Basics of reinforcement learning

Lucian Buşoniu

TMLSS, 20 July 2018

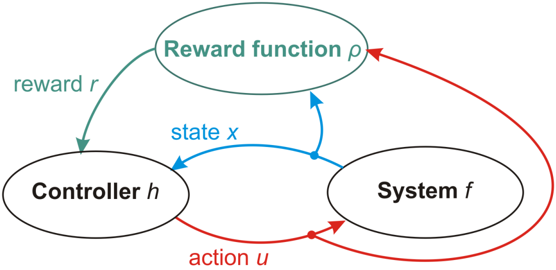
Main idea of reinforcement learning (RL)

Learn a sequential decision policy

to optimize the cumulative performance

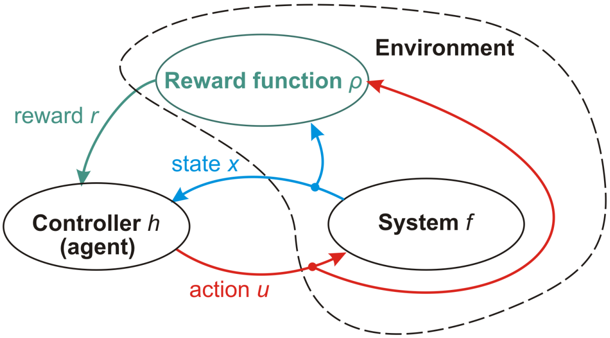
of an unknown system

RL principle



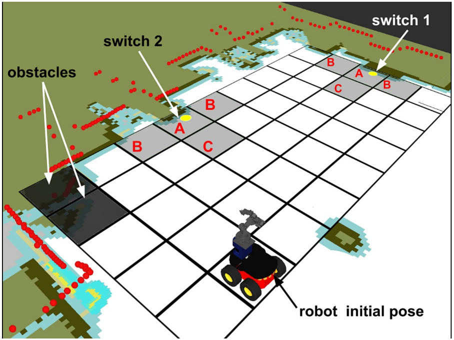
- Interact with system: measure states, apply actions
- Performance feedback in the form of rewards
- Inspired by human and animal learning

RL principle: AI view



- Agent embedded in an environment that feeds back states and rewards

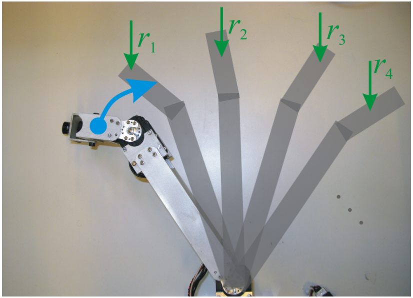
Example: Domestic robot



A domestic robot ensures light switches are off
Abstractization to high-level control (physical actions implemented by low-level controllers)

- States: grid coordinates, switch states
- Actions: movements NSEW, toggling switch
- Rewards: when switches toggled on→off

Example: Robot arm

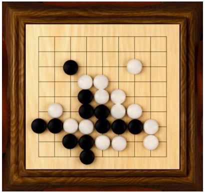


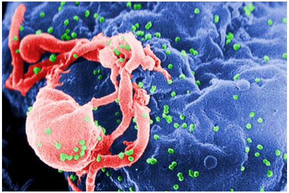
Low-level control

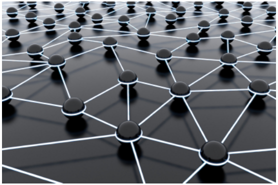
- States: link angles and angular velocities
- Actions: motor voltages
- Rewards: e.g. to reach a desired configuration, give larger rewards as robot gets closer to it

Many other applications

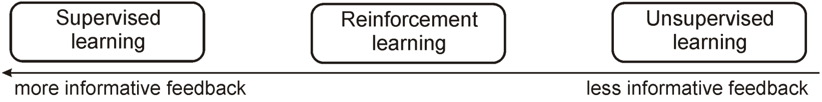
Artificial intelligence, control, medicine, multiagent systems, economics etc.







RL on the machine learning spectrum

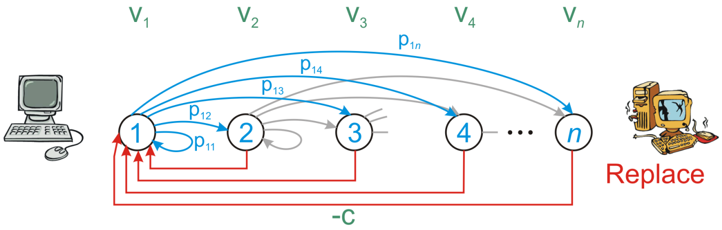


- Supervised: for each training sample, **correct output** known
- Unsupervised: only input samples, **no outputs**; find patterns in the data
- Reinforcement: correct actions not available, **only rewards**

But note: RL finds **dynamical optimal control**!

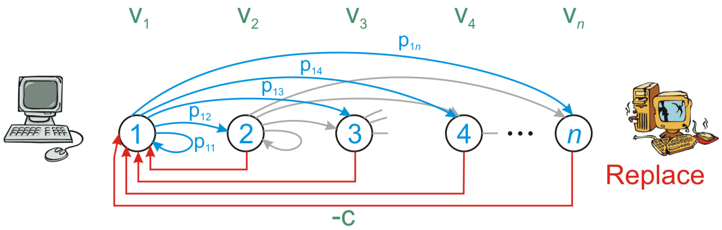
1. Introduction
2. Markov decision process & optimal solution
3. Dynamic programming, DP
4. Monte Carlo, MC
5. Exploration basics
6. Temporal differences, TD
7. Improving data efficiency

Example: Machine replacement



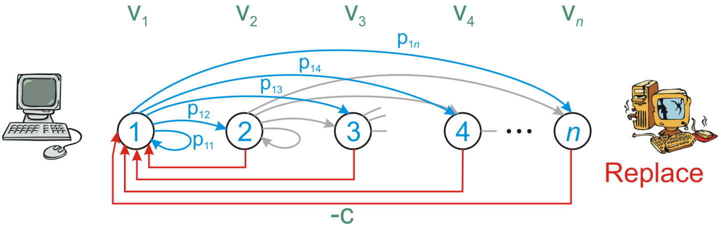
- Consider machine with n different wear levels
  1 = perfect working order, n = fully degraded
- Produces revenue v_i when operating at level i
- Stohastic wear: level i increases to $j > i$ with probas p_{ij}, stays i with $p_{ii} = 1 - p_{i,i+1} - ... - p_{i,n}$
- Machine can be replaced at any time (assumed instantaneously), paying cost c

Machine replacement: States and actions



- State x = wear level,
  state space $X = \{1, 2, \dots, n\}$
- Action u = whether to Wait or Replace,
  action space: $U = \{\mathrm{W}, \mathrm{R}\}$

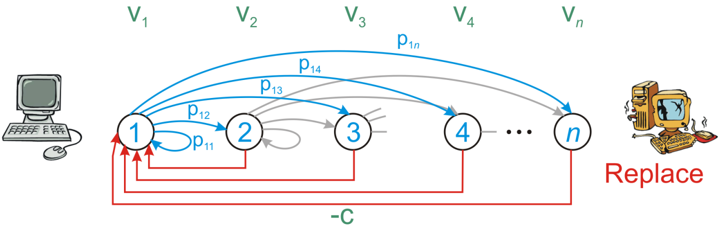
Machine replacement: Transition function



Transition function $f(x, u, x')$ gives the probability of reaching state x' after applying action u in state x:

$$f(x = i, u, x' = j) = \begin{cases} p_{ij} & \text{if } u = \text{W and } i \leq j \\ 1 & \text{if } u = \text{R and } j = 1 \\ 0 & \text{otherwise} \end{cases}$$

Machine replacement: Reward function



Reward function $\rho(x, u, x')$ gives reward resulting from transitioning from x to x' after applying u:

$$\rho(x = i, u, x' = j) = \begin{cases} v_i & \text{if } u = \text{W} \\ -c + v_1 & \text{if } u = \text{R} \end{cases}$$

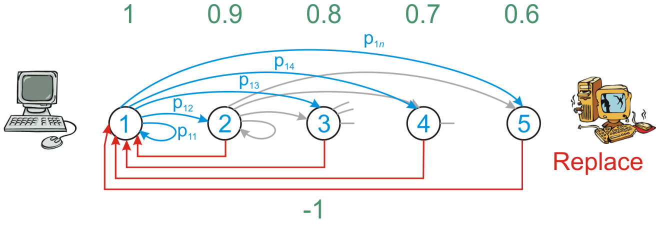
General case: Markov decision process

Markov decision process (MDP)

1. State space X
2. Action space U
3. Transition function $f(x, u, x')$, $\quad f : X \times U \times X \rightarrow [0, 1]$
4. Reward function $\rho(x, u, x')$, $\quad \rho : X \times U \times X \rightarrow \mathbb{R}$

Some MDPs have terminal states (e.g. success, failure), that once reached cannot be left and provide no additional reward

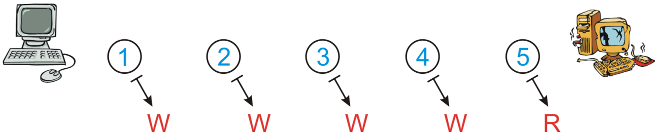
Machine replacement: Specific example



- $n = 5$ wear levels
- Revenue: $v_1 = 1, v_2 = 0.9, ..., v_5 = 0.6$
- Cost of new machine: $c = 1$
- Wear increase probabilities:

$$[p_{ij}] = \begin{bmatrix} 0.6 & 0.3 & 0.1 & 0 & 0 \\ 0 & 0.6 & 0.3 & 0.1 & 0 \\ 0 & 0 & 0.6 & 0.3 & 0.1 \\ 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 1.0 \end{bmatrix}$$

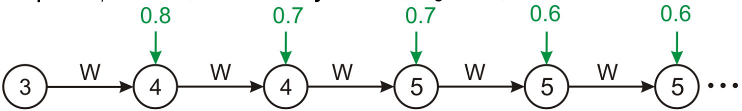
Control policy



- Control policy $h : X \to U$: maps states x to actions u
- Example for machine replacement: $h(1) = \ldots = h(4) = \text{W}$, $h(5) = \text{R}$

Return and objective

Example: $\gamma = 0.9$, h = always wait, $x_0 = 4$, and trial:



$$\gamma^0 r_1 + \gamma^1 r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots = 0.9^0 \cdot 0.8 + 0.9^1 \cdot 0.7 + \\ + 0.9^2 \cdot 0.7 + 0.9^3 \cdot 0.6 + 0.9^4 \cdot 0.6 + \ldots = 6.3710$$

Objective

Find h that maximizes from any x_0
the expected return under the stochastic transitions:

$$R^h(x_0) = \mathrm{E}_{x_1, x_2, \ldots} \left\{ \sum_{k=0}^{\infty} \gamma^k \rho(x_k, h(x_k), x_{k+1}) \right\}$$

Note: There are also other types of return!

Discount factor

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

- represents an increasing uncertainty about the future
- bounds the infinite sum (assuming rewards bounded)
- helps the convergence of algorithms

To choose γ, **trade-off** between:

1. Long-term quality of the solution (large γ)
2. Simplicity of the problem (small γ)

In practice, γ should be sufficiently large so as not to ignore important later rewards

Q-function

Q-function of a policy h is the expected return achieved by executing u_0 in x_0 and then following h

$$Q^h(x_0, u_0) = \mathrm{E}_{x_1} \left\{ \rho(x_0, u_0, x_1) + \gamma R^h(x_1) \right\}$$

Q^h measures the quality of state-action pairs

Bellman equation of Q^h

Go one step further in the equation:

$$\begin{aligned} Q^h(x_0, u_0) &= \mathrm{E}_{x_1}\left\{\rho(x_0, u_0, x_1) + \gamma R^h(x_1)\right\} \\ &= \mathrm{E}_{x_1}\left\{\rho(x_0, u_0, x_1) + \gamma \mathrm{E}_{x_2}\left\{\rho(x_1, h(x_1), x_2) + \gamma R^h(x_2)\right\}\right\} \\ &= \mathrm{E}_{x_1}\left\{\rho(x_0, u_0, x_1) + \gamma Q^h(x_1, h(x_1))\right\} \end{aligned}$$

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

$$\begin{aligned} Q^h(x, u) &= \mathrm{E}_{x'}\left\{\rho(x, u, x') + \gamma Q^h(x', h(x'))\right\} \\ &= \sum_{x'} f(x, u, x')\left[\rho(x, u, x') + \gamma Q^h(x', h(x'))\right] \end{aligned}$$

Optimal solution and Bellman optimality equation

- **Optimal Q-function**: $Q^* = \max_h Q^h$

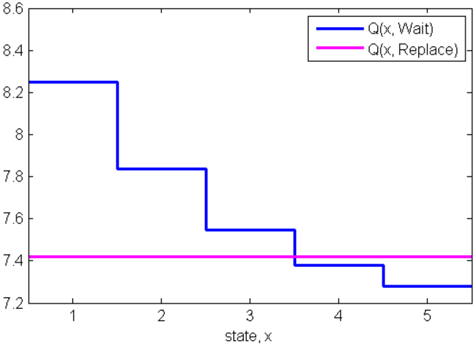
$\Rightarrow$ "Greedy" policy in Q^*: $h^*(x) = \arg\max_u Q^*(x, u)$
is **optimal**, i.e. achieves maximal returns
(if multiple actions maximize, break ties arbitrarily)

Bellman optimality equation (for Q^*)

$$Q^*(x, u) = \mathrm{E}_{x'} \left\{ \rho(x, u, x') + \gamma \max_{u'} Q^*(x', u') \right\}$$
$$= \sum_{x'} f(x, u, x') \left[\rho(x, u, x') + \gamma \max_{u'} Q^*(x', u') \right]$$

Machine replacement: Optimal solution

Discount factor $\gamma = 0.9$



Up next:

Algorithms to find the optimal solution

Algorithm landscape

By model usage:

- Model-based: f, ρ known a priori
- Model-free: f, ρ unknown
- Model-learning: f, ρ learned from data

Model-based usually called dynamic programming (DP); needed as a stepping stone to RL

By interaction level:

- Offline: algorithm runs in advance
- Online: algorithm runs with the system

We focus on exact case: x, u small number of discrete values, so we can exactly represent solutions. In practice, function approximation often needed – covered in Doina's talk

1 Introduction

2 Markov decision process & optimal solution

3 Dynamic programming, DP

4 Monte Carlo, MC

5 Exploration basics

6 Temporal differences, TD

7 Improving data efficiency

Q-iteration

Transforms Bellman optimality equation:

$$Q^*(x,u) = \sum_{x'} f(x,u,x') \left[\rho(x,u,x') + \gamma \max_{u'} Q^*(x',u') \right]$$

into an **iterative procedure**:

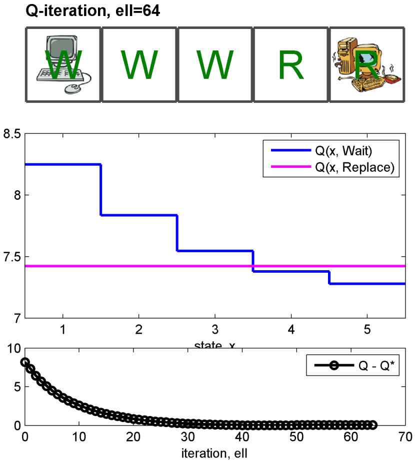
Q-iteration

repeat at each iteration ℓ
  for all x, u **do**
    $Q_{\ell+1}(x,u) \leftarrow \sum_{x'} f(x,u,x') \big[\rho(x,u,x') + \gamma \max_{u'} Q_\ell(x',u') \big]$
  end for
until convergence to Q^*
Once Q^* available: $h^*(x) = \arg\max_u Q^*(x,u)$

Q-iteration belongs to the class of value iteration algorithms

Machine replacement: Q-iteration demo

Discount factor $\gamma = 0.9$



Machine replacement: Q-iteration demo

$$Q_{\ell+1}(x,u) \leftarrow \sum_{x'} f(x,u,x')[\rho(x,u,x') + \gamma \max_{u'} Q_\ell(x',u')]$$

	$x = 1$	$x = 2$	$x = 3$	$x = 4$	$x = 5$
Q_0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
Q_1	1 ; 0	0.9 ; 0	0.8 ; 0	0.7 ; 0	0.6 ; 0
Q_2	1.86 ; 0.9	1.67 ; 0.9	1.48 ; 0.9	1.3 ; 0.9	1.14 ; 0.9
Q_3	2.58 ; 1.67	2.31 ; 1.67	2.05 ; 1.67	1.83 ; 1.67	1.63 ; 1.67
Q_4	3.2 ; 2.33	2.87 ; 2.33	2.55 ; 2.33	2.3 ; 2.33	2.1 ; 2.33
...	...	...	...	...	...
Q_{64}	8.25 ; 7.42	7.84 ; 7.42	7.55 ; 7.42	7.38 ; 7.42	7.28 ; 7.42
Q_{65}	8.25 ; 7.42	7.84 ; 7.42	7.55 ; 7.42	7.38 ; 7.42	7.28 ; 7.42
h^*	W	W	W	R	R

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

Policy iteration

Policy iteration

initialize policy h_0
repeat at each iteration ℓ
  1: policy evaluation: find Q^{h_ℓ}
  2: policy improvement:
    $h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x, u)$
until convergence to h^*

Iterative policy evaluation

Similarly to Q-iteration, transforms Bellman equation for Q^h:

$$Q^h(x,u) = \sum_{x'} f(x,u,x') \left[\rho(x,u,x') + \gamma Q^h(x', h(x'))\right]$$

into an iterative procedure:

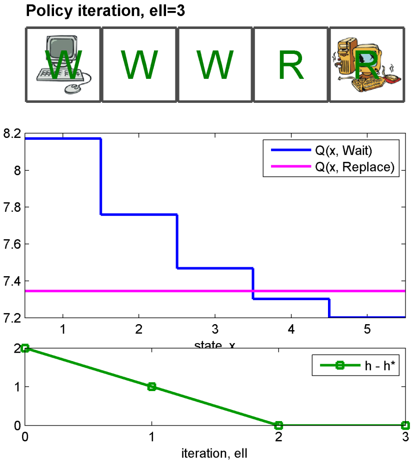
Iterative policy evaluation

repeat at each iteration τ
  for all x, u **do**
    $Q_{\tau+1}(x,u) \leftarrow \sum_{x'} f(x,u,x') \big[\rho(x,u,x') + \gamma Q_\tau(x', h(x'))\big]$
  end for
until convergence to Q^h

(other options exist, e.g. solving the linear system)

Machine replacement: policy iteration demo

Discount factor $\gamma = 0.9$



Machine replacement: policy iteration

$$Q_{\tau+1}(x,u) \leftarrow \sum_{x'} f(x,u,x')[\rho(x,u,x') + \gamma Q_\tau(x', h(x'))]$$

$$h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x,u)$$

	$x=1$	$x=2$	$x=3$	$x=4$	$x=5$
h_0	W	W	W	W	W
Q_0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
Q_1	1 ; 0	0.9 ; 0	0.8 ; 0	0.7 ; 0	0.6 ; 0
Q_2	1.86 ; 0.9	1.67 ; 0.9	1.48 ; 0.9	1.3 ; 0.9	1.14 ; 0.9
Q_3	2.58 ; 1.67	2.31 ; 1.67	2.05 ; 1.67	1.83 ; 1.67	1.63 ; 1.67
...	...	...	...	...	...
Q_{39}	7.51 ; 6.75	6.95 ; 6.75	6.49 ; 6.75	6.17 ; 6.75	5.9 ; 6.75
Q_{40}	7.52 ; 6.75	6.96 ; 6.75	6.5 ; 6.75	6.18 ; 6.75	5.91 ; 6.75
h_1	W	W	R	R	R

...algorithm continues...

Machine replacement: policy iteration (cont'd)

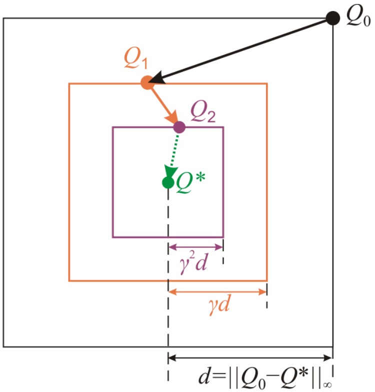
	$x = 1$	$x = 2$	$x = 3$	$x = 4$	$x = 5$
h_1	W	W	R	R	R
Q_0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
...	...	...	...	...	...
Q_{43}	8.01 ; 7.2	7.57 ; 7.2	7.27 ; 7.2	7.17 ; 7.2	7.07 ; 7.2
h_2	W	W	W	R	R
Q_0	0 ; 0	0 ; 0	0 ; 0	0 ; 0	0 ; 0
...	...	...	...	...	...
Q_{43}	8.17 ; 7.35	7.76 ; 7.35	7.47 ; 7.35	7.3 ; 7.35	7.2 ; 7.35
h_3	W	W	W	R	R

Convergence of Q-iteration

- Each iteration a contraction with factor γ in ∞-norm:

$$\|Q_{\ell+1} - Q^*\|_\infty \leq \gamma \|Q_\ell - Q^*\|_\infty$$

$\Rightarrow$ Q-iteration **monotonically converges** to Q^*, **with convergence rate** γ $\Rightarrow$ γ helps convergence



Stopping condition of Q-iteration

- Convergence to Q^* only guaranteed asymptotically, as $\ell \rightarrow \infty$
- In practice, algorithm can be stopped when:

$$\|Q_{\ell+1} - Q_\ell\| \leq \varepsilon_{\text{qiter}}$$

Convergence of policy iteration

Policy evaluation component – like Q-iteration:

- Iterative policy evaluation contraction with factor γ

$\Rightarrow$ **monotonic convergence** to Q^h, with rate γ

Complete policy iteration algorithm:

- Policy is either improved or already optimal
- But the maximum number of improvements is finite! ($|U|^{|X|}$)

$\Rightarrow$ **convergence** to h^* in a finite number of iterations

Stopping conditions of policy iteration

In practice:

- Policy evaluation can be stopped when:

$$\|Q_{\tau+1} - Q_\tau\| \leq \varepsilon_{\text{peval}}$$

- Policy iteration can be stopped when:

$$\|h_{\ell+1} - h_\ell\| \leq \varepsilon_{\text{piter}}$$

- Note: $\varepsilon_{\text{piter}}$ can be taken 0!

Q-iteration vs. policy iteration

Number of iterations to convergence

- Q-iteration $>$ policy iteration

Complexity

- one iteration of Q-iteration
  $>$ one iteration of iterative policy evaluation
- complete Q-iteration **???** complete policy iteration

1 Introduction

2 Markov decision process & optimal solution

3 Dynamic programming, DP

4 Monte Carlo, MC

5 Exploration basics

6 Temporal differences, TD

7 Improving data efficiency

Algorithm landscape

By model usage:

- Model-based: f, ρ known a priori
- Model-free: f, ρ unknown
- Model-learning: f, ρ learned from data

By interaction level:

- Offline: algorithm runs in advance
- Online: algorithm runs with the system

We move to online RL for the remainder of the talk

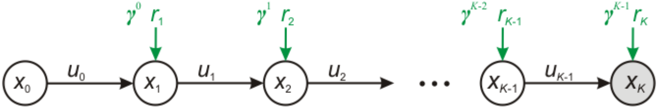
Policy evaluation change

To find Q^h:

- So far: model-based policy evaluation
- Reinforcement learning: model not available!
- Learn Q^h from data obtained by
  online interaction with the system

Monte Carlo policy evaluation

Recall: $Q^h(x_0, u_0) = \mathrm{E}_{x_1}\left\{\rho(x_0, u_0, x_1) + \gamma R^h(x_1)\right\}$



- Trial from (x_0, u_0) to x_K using $u_1 = h(x_1)$, $u_2 = h(x_2)$, etc.
- x_K must be terminal (assumed further) or K large enough

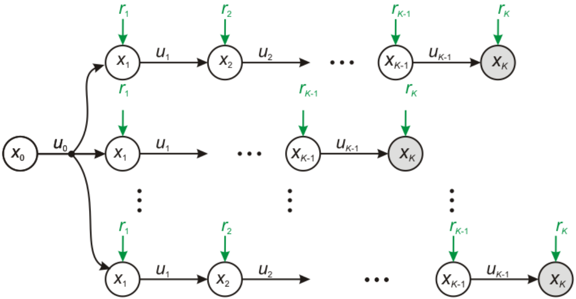
$\Rightarrow$ Return along trial provides a sample of $Q^h(x_0, u_0)$:

$$\sum_{j=0}^{K-1} \gamma^j r_{j+1}$$

- Furthermore, sample of $Q^h(x_k, u_k)$ at each step k:

$$Q^h(x_k, u_k) = \sum_{j=k}^{K-1} \gamma^{j-k} r_{j+1}$$

Monte Carlo policy evaluation (cont'd)



- To learn the expected value, run N trajectories (often called roll-outs)
- Estimated Q-value = **average** of the returns, e.g.

$$Q^h(x_0, u_0) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=0}^{K_i - 1} \gamma^j r_{i,j+1}$$

Monte Carlo policy iteration

Monte Carlo policy iteration

for each iteration ℓ **do**
  run N trials applying h_ℓ
  reset accumulator $A(x, u)$, counter $C(x, u)$ to 0
  for each step k of each trial i **do**
    $A(x_k, u_k) \leftarrow A(x_k, u_k) + \sum_{j=k}^{K_i-1} \gamma^{j-k} r_{i,j+1}$ (return)
    $C(x_k, u_k) \leftarrow C(x_k, u_k) + 1$
  end for
  $Q^{h_\ell}(x, u) \leftarrow A(x, u) / C(x, u)$
  $h_{\ell+1}(x) \leftarrow \arg\max_u Q^{h_\ell}(x, u)$
end for

Need for exploration

$$Q^h(x, u) \leftarrow A(x, u)/\mathbf{C(x, u)}$$

How to ensure $C(x, u) > 0$ – **information** about each (x, u)?

1. Select representative initial states x_0
2. Actions:
   u_0 representative, sometimes different from $h(x_0)$
   and in addition, perhaps:
   u_k representative, sometime different from $h(x_k)$

Exploration-exploitation

- **Exploration** needed:
  actions different from the current policy
- **Exploitation** of current knowledge also needed:
  current policy must be applied

Exploration-exploitation dilemma
– essential in all RL algorithms

(not just in MC)

1 Introduction

2 Markov decision process & optimal solution

3 Dynamic programming, DP

4 Monte Carlo, MC

5 Exploration basics

6 Temporal differences, TD

7 Improving data efficiency

ε-greedy strategy

- Simple solution to the exploration-exploitation dilemma: **ε-greedy**

$$u_k = \begin{cases} h(x_k) = \arg\max_u Q(x_k, u) & \text{with probability } (1 - \varepsilon_k) \\ \text{a uniformly random action} & \text{w.p. } \varepsilon_k \end{cases}$$

- Exploration probability $\varepsilon_k \in (0, 1)$ usually decreased over time
- Main disadvantage: when exploring, actions are fully random, leading to poor performance

Softmax strategy

- Action selection:

$$u_k = u \text{ w.p. } \frac{e^{Q(x_k,u)/\tau_k}}{\sum_{u'} e^{Q(x_k,u')/\tau_k}}$$

  where $\tau_k > 0$ is the **exploration temperature**
- Taking $\tau \to 0$, greedy selection recovered; $\tau \to \infty$ gives uniform random
- Compared to ε-greedy, better actions are more likely to be applied even when exploring

Bandit-based exploration

At single state, exploration modeled as **multi-armed bandit**:

- Action j = arm with reward distribution ρ_j, expectation μ_j
- Best arm (optimal action) has expected value μ^*
- At step k, we pull arm (try action) j_k, getting $r_k \sim \rho_{j_k}$
- **Objective:** After n pulls, small regret: $\sum_{k=1}^{n} \mu^* - \mu_{j_k}$

UCB algorithm

Popular algorithm: after n steps, pick arm with largest **upper confidence bound**:

$$b(j) = \hat{\mu}_j + \sqrt{\frac{3 \log n}{2 n_j}}$$

where:

- $\hat{\mu}_j$ = mean of rewards observed for arm j so far
- n_j = how many times arm j was pulled

These are only a few simple methods, many others exist, e.g. Bayesian exploration, intrinsic rewards etc.

1 Introduction

2 Markov decision process & optimal solution

3 Dynamic programming, DP

4 Monte Carlo, MC

5 Exploration basics

6 Temporal differences, TD

7 Improving data efficiency

DP perspective

1. Start from policy evaluation:
$$Q_{\tau+1}(x,u) \leftarrow \sum_{x'} f(x,u,x')[\rho(x,u,x') + \gamma Q_\tau(x', h(x'))]$$

2. Instead of model, use **transition sample** at each step k, $(x_k, u_k, x_{k+1}, r_{k+1}, u_{k+1})$:
$$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma Q(x_{k+1}, u_{k+1})$$
Note:
$x_{k+1} \sim f(x_k, u_k, \cdot), r_{k+1} = \rho(x_k, u_k, x_{k+1}), u_{k+1} \sim h(x_{k+1})$

3. Turn into **incremental update**:
$$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k,u_k)]$$
$\alpha_k \in (0,1]$ learning rate

Intermediate algorithm

Temporal differences for policy evaluation

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

MC perspective

Temporal differences for policy h evaluation

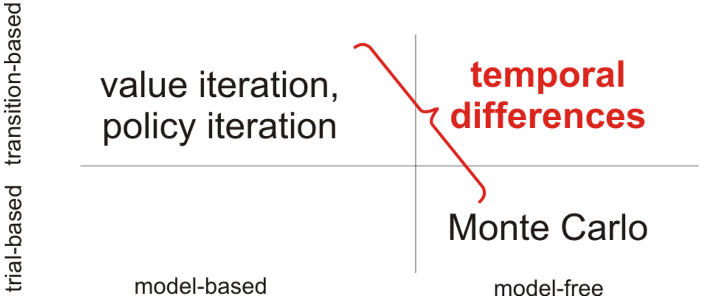
for each trial **do**
  ...
  repeat each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    $Q(x_k, u_k) \leftarrow ...Q...$
  until trial finished
end for

Monte Carlo

for each trial **do**
  execute trial
  ...
  $Q(x, u) \leftarrow A(x, u)/C(x, u)$
end for

MC and DP perspectives

- Learn from online interaction: like MC, unlike DP
- Update after each transition, using previous Q-values: like DP, unlike MC



Exploration-exploitation

choose next action $u_{k+1} \sim h(x_{k+1})$

- Information about $(x, u) \neq (x, h(x))$ needed
  $\Rightarrow$ **exploration**
- h must be followed
  $\Rightarrow$ **exploitation**

Policy improvement

- Previous algorithm: *h* fixed
- Improving *h*: simplest, after each transition, called optimistic policy improvement
- *h* implicit, greedy in *Q*
  (update $Q \Rightarrow$ implicitly improve *h*)
- E.g. ε-greedy:

$$u_{k+1} = \begin{cases} \arg\max_u Q(x_{k+1}, u) & \text{w.p. } (1 - \varepsilon_{k+1}) \\ \text{uniformly random} & \text{w.p. } \varepsilon_{k+1} \end{cases}$$

SARSA

SARSA

for each trial **do**
  init x_0
  choose u_0 with exploration based on Q
  repeat at each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose u_{k+1} with exploration based on Q

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

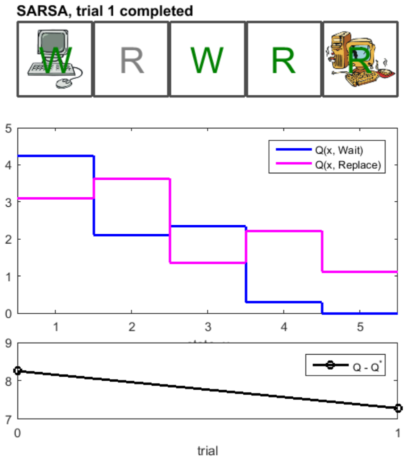
  until trial finished
end for

Origin of the name: $(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1})$ =
(**S**tate, **A**ction, **R**eward, **S**tate, **A**ction)

Machine replacement: SARSA demo

Parameters: $\alpha = 0.1$, $\varepsilon = 0.3$ (constant), single trial

$x_0 = 1$



Q-learning

1. Similarly to SARSA, start from Q-iteration:
   $$Q_{\ell+1}(x,u) \leftarrow \sum_{x'} f(x,u,x')[\rho(x,u,x') + \gamma \max_{u'} Q_\ell(x',u')]$$
2. Instead of model, use at each step k **transition sample** $(x_k, u_k, x_{k+1}, r_{k+1})$:
   $$Q(x_k,u_k) \leftarrow r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u')$$
   Note: $x_{k+1} \sim f(x_k,u_k,\cdot)$, $r_{k+1} = \rho(x_k,u_k,x_{k+1})$
3. Turn into **incremental** update:
   $$Q(x_k,u_k) \leftarrow Q(x_k,u_k) + \alpha_k \cdot [r_{k+1} + \gamma \max_{u'} Q(x_{k+1},u') - Q(x_k,u_k)]$$

Q-learning

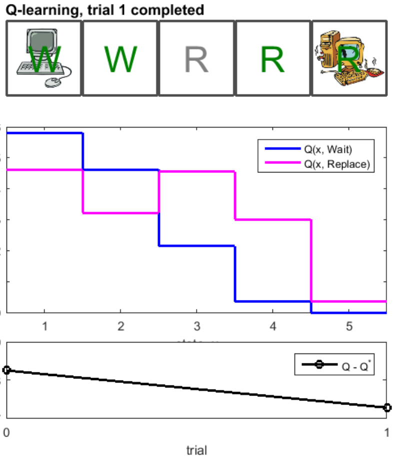
Q-learning

for each trial **do**
  init x_0
  repeat at each step k
    choose u_k with exploration based on Q
    apply 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 trial finished
end for

Machine replacement: Q-learning demo

Parameters: $\alpha = 0.1$, $\varepsilon = 0.3$ (constant), single trial

$x_0 = 1$



Convergence

Conditions for convergence to Q^*
in both SARSA and Q-learning:

1. All pairs (x, u) continue to be updated:
   requires **exploration**, e.g. ε-greedy
2. Technical conditions on α_k (goes to 0, $\sum_{k=0}^{\infty} \alpha_k^2 =$ finite,
   but not too fast, $\sum_{k=0}^{\infty} \alpha_k \to \infty$)

In addition, for SARSA:

3. Policy must become greedy asymptotically
   e.g. for ε-greedy, $\lim_{k \to \infty} \varepsilon_k = 0$

Discussion

SARSA **on-policy**

- Always updates towards the Q-function of the current policy

Q-learning **off-policy**

- Irrespective of the current policy, always updates towards optimal Q-function

Advantages of temporal differences

- Easy to understand and implement
- Low complexity $\Rightarrow$ fast execution

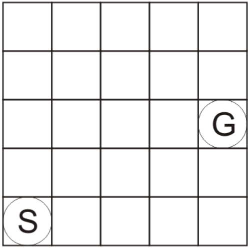
Exploration and α_k sequence **greatly influence** performance

1. Introduction
2. Markov decision process & optimal solution
3. Dynamic programming, DP
4. Monte Carlo, MC
5. Exploration basics
6. Temporal differences, TD
7. Improving data efficiency

Motivation

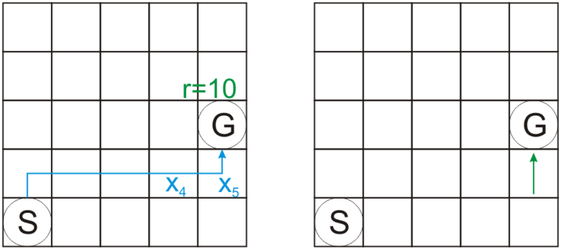
TD uses data inefficiently, and data is often costly.

Example:



- 2D gridworld navigation from **S**tart to **G**oal
- Nonzero reward = 10 only on reaching G (terminal state)

Motivation (cont'd)



- Take SARSA with $\alpha = 1$; initialize $Q = 0$
- Updates along the trial on the left:

$$\dots$$
$$Q(x_4, u_4) = 0 + \gamma \cdot Q(x_5, u_5) = 0$$
$$Q(x_5, u_5) = 10 + \gamma \cdot 0 = 10$$

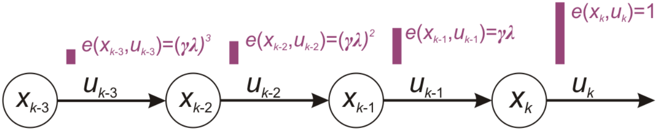
- A new transition from x_4 to x_5 (and hence a new trial) required to propagate the info to x_4!

Ideas presented

1. Eligibility traces
2. Experience replay

Eligibility traces

- Idea: Leave a **trace** along the trial:



- $\lambda \in [0, 1]$ decay rate

Replacing traces

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

Example algorithm: SARSA(λ)

- Recall original SARSA only updates $Q(x_k, u_k)$:
$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$$

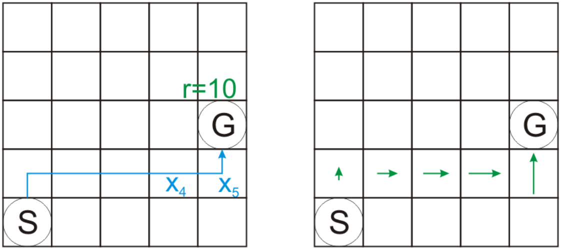
- SARSA(λ) updates **all eligible pairs**:
$$Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot [r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)] \quad \forall x, u$$

SARSA(λ)

SARSA(λ)

for each trial **do**
  init x_0
  $e(x, u) \leftarrow 0 \quad \forall x, u$
  choose u_0 with exploration based on Q
  repeat at each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    choose u_{k+1} with exploration based on Q
    $e(x, u) \leftarrow \lambda\gamma e(x, u) \quad \forall x, u$
    $e(x_k, u_k) \leftarrow 1$
    $Q(x, u) \leftarrow Q(x, u) + \alpha_k \cdot e(x, u) \cdot$
      $[r_{k+1} + \gamma Q(x_{k+1}, u_{k+1}) - Q(x_k, u_k)]$ for all x, u
  until trial finished
end for

Example: Effect of eligibility traces

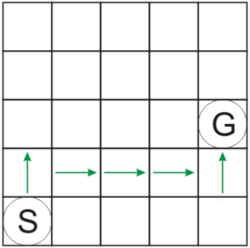


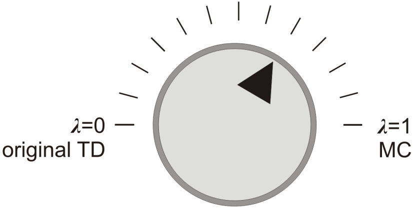
- $\lambda = 0.7$
- Updates until x_4: Q remains 0
- At x_5, the entire trial gets updated:

$$Q(x_5, u_5) = 10 + \gamma 0 = 10$$
$$Q(x_4, u_4) = (\gamma\lambda)[10 + \gamma 0] = 3.5$$
$$Q(x_3, u_3) = (\gamma\lambda)^2[10 + \gamma 0] = 1.225$$
$$\ldots$$

TD versus MC

- $\lambda = 0 \;\Rightarrow$ recovers original algorithms, e.g. SARSA(0)
- $\lambda = 1 \;\Rightarrow$ TD becomes like MC

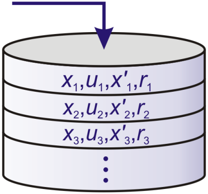




Typical values of λ are around 0.5 to 0.8

Experience replay

- Store each transition $(x_k, u_k, x_{k+1}, r_{k+1})$ in a database



- At each step, **replay** N transitions from the database (in addition to regular updates)

Q-learning with experience replay

Q-learning with experience replay

for each trial **do**
  init x_0
  repeat at each step k
    apply u_k, measure x_{k+1}, receive r_{k+1}
    $Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$
    $$[r_{k+1} + \gamma \max_{u'} Q(x_{k+1}, u') - Q(x_k, u_k)]$$
    add $(x_k, u_k, x_{k+1}, r_{k+1})$ to database
    ReplayExperience
  until trial finished
end for

ReplayExperience procedure

ReplayExperience

loop N times
  retrieve a transition (x, u, x', r) from database
  $$Q(x, u) \leftarrow Q(x, u) + \alpha \cdot [r + \gamma \max_{u'} Q(x', u') - Q(x, u)]$$
end loop

Retrieval order:

1. Backwards along trials, best for classical algos
2. Randomly, helps e.g. in deep RL
3. etc.

Textbooks

- Sutton & Barto, *Reinforcement Learning: An Introduction*, 1998 (+ ongoing 2nd edition, 2017).
- Bertsekas, *Dynamic Programming and Optimal Control*, vol. 2, 4th ed., 2012.
- Szepesvári, *Algorithms for Reinforcement Learning*, 2010.
- Buşoniu, Babuška, De Schutter, & Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 2010.