

Reinforcement learning

Master CPS, Year 2 Semester 1

Lucian Buşoniu, Florin Gogianu



Part III

Exact reinforcement learning



Part III in plan

- Reinforcement learning problem
- Optimal solution
- Exact dynamic programming
- **Exact reinforcement learning**
- Approximation techniques
- Approximate dynamic programming
- Approximate reinforcement learning



Algorithm landscape

By model usage:

- **Model-based**: f, ρ known a priori
- **Model-free**: f, ρ unknown (reinforcement learning)

By interaction level:

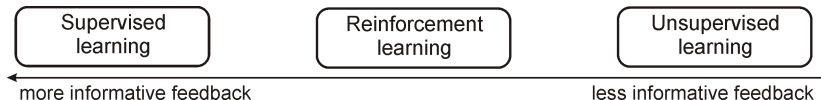
- **Offline**: algorithm runs in advance
- **Online**: algorithm runs with the system

Exact vs. approximate:

- **Exact**: x, u small number of discrete values
- **Approximate**: x, u continuous (or many discrete values)



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**!



Contents part III

- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
- 4 Accelerating TD methods
- 5 Recap



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
- 4 Accelerating TD methods
- 5 Recap



Reminder: Policy iteration

Policy iteration with Q-functions

initialize policy h_0 arbitrarily

repeat at each iteration ℓ

1: **policy evaluation**: find Q^{h_ℓ}

2: **policy improvement**:

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

until convergence to h^*

Note: In RL, we generally use **Q-functions** so policy improvement does not require a model



Policy evaluation

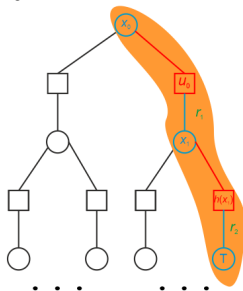
To find Q^h :

- So far: model-based methods
- Reinforcement learning: **model not available**
- Learn Q^h from offline data or via **online interaction with the system**



“Monte-Carlo” policy evaluation

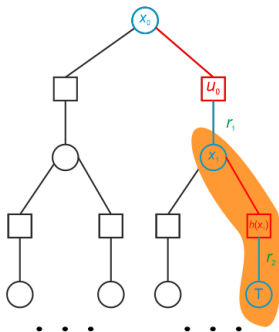
Recall: $Q^h(x_0, u_0) = \sum_{k=0}^{\infty} \gamma^k r_{k+1}$



- Trajectory from (x_0, u_0) to x_K (**terminal**) using $u_1 = h(x_1)$, $u_2 = h(x_2)$, etc.
- $\Rightarrow Q^h(x_0, u_0) = \text{return along trajectory:}$

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

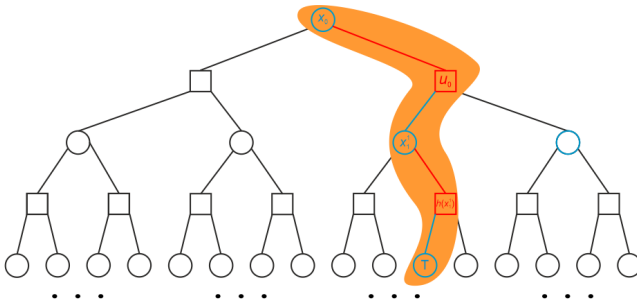
“Monte-Carlo” policy evaluation (cont’d)



- Moreover, 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: Stochastic case



- N trajectories (differing due to stochastic transitions)
- Estimated Q value = **mean** 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

```

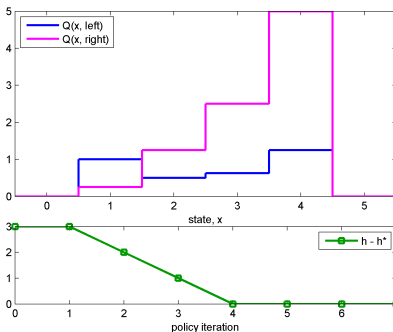
for each iteration  $\ell$  do
  perform  $N$  trajectories applying  $h_\ell$ 
  reset to 0 accumulator  $A(x, u)$ , counter  $C(x, u)$ 
  for each step  $k$  of each trajectory  $i$  do
     $A(x_k, u_k) \leftarrow A(x_k, u_k) + \sum_{j=k}^{K_i-1} \gamma^{j-k} r_{i,j+1}$  (return)
     $C(x_k, u_k) \leftarrow C(x_k, u_k) + 1$ 
  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
  
```

Note: must guarantee that **terminal state is reached!**



Cleaning robot: Monte Carlo, demo

Monte Carlo, trial 70 [piter 7 done, peval 10]



- 1 Monte Carlo, MC
- 2 **Exploration**
- 3 Temporal differences, TD
- 4 Accelerating TD methods
- 5 Recap



The need for exploration

In the MC estimate:

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

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

① **Initial states** x_0 representative

② **Actions:**

u_0 representative, sometimes different from $h(x_0)$
and additionally, possibly:

u_k representative, sometimes different from $h(x_k)$



Exploration-exploitation dilemma

- **Exploration** is necessary:
actions different from the current policy
- **Exploitation** of current knowledge is necessary:
current policy must be applied for good performance

The exploration-exploitation dilemma
– essential in all RL algorithms

(not only in MC)



ϵ -greedy strategy

- Simple solution to the exploration-exploitation dilemma:

ϵ -greedy

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

- Exploration probability** $\epsilon_k \in (0, 1)$
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 $\tau \rightarrow 0$, greedy selection recovered;
 $\tau \rightarrow \infty$ 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

Often-used algorithm: after n steps, pick arm with largest **upper confidence bound**:

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

where:

- $\hat{\mu}_j$ = mean of rewards observed for arm j so far
- n_j = how many times arm j was pulled
- c tunable constant, e.g. $3/2$

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



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
 - Introduction
 - SARSA
 - Q-learning
 - Discussion
- 4 Accelerating TD methods
- 5 Recap



Monte-Carlo with incremental updates

Consider the return sample from step k onwards:

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

Instead of averaging such samples to get Q , perform **incremental updates**:

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k [R_k - Q(x_k, u_k)]$$

where α_k is a step size, or **learning rate**



Discussion

- Incremental MC motivated by time-varying problems (recent samples have larger weights), but works in time-invariant MDPs as well
 - No longer need to store accumulators A and counters C , just directly the Q-values
 - If α satisfies “stochastic-approximation” conditions:
 - ① decreases to 0, $\sum_{k=0}^{\infty} \alpha_k^2 = \text{finite}$
 - ② but not too quickly, $\sum_{k=0}^{\infty} \alpha_k \rightarrow \infty$
- method converges: $\lim \text{when } \# \text{ of samples} \rightarrow \infty = \text{Q-value}$



Temporal differences (TD)

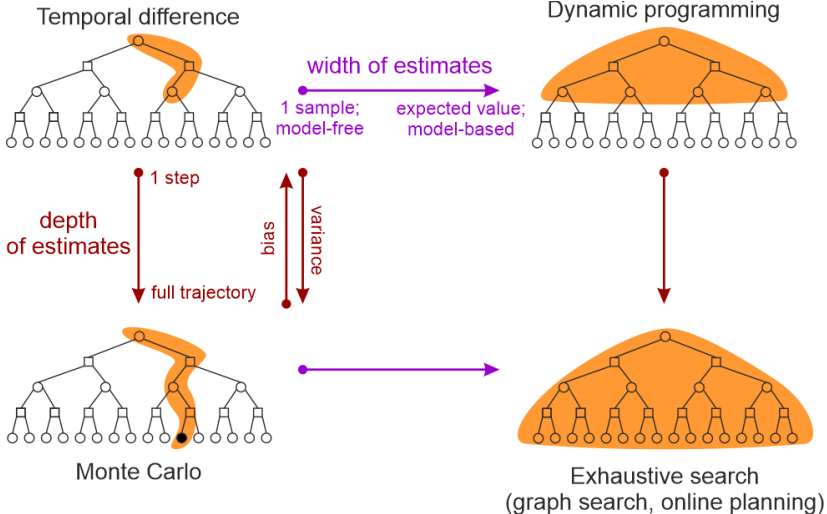
Otherwise, update remains the same, but let us make the return estimate explicit:

$$\begin{aligned} Q(x_k, u_k) &\leftarrow Q(x_k, u_k) + \alpha_k [\hat{R}_k - Q(x_k, u_k)] \\ &= Q(x_k, u_k) + \alpha_k [r_{k+1} + \gamma Q(x_{k+1}, h(x_{k+1})) - Q(x_k, u_k)] \end{aligned}$$

- [...] is the **temporal difference** between two estimates of $Q(x_k, u_k)$, using information at subsequent time steps
- Model-free, data-based updates (like MC): r_{k+1}, x_{k+1}
e.g. observed while interacting online
- Updates estimate $Q(x_k, u_k)$ using another estimate, $Q(x_{k+1}, h(x_{k+1}))$: **bootstrapping**
- Dynamic programming also bootstraps, but using a model



TD vs. MC vs. DP: Unified perspective



TD for policy evaluation

TD for policy evaluation

for each trajectory **do**

 initialize x_0 , choose the initial action u_0

repeat at each step k

 apply u_k , measure x_{k+1} , receive r_{k+1}

 choose the **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 trajectory finished

end for

Note: we replaced $h(x_{k+1})$ by u_{k+1} , chosen according to h



Exploration-exploitation

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

- Information about $(x, u) \neq (x, h(x))$ necessary
 $\Rightarrow$ **exploration**
- h must be followed
 $\Rightarrow$ **exploitation**
- E.g. ε -greedy:

$$u_{k+1} = \begin{cases} h(x_{k+1}) & \text{w.p. } (1 - \varepsilon_{k+1}) \\ \text{unif. random} & \text{w.p. } \varepsilon_{k+1} \end{cases}$$

- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD**
 - Introduction
 - SARSA**
 - Q-learning
 - Discussion
- 4 Accelerating TD methods
- 5 Recap



Recall: MC

MC policy iteration

```

for each iteration  $\ell$  do
  perform  $N$  trajectories applying  $h_\ell$ 
  reset to 0 accumulator  $A(x, u)$ , counter  $C(x, u)$ 
  for each step  $k$  of each trajectory  $i$  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
  
```

Optimistic policy improvement

- Policy unchanged for N trajectories
⇒ Algorithm learns slowly
- Policy improvement after each trajectory
= **optimistic**
- We will also use ϵ -greedy exploration



Optimistic MC

Optimistic MC

initialize to 0 accumulator $A(x, u)$, counter $C(x, u)$

for each trajectory **do**

execute trajectory, e.g., applying ε -greedy:

$$u_k = \begin{cases} \text{arg max}_u Q(x_k, u) & \text{w.p. } (1 - \varepsilon_k) \\ \text{unif. random} & \text{w.p. } \varepsilon_k \end{cases}$$

for each step k **do**

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

$$C(x_k, u_k) \leftarrow C(x_k, u_k) + 1$$

end for

$$Q(x, u) \leftarrow A(x, u) / C(x, u)$$

end for

- h implicit, greedy in Q
- update of $Q \Rightarrow$ implicit improvement of h



Optimism in TD

- Earlier TD algorithm: fixed h
- What is the fastest we can improve h in TD?
After each transition.
- ⇒ interpretation: policy iteration
optimistic at the transition level
- h implicit, greedy in Q
 (updating $Q \Rightarrow$ implicit improvement of h)



SARSA

SARSA with ε -greedy

for each trajectory **do**

initialize x_0

$$u_0 = \begin{cases} \arg \max_u Q(x_0, u) & \text{w.p. } (1 - \varepsilon_0) \\ \text{unif. random} & \text{w.p. } \varepsilon_0 \end{cases}$$

repeat at each step k

apply u_k , measure x_{k+1} , receive r_{k+1}

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

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$$

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

until trajectory finished

end for



Origin of name “SARSA”

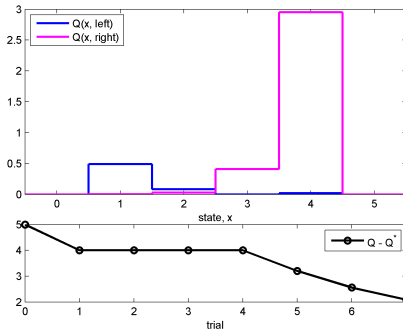
$(x_k, u_k, r_{k+1}, x_{k+1}, u_{k+1}) =$
 (State, Action, Reward, State, Action) = **SARSA**



Cleaning robot: SARSA, demo

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

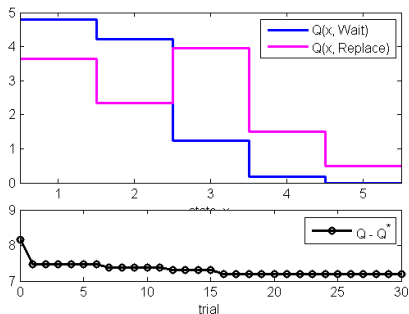
SARSA, trial 8, step 3



Machine replacement: SARSA, demo

Parameters: $\alpha = 0.1$, $\varepsilon = 0.3$ (constant), 20 steps per trajectory
 $x_0 = 1$

SARSA, trial 30 completed



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD**
 - Introduction
 - SARSA
 - **Q-learning**
 - Discussion
- 4 Accelerating TD methods
- 5 Recap



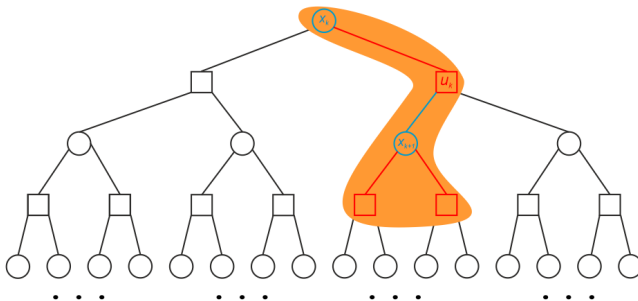
Bootstrapping estimate of Q^*

Bellman optimality equation:

$$Q^*(x, u) = E_{x'} \left\{ \tilde{p}(x, u, x') + \gamma \max_{u'} Q^*(x', u') \right\}$$

leads to the estimate:

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



TD update for Q^*

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k [\hat{Q}_k - Q(x_k, u_k)]$$

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



Q-learning

Q-learning with ϵ -greedy

for each trajectory **do**

initialize x_0

repeat at each step k

$$u_k = \begin{cases} \arg \max_u Q(x_k, u) & \text{w.p. } (1 - \epsilon_k) \\ \text{unif. random} & \text{w.p. } \epsilon_k \end{cases}$$

apply u_k , measure x_{k+1} , receive r_{k+1}

$$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$$

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

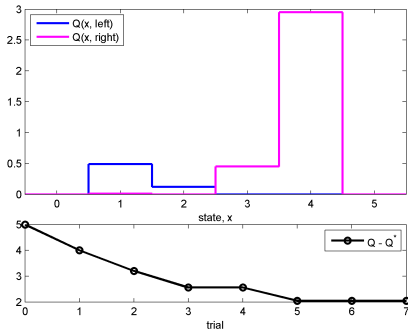
until trajectory finished

end for

Cleaning robot: Q-learning, demo

Parameters – same as SARSA: $\alpha = 0.2$, $\varepsilon = 0.3$ (constant)
 $x_0 = 2$ or 3 (random)

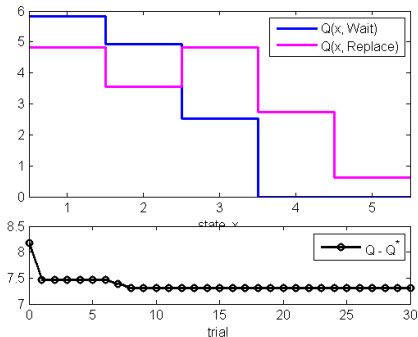
Q-learning, trial 8, step 3



Machine replacement: Q-learning, demo

Parameters: $\alpha = 0.1$, $\varepsilon = 0.3$ (constant), 20 steps per trajectory
 $x_0 = 1$

Q-learning, trial 30 completed



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD**
 - Introduction
 - SARSA
 - Q-learning
 - Discussion**
- 4 Accelerating TD methods
- 5 Recap



Convergence

Conditions for convergence to Q^* :

- 1 All pairs (x, u) continue to be updated:
ensured by **exploration**, e.g. ϵ -greedy
- 2 Stochastic-approximation conditions: learning rate decreases to zero, $\sum_{k=0}^{\infty} \alpha_k^2 = \text{finite}$, but not too quickly:
$$\sum_{k=0}^{\infty} \alpha_k \rightarrow \infty$$

Additionally, for SARSA:

- 3 The policy must become greedy at infinity
e.g. $\lim_{k \rightarrow \infty} \epsilon_k = 0$



On-policy / off-policy

SARSA: on-policy

- Always estimates the Q-function of the current policy

Q-learning: **off-policy**

- Independently of the current policy, always estimates the optimal Q-function



TD: Discussion

Advantages

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

SARSA vs. Q-learning

- SARSA less complex than Q-learning (no max in the Q-function update)

Learning rate and exploration sequences α_k, ε_k
significantly influence performance

Main disadvantage

- Large amount of data required



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
- 4 Accelerating TD methods**
 - Motivation
 - Experience replay
 - n-step returns
- 5 Recap



The need to accelerate TD

Main disadvantage: TD learns slowly – **requires a lot of data**

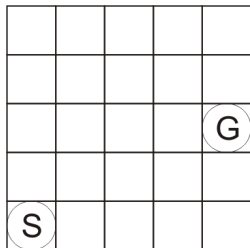
In practice, data costs:

- time
- profit (low performance due to exploration)
- system wear

Accelerating RL = **efficient use of data**

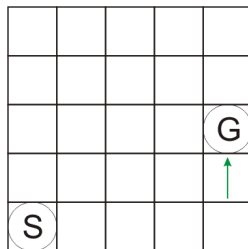
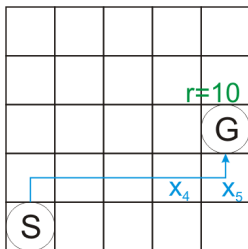


Example: 2D navigation



- Navigation in a discrete 2D world from **Start** to **Goal**
- Reward = 10 only upon reaching G (terminal state)

Example: TD



- We choose SARSA, $\alpha = 1$; initialize $Q = 0$
- Updates along trajectory on the left:

...

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

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

- A new transition from x_4 to x_5 needed to propagate the information to x_4 !

Accelerating TD: 2 ideas

- 1 Store and **replay experience**
- 2 Use **n-step returns**

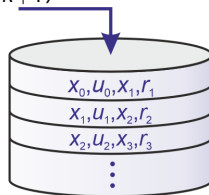


- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
- 4 Accelerating TD methods**
 - Motivation
 - **Experience replay**
 - n-step returns
- 5 Recap



Experience replay (ER)

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



- At each step, **replay** m transitions from database (in addition to normal updates)

Q-learning with ER

Q-learning with ER

for each trajectory **do**

 initialize x_0

repeat at each step k

 choose action u_k , e.g. ϵ -greedy in Q

 apply u_k , measure x_{k+1} , receive r_{k+1}

$Q(x_k, u_k) \leftarrow Q(x_k, u_k) + \alpha_k \cdot$

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

 add $(x_k, u_k, x_{k+1}, r_{k+1})$ to the database

 ReplayExperience

until trajectory ends

end for



ReplayExperience procedure

ReplayExperience

loop m times

 fetch a transition (x, u, x', r) from the database

$Q(x, u) \leftarrow Q(x, u) + \alpha \cdot$

$[r + \gamma \max_{u'} Q(x', u') - Q(x, u)]$

end loop

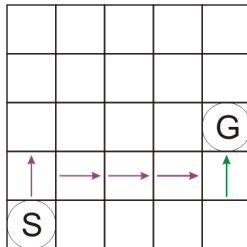
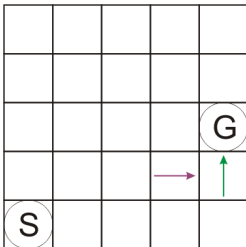
Replay direction

Order of replaying transitions:

- 1 Forward
- 2 Backward
- 3 Arbitrary

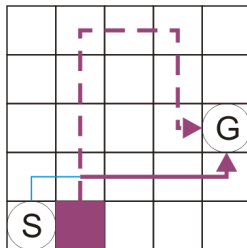
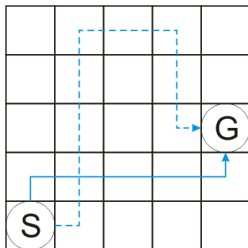


Example: Influence of replay direction



- Green: normal updates, purple: experience replay
- Left: forward replay; right: backward replay
- **Backward replay** preferable in exact RL

Example: Aggregating information

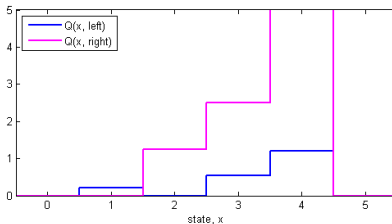


- Experience replay **aggregates information** from multiple trajectories
- The indicated cell benefits from information along both trajectories

Cleaning robot: Q-learning with ER, demo

Parameters: $\alpha = 0.2$, $\varepsilon = 0.3$, $m = 5$, backward direction
 $x_0 = 2$ or 3 (random)

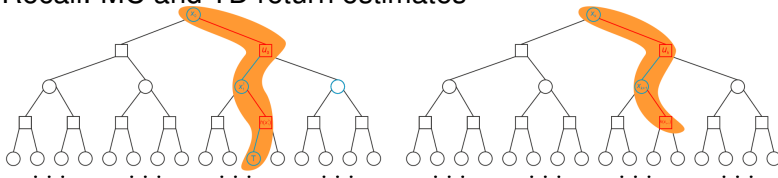
ER-Q-learning, trial 13, step 2 [replaying trial 8, step 2]



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
- 4 Accelerating TD methods**
 - Motivation
 - Experience replay
 - n-step returns**
- 5 Recap



Recall: MC and TD return estimates

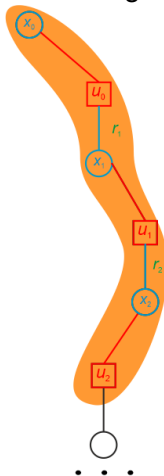


$$R_k = \sum_{j=k}^{K-1} \gamma^{j-k} r_{j+1}$$

$$\hat{R}_k = r_{k+1} + \gamma Q(x_{k+1}, h(x_{k+1}))$$

Is there something in-between?

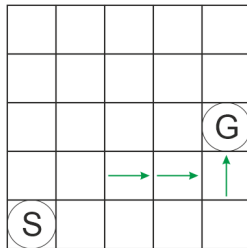
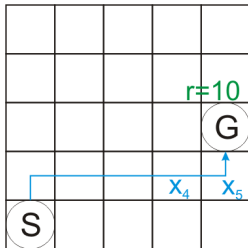
Middle ground: n-step return



SARSA (on-policy):

$$\hat{R}_k = r_{k+1} + \gamma r_{k+2} + \dots + \gamma^{n-1} r_{k+n} \\ + \gamma^n Q(x_{k+n}, u_{k+n})$$

Example: Effect of n-step return



For $n = 3$:

$$Q(x_5, u_5) = 10 + 0 \quad (\text{terminal})$$

$$Q(x_4, u_4) = 0 + \gamma 10 + 0 \quad (\text{terminal})$$

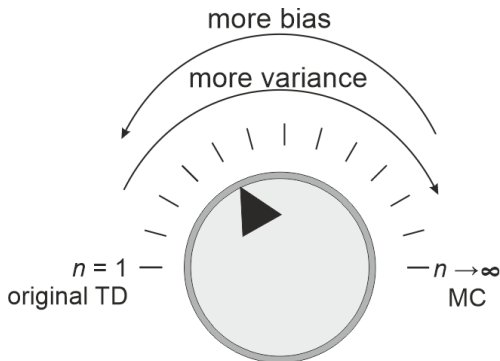
$$Q(x_3, u_3) = 0 + \gamma 0 + \gamma^2 10 + 0 \quad (\text{terminal})$$

$$Q(x_2, u_2) = 0 + \gamma 0 + \gamma^2 0 + \gamma^3 0 \quad (\text{bootstrap})$$

...

TD versus MC

- $n = 1$ recovers TD, $n \rightarrow \infty$ recovers MC
- Intermediate values mix TD and MC, leading to a tunable bias-variance tradeoff



- 1 Monte Carlo, MC
- 2 Exploration
- 3 Temporal differences, TD
- 4 Accelerating TD methods
- 5 Recap



Recap: Methods in Part III

Monte-Carlo methods, MC:

- MC policy iteration
- MC with incremental updates

Exploration-exploitation dilemma:

- ϵ -greedy widely used
- Many other solutions exist, like UCB

Temporal differences, TD:

- TD for policy evaluation
- Optimistic policy improvements
- SARSA
- Q-learning

Accelerating TD:

- Experience replay
- n-step returns



Key terms in this part

- Monte-Carlo methods
- exploration-exploitation dilemma
- ϵ -greedy, softmax, bandits
- optimistic policy improvement
- learning rate
- bootstrapping
- temporal differences
- SARSA
- Q-learning
- experience replay
- n-step returns



Exercises

- 1 Does the exponential schedule $\alpha_k = \alpha^k$, with $\alpha \in (0, 1)$ a constant, satisfy the stochastic approximation conditions?
- 2 Is Q-learning guaranteed to converge when $\varepsilon_k = \varepsilon$, a constant in $(0, 1)$? What about SARSA? How about when you use an exponential decrease $\varepsilon_k = \varepsilon^k$?
- 3 Would a Monte-Carlo algorithm that improves the policy after every transition (like TD) make sense?
- 4 Would Q-learning (without n-step returns as they are nontrivial in the off-policy case) propagate information faster than SARSA for the gridworld trajectory example?



Exercises (cont'd)

- 5 Assuming that we have access to a model **only for the purposes of policy improvement**, provide V-function alternates for all algorithms in this part. Do this in the same order as for Q-functions:
- Monte Carlo estimates, averaging-based and incremental
 - Bootstrapping estimates and updates
 - Policy evaluation, SARSA, and Q-learning

Don't forget to draw trees and highlights, it will help you visualize things.

