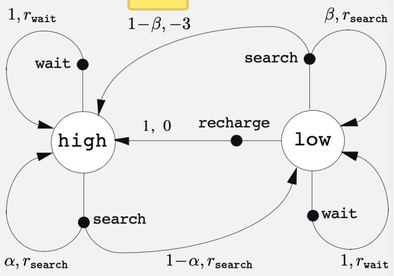
'Markov' decision process: new state and reward determined by preceding state and action alone (and in particular not by anything longer ago)

# Example: recycling robot

## from the book



| $s$ | $a$ | $s'$ | $p(s' \mid s, a)$ | $r(s, a, s')$ |
|---|---|---|---|---|
| high | search | high | $\alpha$ | $r_{\texttt{search}}$ |
| high | search | low | $1 - \alpha$ | $r_{\texttt{search}}$ |
| low | search | high | $1 - \beta$ | $-3$ |
| low | search | low | $\beta$ | $r_{\texttt{search}}$ |
| high | wait | high | 1 | $r_{\texttt{wait}}$ |
| high | wait | low | 0 | - |
| low | wait | high | 0 | - |
| low | wait | low | 1 | $r_{\texttt{wait}}$ |
| low | recharge | high | 1 | 0 |
| low | recharge | low | 0 | - |


$1, r_{\texttt{wait}}$
$1-\beta, -3$
$\beta, r_{\texttt{search}}$
wait
search
high
$1,\ 0$
recharge
low
search
wait
$\alpha, r_{\texttt{search}}$
$1-\alpha, r_{\texttt{search}}$
$1, r_{\texttt{wait}}$