Reinforcement Learning

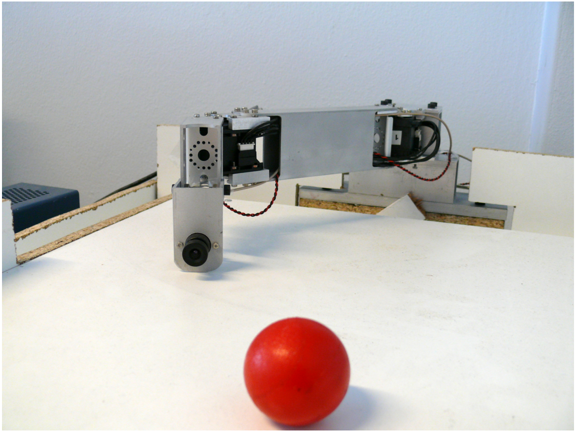
Part I: The Classical Setting

Ivo Grondman Robert Babuška

Knowledge-Based Control Systems
2012-03-05

Demo: RL for a robot goalkeeper

Learn how to catch ball, using video camera image



Outline

1. Reinforcement learning basics
   - Introduction
   - Elements of RL
   - RL solution
2. Algorithms
3. Accelerating RL

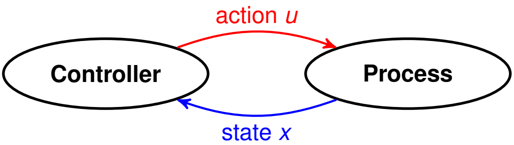
Why learning?

Learning can find solutions that:

1. cannot be found in advance
   - problem too complex
     (e.g., controlling highly nonlinear systems)
   - problem not fully known beforehand
     (e.g., robotic exploration of extraterrestrial planets)
2. steadily improve
3. adapt to time-varying environments

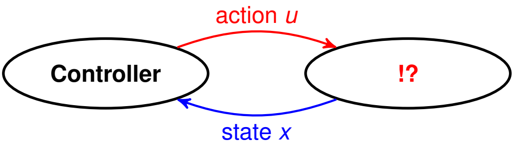
Essential for any **intelligent** system

Principle of RL



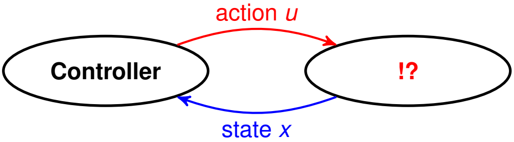
- Interact with a system through states and actions
- Inspired by human and animal learning
- Receive rewards as performance feedback

Principle of RL



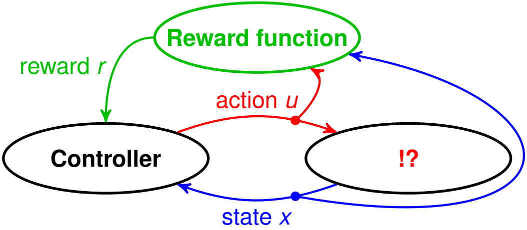
- Interact with a system through states and actions
- Inspired by human and animal learning
- Receive rewards as performance feedback

Principle of RL



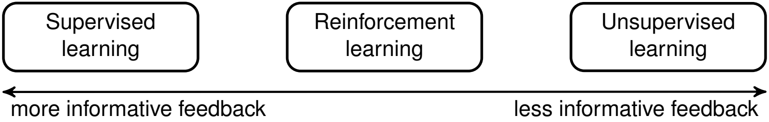
- Interact with a system through states and actions
- Inspired by human and animal learning
- Receive rewards as performance feedback

Principle of RL



- Interact with a system through states and actions
- Inspired by human and animal learning
- Receive rewards as performance feedback

RL on the Machine Learning spectrum



Spectrum: Supervised learning

Supervised learning	Reinforcement learning	Unsupervised learning
more informative feedback		less informative feedback

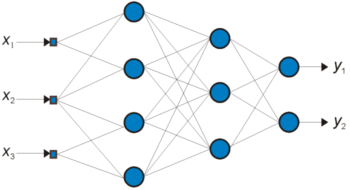
- For each input sample x, **correct output** y is known
- Infer input-output relationship $y \approx g(x)$
- Example: neural networks

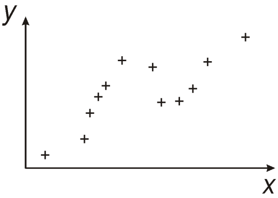
Spectrum: Supervised learning

Supervised learning — Reinforcement learning — Unsupervised learning

more informative feedback ⟷ less informative feedback

- For each input sample x, **correct output** y is known
- Infer input-output relationship $y \approx g(x)$
- Example: neural networks



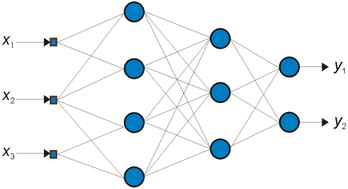


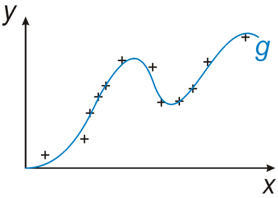
Spectrum: Supervised learning

Supervised learning | Reinforcement learning | Unsupervised learning

more informative feedback ⟷ less informative feedback

- For each input sample x, **correct output** y is known
- Infer input-output relationship $y \approx g(x)$
- Example: neural networks





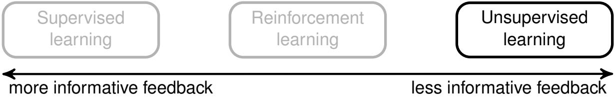
Spectrum: Unsupervised learning

Supervised learning | Reinforcement learning | Unsupervised learning

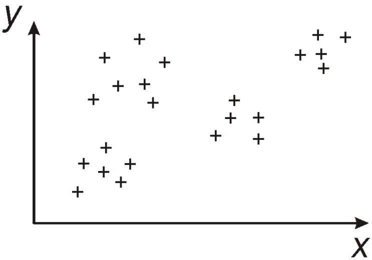
more informative feedback ⟷ less informative feedback

- Only input samples available – **no outputs**
- Find patterns in the data
- Example: clustering

Spectrum: Unsupervised learning



- Only input samples available – **no outputs**
- Find patterns in the data
- Example: clustering

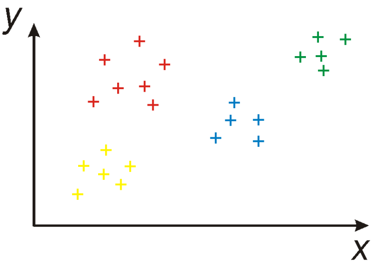


Spectrum: Unsupervised learning

Supervised learning — Reinforcement learning — **Unsupervised learning**

more informative feedback ⟷ less informative feedback

- Only input samples available – **no outputs**
- Find patterns in the data
- Example: clustering



Spectrum: Reinforcement learning

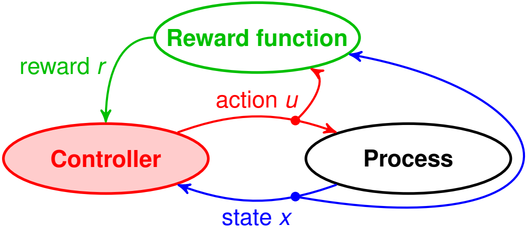
Supervised learning | Reinforcement learning | Unsupervised learning

more informative feedback ⟷ less informative feedback

- Correct outputs not available, **only rewards**
- Find optimal control behavior

Reinforcement learning = Control

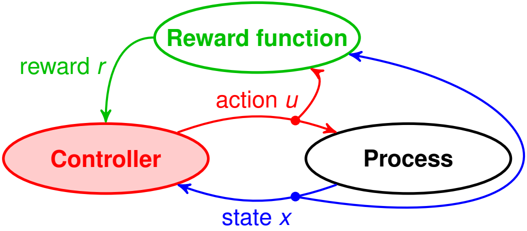
Reinforcement learning is about **control**:
optimal, adaptive, and model-free



This lecture: **classical RL** – discrete states and actions

Reinforcement learning = Control

Reinforcement learning is about **control**:
optimal, adaptive, and model-free



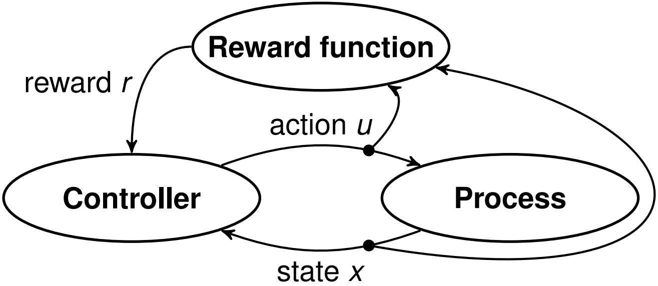
This lecture: **classical RL** – discrete states and actions

1 Reinforcement learning basics
- Introduction
- Elements of RL
- RL solution

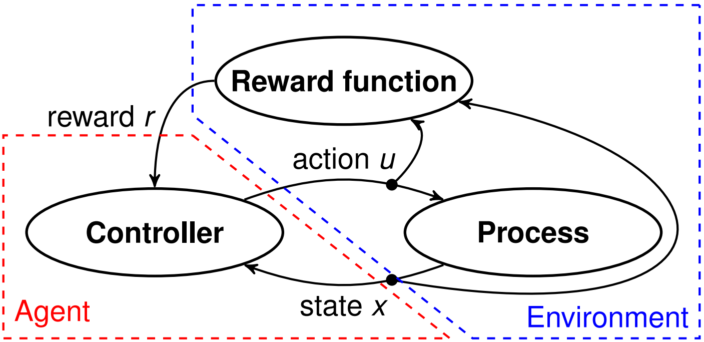
2 Algorithms

3 Accelerating RL

Environment and agent



Environment and agent



The environment

The environment is modeled by an MDP:

Markov Decision Process (MDP)

An MDP is a tuple $\langle X, U, f, \rho \rangle$ where:

- X is the finite state space
- U is the finite action space
- $f : X \times U \rightarrow X$ is the state transition function
- $\rho : X \times U \rightarrow \mathbb{R}$ is the reward function

$x_{k+1} = f(x_k, u_k)$, with k the discrete time

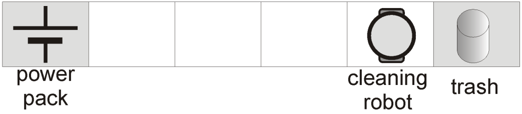
Note: stochastic formulation is possible

The agent

The agent is a state feedback controller:

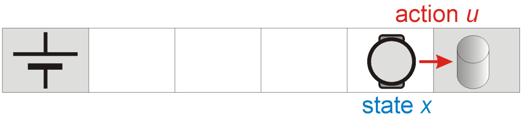
- Learns optimal mapping from states to actions
- Policy $\pi : X \mapsto U$ is the control law

A simple cleaning robot example



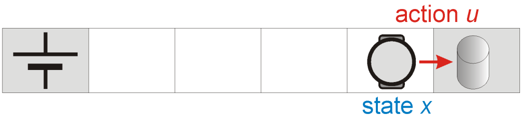
- Cleaning robot in a 1-D world
- Goal: pick up trash (reward $+5$) or power pack (reward $+1$)
- After picking up item, episode terminates

Cleaning robot: State & action

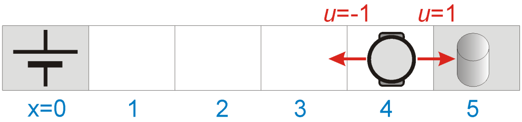


- Robot in given state x (cell)
- and takes action u (e.g., move right)

Cleaning robot: State & action

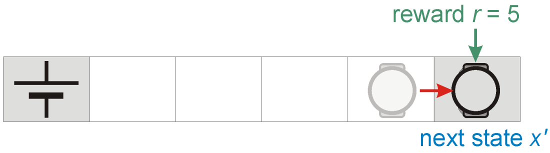


- Robot in given state x (cell)
- and takes action u (e.g., move right)



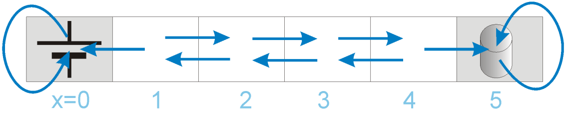
- State space $X = \{0, 1, 2, 3, 4, 5\}$
- Action space $U = \{-1, 1\} = \{\text{left, right}\}$

Cleaning robot: Transition & reward



- Robot reaches next state x'
- and receives reward r = quality of transition (here, $+5$ for collecting trash)

Cleaning robot: Transition & reward functions



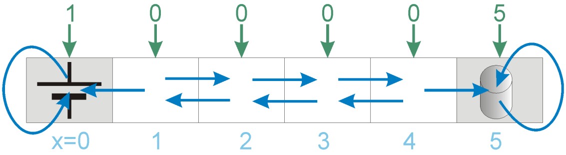
- Transition function (process behavior):

$$x' = f(x,u) = \begin{cases} x & \text{if } x \text{ is terminal (0 or 5)} \\ x + u & \text{otherwise} \end{cases}$$

- Reward function (immediate performance):

$$r = \rho(x,u) = \begin{cases} 1 & \text{if } x = 1 \text{ and } u = -1 \text{ (powerpack)} \\ 5 & \text{if } x = 4 \text{ and } u = 1 \text{ (trash)} \\ 0 & \text{otherwise} \end{cases}$$

Cleaning robot: Transition & reward functions



- Transition function (process behavior):

$$x' = f(x, u) = \begin{cases} x & \text{if } x \text{ is terminal (0 or 5)} \\ x + u & \text{otherwise} \end{cases}$$

- Reward function (immediate performance):

$$r = \rho(x, u) = \begin{cases} 1 & \text{if } x = 1 \text{ and } u = -1 \text{ (powerpack)} \\ 5 & \text{if } x = 4 \text{ and } u = 1 \text{ (trash)} \\ 0 & \text{otherwise} \end{cases}$$

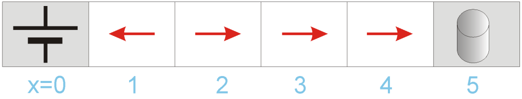
Cleaning robot: Policy

- Policy π: mapping from x to u (state feedback)
- Determines controller behavior

Cleaning robot: Policy

- Policy π: mapping from x to u (state feedback)
- Determines controller behavior

Example:



$$\pi(0) = * \qquad \pi(1) = -1 \qquad \pi(2) = 1$$
$$\pi(3) = 1 \qquad \pi(4) = 1 \qquad \pi(5) = *$$

$*$ action irrelevant in terminal state

Learning goal

Find π that maximizes discounted return:

$$R^{\pi}(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k))$$

from any x_0

Discount factor $\gamma \in [0, 1)$:

- induces a “pseudo-horizon” for optimization
- bounds infinite sum
- encodes increasing uncertainty about the future
- helps convergence of algorithms

Learning goal

Find π that maximizes discounted return:

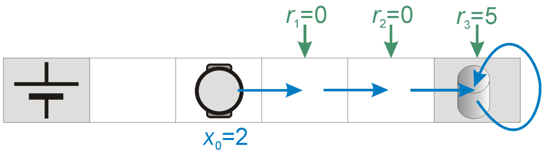
$$R^{\pi}(x_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k))$$

from any x_0

Discount factor $\gamma \in [0, 1)$:

- induces a “pseudo-horizon” for optimization
- bounds infinite sum
- encodes increasing uncertainty about the future
- helps convergence of algorithms

Cleaning robot: Return



Assume π always goes right

$$
\begin{aligned}
R^{\pi}(2) &= \gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + \gamma^3 0 + \gamma^4 0 + \ldots \\
&= \gamma^2 \cdot 5
\end{aligned}
$$

Because x_3 is terminal, all remaining rewards are 0

Value function

One of these two is used:

- **V-function** (state value) of policy π:

$$V^\pi(x_0) = R^\pi(x_0)$$

- **Q-function** (state-action value) of policy π:

$$Q^\pi(x_0, u_0) = \rho(x_0, u_0) + \gamma R^\pi(x_1)$$

(return after taking u_0 in x_0 and then following π)

Q-function

$$
\begin{aligned}
R^\pi(x_0) &= \sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k)) \\
&= \rho(x_0, \pi(x_0)) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, \pi(x_k)) \\
&= \rho(x_0, \pi(x_0)) + \gamma \sum_{k=0}^{\infty} \gamma^k \rho(x_{k+1}, \pi(x_{k+1})) \\
&= \rho(x_0, \pi(x_0)) + \gamma R^\pi(x_1)
\end{aligned}
$$

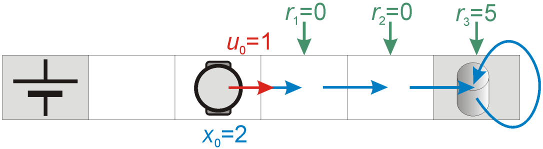
Q-function makes first action a free variable u_0:

$$
Q^\pi(x_0, u_0) = \rho(x_0, u_0) + \gamma R^\pi(x_1)
$$

Q-function (cont'd)

$$Q^{\pi}(x_0, u_0) = \rho(x_0, u_0) + \gamma R^{\pi}(x_1)$$

- First action in the sequence independent of policy
- Rest of the sequence follows the policy



- Q-function allows direct derivation of policy

Bellman equation

- Develop Q-function one step ahead:

$$
\begin{aligned}
Q^\pi(x_0, u_0) &= \rho(x_0, u_0) + \gamma R^\pi(x_1) \\
&= \rho(x_0, u_0) + \gamma[\rho(x_1, \pi(x_1)) + \gamma R^\pi(x_2)] \\
&= \rho(x_0, u_0) + \gamma Q^\pi(x_1, \pi(x_1))
\end{aligned}
$$

Remember: $x_1 = f(x_0, u_0)$

$\Rightarrow$ **Bellman equation for Q^π**

$$Q^\pi(x, u) = \rho(x, u) + \gamma Q^\pi(f(x, u), \pi(f(x, u)))$$

Bellman equation

- Develop Q-function one step ahead:

$$\begin{aligned} Q^\pi(x_0, u_0) &= \rho(x_0, u_0) + \gamma R^\pi(x_1) \\ &= \rho(x_0, u_0) + \gamma[\rho(x_1, \pi(x_1)) + \gamma R^\pi(x_2)] \\ &= \rho(x_0, u_0) + \gamma Q^\pi(x_1, \pi(x_1)) \end{aligned}$$

Remember: $x_1 = f(x_0, u_0)$

$\Rightarrow$ Bellman equation for Q^π

$$Q^\pi(x, u) = \rho(x, u) + \gamma Q^\pi(f(x, u), \pi(f(x, u)))$$

Optimal solution

- **Optimal Q-function**:

$$Q^* = \max_{\pi} Q^{\pi}$$

⇒ Greedy policy in Q^*:

$$\pi^*(x) = \arg\max_{u} Q^*(x, u)$$

is **optimal** (achieves maximal returns)

Bellman optimality equation (for Q^*)

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

Optimal solution

- **Optimal Q-function**:

$$Q^* = \max_\pi Q^\pi$$

$\Rightarrow$ Greedy policy in Q^*:

$$\pi^*(x) = \arg\max_u Q^*(x, u)$$

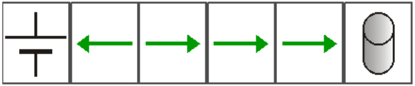
is **optimal** (achieves maximal returns)

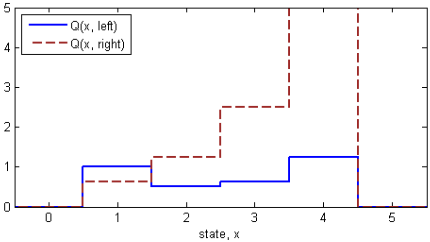
Bellman optimality equation (for Q^*)

$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

Cleaning robot: Optimal solution

Discount factor $\gamma = 0.5$





1. Reinforcement learning basics
2. Algorithms
   - Q-learning
   - SARSA
3. Accelerating RL

Off-policy online RL: Q-learning

Off-policy: **find** Q^*, use it to compute π^*

1. Take Bellman optimality equation at some (x, u):
$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$

2. Turn into **iterative update**:
$$Q(x, u) \leftarrow \rho(x, u) + \gamma \max_{u'} Q(f(x, u), u')$$

3. Instead of model f, ρ, use **transition sample** $(x_k, u_k, x_{k+1}, r_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u')$$
Note: $x_{k+1} = f(x_k, u_k)$, $r_{k+1} = \rho(x_k, u_k)$

Off-policy online RL: Q-learning

Off-policy: **find Q^***, use it to compute π^*

1. Take Bellman optimality equation at some (x, u):
$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$
2. Turn into **iterative update**:
$$Q(x, u) \leftarrow \rho(x, u) + \gamma \max_{u'} Q(f(x, u), u')$$
3. Instead of model f, ρ, use **transition sample** $(x_k, u_k, x_{k+1}, r_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u')$$
Note: $x_{k+1} = f(x_k, u_k)$, $r_{k+1} = \rho(x_k, u_k)$

Off-policy online RL: Q-learning

Off-policy: **find Q^***, use it to compute π^*

1. Take Bellman optimality equation at some (x, u):
$$Q^*(x, u) = \rho(x, u) + \gamma \max_{u'} Q^*(f(x, u), u')$$
2. Turn into **iterative update**:
$$Q(x, u) \leftarrow \rho(x, u) + \gamma \max_{u'} Q(f(x, u), u')$$
3. Instead of model f, ρ, use **transition sample** $(x_k, u_k, x_{k+1}, r_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u')$$
Note: $x_{k+1} = f(x_k, u_k), r_{k+1} = \rho(x_k, u_k)$

Q-learning (cont'd)

4. Finally, make update **incremental**:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \\ \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k) \right]$$

with learning rate $\alpha_k \in (0, 1]$.

The expression

$$r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)$$

is called the **temporal difference**.

Complete Q-learning algorithm

Q-learning

for every trial **do**
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$
$$[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$
  until terminal state
end for

Complete Q-learning algorithm

Q-learning

for every trial **do**
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    $$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right]$$
  until terminal state
end for

Exploration-exploitation tradeoff

- Essential condition for convergence to Q^*: all (x, u) pairs must be visited infinitely often

$\Rightarrow$ **Exploration** necessary: sometimes, choose actions randomly

- **Exploitation** of current knowledge is also necessary: sometimes, choose actions greedily:

$$u_k = \arg\max_{\bar{u}} Q(x_k, \bar{u})$$

Exploration-exploitation tradeoff crucial
for performance of online RL

Exploration-exploitation tradeoff

- Essential condition for convergence to Q^*:
  all (x, u) pairs must be visited infinitely often

$\Rightarrow$ **Exploration** necessary:
sometimes, choose actions randomly

- **Exploitation** of current knowledge is also necessary:
  sometimes, choose actions greedily:

$$u_k = \arg\max_{\bar{u}} Q(x_k, \bar{u})$$

> Exploration-exploitation tradeoff crucial
> for performance of online RL

Exploration-exploitation: ε-greedy strategy

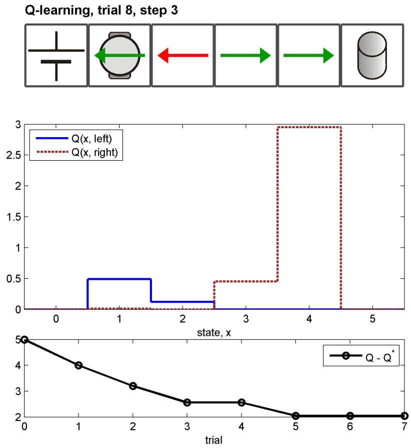
- Simple solution: **ε-greedy**

$$u_k = \begin{cases} \arg\max_{\bar{u}} Q(x_k, \bar{u}) & \text{with probability } (1 - \varepsilon_k) \\ \text{a random action} & \text{with probability } \varepsilon_k \end{cases}$$

- Exploration probability $\varepsilon_k \in (0, 1)$
is usually decreased over time

Cleaning robot: Q-learning demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$ (constant)
$x_0 = 2$ or 3 (randomly)



1 Reinforcement learning basics

2 Algorithms
- Q-learning
- SARSA

3 Accelerating RL

On-policy online RL: SARSA

On-policy: **find Q^π**, improve π, repeat

Similar to Q-learning:

1. Take Bellman equation for Q^π, at some (x, u):
$$Q^\pi(x, u) = \rho(x, u) + \gamma Q^\pi(f(x, u), \pi(f(x, u)))$$
2. Turn into iterative update:
$$Q(x, u) \leftarrow \rho(x, u) + \gamma Q(f(x, u), \pi(f(x, u)))$$
3. Use sample $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note: $u_{k+1} = \pi(f(x_k, u_k))$, π = **policy being followed**

On-policy online RL: SARSA

On-policy: **find Q^π**, improve π, repeat

Similar to Q-learning:

1. Take Bellman equation for Q^π, at some (x, u):
$$Q^\pi(x, u) = \rho(x, u) + \gamma Q^\pi(f(x, u), \pi(f(x, u)))$$
2. Turn into iterative update:
$$Q(x, u) \leftarrow \rho(x, u) + \gamma Q(f(x, u), \pi(f(x, u)))$$
3. Use sample $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note: $u_{k+1} = \pi(f(x_k, u_k))$, π = **policy being followed**

On-policy online RL: SARSA

On-policy: **find Q^π**, improve π, repeat

Similar to Q-learning:

1. Take Bellman equation for Q^π, at some (x, u):
$$Q^\pi(x, u) = \rho(x, u) + \gamma Q^\pi(f(x, u), \pi(f(x, u)))$$
2. Turn into iterative update:
$$Q(x, u) \leftarrow \rho(x, u) + \gamma Q(f(x, u), \pi(f(x, u)))$$
3. Use sample $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ at each step k:
$$Q(x_k, u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note: $u_{k+1} = \pi(f(x_k, u_k))$, π **= policy being followed**

SARSA (cont'd)

4. Make update incremental:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)\right]$$

Note that

$$r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)$$

is the **temporal difference** here

$(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ =
(**S**tate, **A**ction, **R**eward, **S**tate, **A**ction) = **SARSA**

Complete SARSA algorithm

SARSA

for every trial **do**
  initialize x_0, choose initial action u_0
  repeat for each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose **next** action u_{k+1}
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$
    $[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$
  until terminal state
end for

Exploration-exploitation in SARSA

- For convergence—besides infinite exploration—SARSA requires **policy to eventually become greedy**
- E.g., ε-greedy

$$u_k = \begin{cases} \arg\max_{\bar{u}} Q(x_k, \bar{u}) & \text{with probability } (1 - \varepsilon_k) \\ \text{a random action} & \text{with probability } \varepsilon_k \end{cases}$$

with $\lim_{k\to\infty} \varepsilon_k = 0$

- Greedy actions $\Rightarrow$ policy implicitly improved!
(Recall on-policy: find Q^π, **improve** π, repeat)

Exploration-exploitation in SARSA

- For convergence—besides infinite exploration—SARSA requires **policy to eventually become greedy**
- E.g., ε-greedy

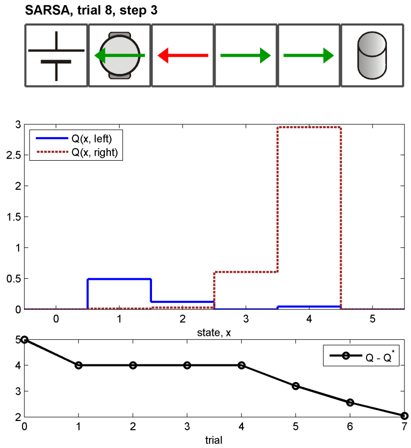
$$u_k = \begin{cases} \arg\max_{\bar{u}} Q(x_k, \bar{u}) & \text{with probability } (1 - \varepsilon_k) \\ \text{a random action} & \text{with probability } \varepsilon_k \end{cases}$$

with $\lim_{k \to \infty} \varepsilon_k = 0$

- Greedy actions $\Rightarrow$ policy implicitly improved!
(Recall on-policy: find Q^π, **improve** π, repeat)

Cleaning robot: SARSA demo

Parameters like Q-learning: $\alpha = 0.2$, $\varepsilon = 0.3$ (constant)
$x_0 = 2$ or 3 (randomly)



Accelerating RL

In practice, transition data costs:

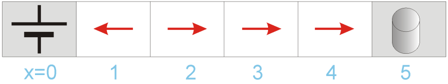
- time
- profits (suboptimal performance due to exploration)
- process wear & tear

Fast RL = **use data efficiently**

(computational demands are secondary)

Cleaning robot without acceleration

Evaluate the policy π with $\gamma = 0.5, \alpha = 0.2$



First encounter with $x = 3$ gives:

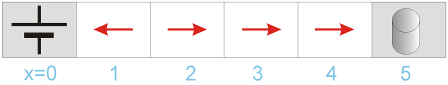
1. $Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)] = 0$
2. $Q^\pi(4,1) \leftarrow Q^\pi(4,1) + \alpha[5 + \gamma 0 - Q^\pi(4,1)] = 1$

Next encounter with $x = 3$ gives:

1. $Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)] = \alpha\gamma Q^\pi(4,1) = 0.1$
2. $Q^\pi(4,1) \leftarrow Q^\pi(4,1) + \alpha[5 + \gamma 0 - Q^\pi(4,1)] = 1.8$

Cleaning robot without acceleration

Evaluate the policy π with $\gamma = 0.5, \alpha = 0.2$



First encounter with $x = 3$ gives:

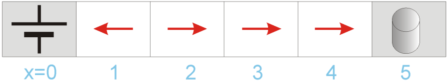
1. $Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)] = 0$
2. $Q^\pi(4,1) \leftarrow Q^\pi(4,1) + \alpha[5 + \gamma 0 - Q^\pi(4,1)] = 1$

Next encounter with $x = 3$ gives:

1. $Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)] = \alpha\gamma Q^\pi(4,1) = 0.1$
2. $Q^\pi(4,1) \leftarrow Q^\pi(4,1) + \alpha[5 + \gamma 0 - Q^\pi(4,1)] = 1.8$

Cleaning robot without acceleration

Evaluate the policy π with $\gamma = 0.5, \alpha = 0.2$



First encounter with $x = 3$ gives:

1. $Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)] = 0$
2. $Q^\pi(4,1) \leftarrow Q^\pi(4,1) + \alpha[5 + \gamma 0 - Q^\pi(4,1)] = 1$

Next encounter with $x = 3$ gives:

1. $Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)] = \alpha\gamma Q^\pi(4,1) = 0.1$
2. $Q^\pi(4,1) \leftarrow Q^\pi(4,1) + \alpha[5 + \gamma 0 - Q^\pi(4,1)] = 1.8$

Cleaning robot without acceleration (cont'd)

Change in $Q^\pi(4,1)$ obviously influences $Q^\pi(3,1)$ as

$$Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)]$$

always holds!

Main idea

- Do not wait for state $x = 3$ to pop up again for the update.
- Update $Q^\pi(3,1)$ immediately when $Q^\pi(4,1)$ and/or other successor states are updated.

Cleaning robot without acceleration (cont'd)

Change in $Q^\pi(4,1)$ obviously influences $Q^\pi(3,1)$ as

$$Q^\pi(3,1) \leftarrow Q^\pi(3,1) + \alpha[0 + \gamma Q^\pi(4,1) - Q^\pi(3,1)]$$

always holds!

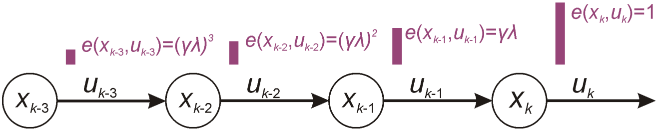
Main idea

- Do not wait for state $x = 3$ to pop up again for the update.
- Update $Q^\pi(3,1)$ immediately when $Q^\pi(4,1)$ and/or other successor states are updated.

1. Reinforcement learning basics
2. Algorithms
3. Accelerating RL
   - Eligibility traces
   - Experience replay

Eligibility traces

- Leave decaying **trace** along state-action trajectory:



- $\lambda \in [0, 1]$ decay rate, γ discount factor
- Implementation:

$e(x, u) \leftarrow 0 \qquad \forall x, u$
for each step k **do**
  $e(x, u) \leftarrow \lambda\gamma e(x, u) \qquad \forall x, u$
  $e(x_k, u_k) \leftarrow 1$
end for

Q(λ)-learning

- Recall basic Q-learning only updates $Q(x_k, u_k)$:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right]$$

- Q(λ)-learning updates **all eligible pairs**:

$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right] \quad \forall x, u$$

- Note: exploratory actions break causality
  $\Rightarrow$ reset eligibility trace to 0

Q(λ)-learning

- Recall basic Q-learning only updates $Q(x_k, u_k)$:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right]$$

- Q(λ)-learning updates **all eligible pairs**:

$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right] \qquad \forall x, u$$

- Note: exploratory actions break causality
  $\Rightarrow$ reset eligibility trace to 0

Q(λ)-learning

- Recall basic Q-learning only updates $Q(x_k, u_k)$:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \\ [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$

- Q(λ)-learning updates **all eligible pairs**:

$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot \\ [r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)] \qquad \forall x, u$$

- Note: exploratory actions break causality
  $\Rightarrow$ reset eligibility trace to 0

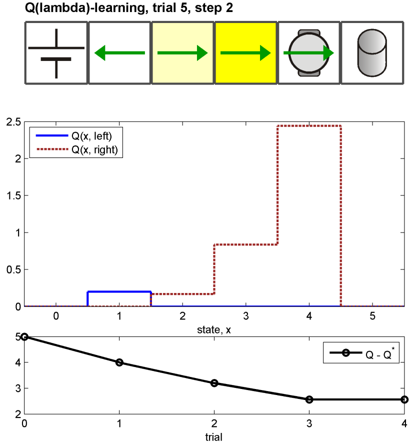
Complete Q(λ)-learning algorithm

Q(λ)-learning

for every trial **do**
  $e(x, u) \leftarrow 0 \qquad \forall x, u$
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    if u_k exploratory **then** $e(x, u) \leftarrow 0 \qquad \forall x, u$
    else $e(x, u) \leftarrow \lambda\gamma e(x, u) \qquad \forall x, u$
    end if
    $e(x_k, u_k) \leftarrow 1$
    $Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot$
      $[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)] \quad \forall x, u$
  until terminal state
end for

Cleaning robot: Q(λ)-learning demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$ (like basic Q-learning), $\lambda = 0.5$
$x_0 = 2$ or 3 (randomly)



SARSA(λ)

Similar to Q-learning:

- Basic SARSA:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

- SARSA(λ)-learning:

$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot \mathbf{e(x, u)} \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)] \qquad \forall x, u$$

- SARSA on-policy, including exploration
  $\Rightarrow$ exploratory actions not a problem

SARSA(λ)

Similar to Q-learning:

- Basic SARSA:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

- SARSA(λ)-learning:

$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot \mathbf{e(x, u)} \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)] \qquad \forall x, u$$

- SARSA on-policy, including exploration
  $\Rightarrow$ exploratory actions not a problem

SARSA(λ)

Similar to Q-learning:

- Basic SARSA:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

- SARSA(λ)-learning:

$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot \mathbf{e(x, u)} \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)] \qquad \forall x, u$$

- SARSA on-policy, including exploration
  $\Rightarrow$ exploratory actions not a problem

Complete SARSA(λ) algorithm

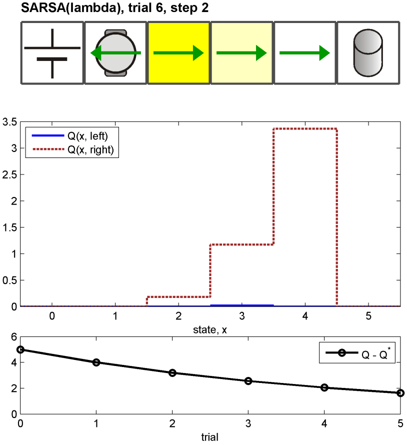
SARSA(λ)

for every trial **do**
- $e(x,u) \leftarrow 0 \qquad \forall x,u$
- initialize x_0, choose initial action u_0
- **repeat** for each step k
  - apply u_k, measure x_{k+1}, receive r_{k+1}
  - choose next action u_{k+1}
  - $e(x,u) \leftarrow \lambda\gamma e(x,u) \qquad \forall x,u$
  - $e(x_k,u_k) \leftarrow 1$
  - $$Q(x,u) \leftarrow Q(x,u) + \alpha_k \cdot e(x,u) \cdot \left[r_{k+1} + \gamma Q(x_{k+1},u_{k+1}) - Q(x_k,u_k)\right] \quad \forall x,u$$
- **until** terminal state

end for

Cleaning robot: SARSA(λ) demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$ (like basic SARSA), $\lambda = 0.5$
$x_0 = 2$ or 3 (randomly)



Effects of eligibility traces

- Accelerates learning: fewer trials to convergence
- However: too large λ can make algorithm settle on suboptimal solution!

1 Reinforcement learning basics

2 Algorithms

3 Accelerating RL
- Eligibility traces
- Experience replay

Experience replay (ER)

- Store each transition sample $(x_k, u_k, x_{k+1}, r_{k+1})$ into a database
- At every step, **replay** N transitions from the database
- Improvement: replay most informative samples first: **prioritized sweeping** (not considered here)

Experience replay (ER)

- Store each transition sample $(x_k, u_k, x_{k+1}, r_{k+1})$ into a database
- At every step, **replay** N transitions from the database
- Improvement: replay most informative samples first: **prioritized sweeping** (not considered here)

ER Q-learning

Q-learning with experience replay

for every trial **do**
  initialize x_0
  repeat for each step k
    take action u_k
    measure x_{k+1}, receive r_{k+1}
    $$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot \left[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)\right]$$
    add $(x_k, u_k, x_{k+1}, r_{k+1})$ to database
    ReplayExperience
  until terminal state
end for

ER Q-learning (cont'd)

ReplayExperience

loop N times

retrieve a sample (x, u, x', r) from database

$$Q(x, u) \leftarrow Q(x, u) + \alpha \cdot \left[r + \gamma \max_{u'} Q(x', u') - Q(x, u)\right]$$

end loop

Summary

- **Reinforcement learning** =
  optimal, adaptive, model-free control
- Principle: reward signal as performance feedback
- Inspired from human and animal learning,
  but solid mathematical foundation
- Classical RL: small, discrete X and U (this lecture)

A final look at the algorithms

	Off-policy	On-policy
Basic RL	Q-learning Param: γ, α_k, ε_k	SARSA Param: γ, α_k, ε_k
RL with eligibility traces	Q(λ)-learning Param: γ, α_k, ε_k, λ	SARSA(λ) Param: γ, α_k, ε_k, λ

Typical parameter values:

- γ 0.9 or larger
- α_k under 0.5 or diminishing schedule
- ε_k around 0.1 or diminishing schedule
- λ between 0.5 and 0.9

A final look at the algorithms

	Off-policy	On-policy
Basic RL	Q-learning Param: γ, α_k, ε_k	SARSA Param: γ, α_k, ε_k
RL with eligibility traces	Q(λ)-learning Param: γ, α_k, ε_k, λ	SARSA(λ) Param: γ, α_k, ε_k, λ

Typical parameter values:

- γ 0.9 or larger
- α_k under 0.5 or diminishing schedule
- ε_k around 0.1 or diminishing schedule
- λ between 0.5 and 0.9

Next lecture

Still to address:

- Continuous state and action spaces X, U
- More algorithms: actor-critic, model-learning, etc.

Part II – RL using function approximation