RL 3: Reinforcement Learning

$\mathcal{Q}$-Learning

Michael Herrmann

University of Edinburgh, School of Informatics

19/01/2016

Last time: Multi-Armed Bandits (points to remember)

- MAB applications do exist (e.g. experimental design)
- Exploration-exploitation dilemma: Both the reward and information about the reward are important. Although dilemma in general not unambiguously solvable, for MAB reasonable solutions exist.
- Choose action based on (estimated) value ($\hat{Q}$)
    - deterministically, e.g. $a_{t+1} = \arg\max_a \hat{Q}_t(a)$ (greedy)
    - stochastically, e.g. $p(a_{t+1}) = \frac{e^{\hat{Q}_t(a_{t+1})/\tau}}{\sum_{b=1}^{N} e^{\hat{Q}_t(b)/\tau}}$ (Boltzmann)
    - mixed, e.g. ε-greedy, i.e. random in $\varepsilon \cdot 100\%$ of trials
- Value = Expectation of the action-specific reward distributions
- $\hat{Q}_k = \hat{Q}_{k-1} + \frac{1}{k}\left(r_k - \hat{Q}_{k-1}\right)$ yields the exact mean of r_k for $\hat{Q}_0 = 0$
- $\hat{Q}_{k+1} = (1-\alpha)\,\hat{Q}_k + \alpha r_{k+1}$ gives an exponentially weighted average (for non-stationary problems; initialisation is less critical)
- Value can include other information, e.g. confidence
- Regret: $\rho = \frac{1}{T}\left(T\mu^* - \sum_{t=1}^{T} \bar{r}_t\right)$ where $\mu^* = \max_k \mu_k$
- Optimistic initialisation of $\hat{Q}$ often provides a good exploration

For more details see e.g. ICML 2011 Tutorial Introduction to Bandits: Algorithms and Theory, Jean-Yves Audibert, Remi Munos https://sites.google.com/site/banditstutorial/

Three Aspects of Learning (mainly in robots)

- Selection of appropriate actions
  - reinforcement learning
- Adaptation of perceptual mechanisms
  - standard machine learning
  - obtaining more or more relevant information about the system
  - usually assumes a fixed data distribution
- Shaping of the learning problem
  - data distribution depends on the actions of the agent
  - active learning
  - an effect caused by reinforcement learning (which is sometimes ignored in the analysis of the algorithm)

Overview

- MAB
- RL algorithms and applications
- RL theory (MDPs)
- POMDPs
- RL in continuous space and time
- Variants of RL
  - hierarchical RL
  - collective RL
  - multi-objective RL
  - intrinsically motivated RL
  - inverse RL
  - ...
- RL in biology and psychology

Towards associative RL and temporal credit assignment

Previously,

- we were in a single casino and the only decision was to pull one from a set of N arms
- not more than a single state!

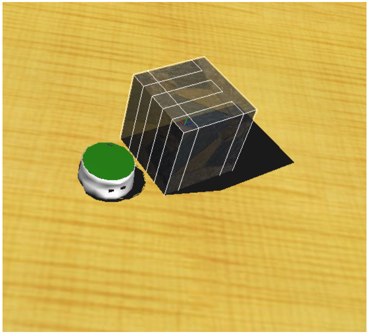
Now,

- what if there is more than one state and each state has a different reward distribution?
- what if actions do not only earn you a reward but also move you to another state?
- what happens if you only obtain a net reward corresponding to a long sequence of arm pulls (at the end)?

We'll need a policy (a map of states to actions) ...

A simple example

- A robot learns to "stop-and-back-up-and-turn" (a_1) if near an obstacle (s_1) and to "go" (a_2) if away from an obstacle (s_2) given the reward signals $r(s_2) = 1$ and $r(s_1) = 0$
- How does the robot find out that a_2 is actually better in s_1 although s_1 always implies $r = 0$?
- How can the robot be kept from preferring a_1 in state s_2?
- What can go wrong?



Brute force

1. For each possible policy, sample returns while following it
2. Choose the policy with the largest expected return

If a trial takes T time steps and a

Policy π assigns to each time $t \leq T$ an action $a \in \mathcal{A}$ by

$$\pi : \mathcal{T} \mapsto \mathcal{A},$$

where $\mathcal{T}$ is the set of decision times, i.e. $|\mathcal{T}| = T$.

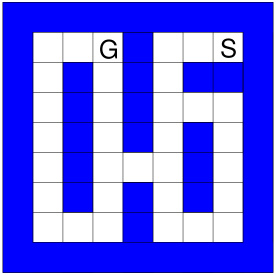
In principle, there are thus $|\mathcal{A}|^T$ policies.

(assuming the cardinalities $|\mathcal{A}|$ and T are finite)

For the bandit problem with $T = 1$ this worked well

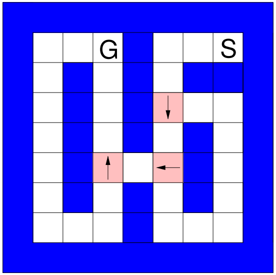
Improving beyond brute force

Consider a maze of 7×7 fields, i.e. $|\mathcal{T}| \leq 49$. There are four directions: up, down, left and right (or N, E, S, W) So there are at most 4^{49} policies.
Many can be excluded when using the immediate punishment given at bumping into a wall.
Employing in addition an inhibition-of-return principle usually even less actions are admissible.
Many policies differ only in the part after the goal or are equivalent (e.g. "first up then left" may equal "first left then up" if there were no obstacles).



Improving beyond brute force

- Branch and bound
- Define policy based on states
  $s \in \mathcal{S}$: $\quad \pi : \mathcal{S} \mapsto \mathcal{A}$
  - Some states are inaccessible (try not to represent what is never experienced)
  - Avoid reaching at state repeatedly in one run (e.g. by intrinsic reward/punishment)
  - But: states need to incorporate all the relevant information*



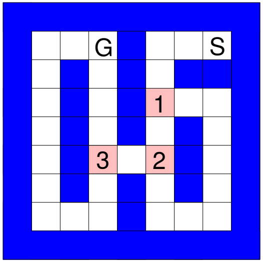
An efficient solution. Trivial action choices are not shown.

- Keep in mind for later:
  - Allow samples generated from one policy to influence the estimates made for another.
  - Introduce a *geometry* on the set of states.

*E.g. in the pendulum swing-up task states typically include position *and* velocity.

Preliminary considerations

- If information about the task is available, consider using it
- Configurations of different actions may be equivalent $a_1 = E$ and $a_3 = N$ is as good as $a_1 = S$ and $a_3 = S$ (and $a_2 = W$)
- or even misleading: $a_1 = E$ and $a_2 = W$ is better than $a_1 = S$ and $a_2 = S$, although the globally best policy has $a_1 = S$ (and $a_2 = W$, $a_3 = N$)
- Earlier decisions may influence later decisions (consider agent-centred code: L, R, S, B)
- Be that as it may, do we have a chance to deal with problems of this type if the are stochastic?
- There are a number of approaches, we choose one (and consider others later)



State-action values: Towards $\mathcal{Q}$-Learning

Assume that the cardinalities $|\mathcal{A}|$ and $|\mathcal{S}|$ are finite, and $\mathcal{T} \subseteq \mathbb{N}_0$.

We aim at expressing values related to state-action pairs:

$$\mathcal{Q}(s, a)$$

Intuitively, the $\mathcal{Q}$ function expresses the *expected*[*] reward when being in state s and applying a.

By definition, states contain the relevant information about the system, i.e. the value of an action can depend only on the state (i.e. not on time nor any hidden variables)

[*]We will see later that 'expected' does not imply here the estimation of the mean reward when applying a many times in state s. It is rather a measure of how much reward the agent will obtain when moving from state to state (we will still have to average in a stochastic problem).

Value (beyond immediate reward)

In many cases, it may be worth to pass on a small immediate reward, but in this way securing a bigger reward later ($\rightarrow$ *delayed gratification*).

Similarly, often there is no specific reward, e.g. in some states all actions will lead to a 'reward' $r = 0$

Although actions are thus not distinguishable w.r.t. reward, they may lead the agent to other states.

We could still try to choose an action that leads us to a better state.

$\Rightarrow$ There are now two types of reward: the *immediate reward r* (which may be zero) and the *value* of the subsequent state

But what is the value of the subsequent state and how do we obtain it?

State value and state-action value

The value $V(s_1)$ of a state s_1 is defined as the value of the best action that can be applied in this state

$$V(s_1) = \max_{a \in \mathcal{A}} \left(\mathcal{Q}(s_1, a) \right)$$

- Do we have to solve a MAB problem for each state?
- These MAB problems would not be independent!
- How do we trade value across the MABs?

Towards $\mathcal{Q}$-Learning

Taken together, if s_1 is reached from s due to a, then (preliminary)

$$\begin{aligned}\mathcal{Q}(s,a) &= r(s,a) + V(s_1)\\ &= r(s,a) + \max_{a\in\mathcal{A}}\left(\mathcal{Q}(s_1,a)\right)\end{aligned}$$

Immediate reward

plus

Value of the next state

(if there is any)

(assuming best possible action)

s_1 reached from s by action a

If we knew and apply the best action for each of the subsequent states, then $\mathcal{Q}$ is just the sum of all future rewards.

Iterative learning rule

$$\rho_t = r\left(s_{t-1}, a_{t-1}\right) + V\left(s_t\right)$$

Combined reward = immediate reward + value of new state

$$\begin{aligned}
\mathcal{Q}_t\left(s_{t-1}, a_{t-1}\right) &= \mathcal{Q}_{t-1}\left(s_{t-1}, a_{t-1}\right) + \eta\left(\rho_t - \mathcal{Q}_{t-1}\left(s_{t-1}, a_{t-1}\right)\right) \\
&= \left(1-\eta\right)\mathcal{Q}_{t-1}\left(s_{t-1}, a_{t-1}\right) + \eta\rho_t \\
\Delta\mathcal{Q}_t\left(s_{t-1}, a_{t-1}\right) &= \eta\left(\rho_t - \mathcal{Q}_{t-1}\left(s_{t-1}, a_{t-1}\right)\right)
\end{aligned}$$

When observing a state transition from s_{t-1} to s_t under action a_{t-1}, then use a sliding average to estimate the combined reward.

Notes:

- Not all actions $a \in \mathcal{A}$ need to be admissible in all states $s \in \mathcal{S}$.
- For $\mathcal{Q}_t$ the parameter t refers to update steps, for s_t and a_t is refers to the physical time of the system. Often it is reasonable to keep the two parameters in sync.
- Temporally a_{t-1} is in between s_{t-1} and s_t, we use the index $t-1$ by convention

The $\mathcal{Q}$-learning algorithm (greedy non-discounted version)

1. Choose $\mathcal{A}$, $\mathcal{S}$ and parameter η
2. Initialise $\mathcal{Q}_0(s,a)$ for all $a \in \mathcal{A}$ and all $s \in \mathcal{S}$, e.g. using small random values
3. Obtain information on state s_0
4. For all $t \geq 1$
    1. Receive reward r_{t-1} (if reward depends only on state s_{t-1})
    2. Select and perform action $a_{t-1} = \arg\max_{a \in \mathcal{A}} (\mathcal{Q}(s_{t-1}, a))$
    3. Wait until time t, then obtain information on state s_t
    4. Determine value of the new state
    $$V(s_t) = \max_{a \in \mathcal{A}} (\mathcal{Q}(s_t, a))$$
    5. Update the $\mathcal{Q}$-function

$$\mathcal{Q}_t(s_{t-1}, a_{t-1}) = \mathcal{Q}_{t-1}(s_{t-1}, a_{t-1}) + \eta(r_{t-1} + V(s_t) - \mathcal{Q}_{t-1}(s_{t-1}, a_{t-1}))$$

    6. Test whether goal was reached $\rightarrow$ restart agent and GOTO 3
    7. Test whether the agent is good enough $\rightarrow$ return(success)

Remarks on the $\mathcal{Q}$-Learning algorithm

1. Action selected according to $a_{t-1} = \arg\max_{a \in \mathcal{A}} \left(\mathcal{Q}\left(s_{t-1}, a\right)\right)$ which is greedy. Other choices for the action selection are possible, e.g. ε-greedy with decaying ε. This does not (directly) affect the other parts of the algorithm.
2. The parameter η is a learning rate. It performs an exponential average restricted to the times when a particular state-action pair occurred.
3. Be careful when restarting the agent. The value for a goal state should not be influenced by the following starting state.
4. Often it does not matter whether the $\mathcal{Q}$-function has actually converged. The precise values may not matter, as long as the max-operations yield the correct result.

A silly example (deterministic case)

Imagine a person moving randomly through Edinburgh in search for the Informatics Forum. She eventually finds it and considers this a rewarding experience. The person doesn't remember her route through the City, except that before she got to the Forum she was in a place called George Square.

At another trip she happens to arrive at George Square again. She knows now how to get to the Forum and also remembers that right before she got to George Square she was on a large Meadow.

At another trip she arrives at the Forum coming from the Old Medical School.

At another trip she happens to arrive at the Old Medical School after having been at Forrest Road.

Much later ... ... all places at Edinburgh are nodes in a tree with the Forum as the root. The agent still has no idea about the city layout, but knows from each place how to continue in order to arrive eventually at the Forum.

Value functions for delayed reward

The value of a state is the (discounted) total future reward that is expected when choosing the presently known best action for this state continue with the policy that is currently considered optimal.

$$\begin{aligned} V(s_t) &= r(s_t, a^*(s_t)) + r(s_{t+1}, a^*(s_{t+1})) + r(s_{t+2}, a^*(s_{t+2})) + \cdots \\ &= \sum_{k=t}^{\infty} r(s_k, a^*(s_k)) \end{aligned}$$

Discounted version $\gamma \leq 1$:

$$\begin{aligned} V(s_t) &= r(s_t, a^*(s_t)) + \gamma r(s_{t+1}, a^*(s_{t+1})) + \gamma^2 r(s_{t+2}, a^*(s_{t+2})) + \cdot\cdot \\ &= \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a^*(s_k)) \end{aligned}$$

Discount parameter γ introduces a time horizon the length $\frac{1}{1-\gamma}$ of which depends on the problem.

Discounted update

$$
\begin{aligned}
V(s_t) &= \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a^*(s_k)) \\
&= r(s_t, a^*(s_t)) + \sum_{k=t+1}^{\infty} \gamma^{k-t} r(s_k, a^*(s_k)) \\
&= r(s_t, a^*(s_t)) + \gamma \sum_{k=t+1}^{\infty} \gamma^{k-(t+1)} r(s_k, a^*(s_k)) \\
&= r(s_t, a^*(s_t)) + \gamma V(s_{t+1})
\end{aligned}
$$

The $\mathcal{Q}$-function is updated analogously, but for an arbitrary action

$$\mathcal{Q}_{t+1}(s_t, a_t) = \mathcal{Q}_t(s_t, a_t) + \eta \left(r(s_t, a_t) + \gamma V(s_{t+1}) - \mathcal{Q}_t(s_t, a_t) \right)$$

Now, immediate reward is preferable to delayed reward.

$\mathcal{Q}$-learning was first introduced by C. J. C. H. Watkins (1989)

Toy Example: 1D maze
Problem set up

- Track $\mathcal{S}$: $|\mathcal{S}| = N$, spaces numbered $1, \ldots, N$,
- States $s_t \in \mathcal{S}$, state at time $t \geq 0$
- Actions: $a_t \in \mathcal{A} = \{\text{left}, \text{right}\} \equiv \{-1, +1\}$
- Reward: $r(s, a) = r(s) = \begin{cases} 1 & \text{if } s = 1 \text{ (goal)} \\ 0 & \text{otherwise} \end{cases}$
- Initialisation $\mathcal{Q}(s, a) = 0 \ \forall s \in \mathcal{S}, a \in \mathcal{A}$ (not really optimistic)
- Exploration: ε-greedy with (initially) $\varepsilon = 1$, i.e. random moves later $\varepsilon \to 0$ and $a_t = \arg\max_a Q(s_t, a)$ for non-random moves
- Starting at random state $s_0 = k \in \mathcal{S}$,
- Restart after reaching goal or after timeout
- Return after convergence of $\mathcal{Q}$-function or at sufficient performance

Toy Example: 1D maze, solution for $\eta = 1$

- Random moves until $s_t = 1$, then $r_t = 1$. Now, make any move a and update $\mathcal{Q}(1, a) = 1$, restart
- If $s_{t+1} > 1$, then $r_{t+1} = 0$. Perform a random move a leading to s_{t+2}
  - if $s_{t+2} = 1$ then update $\mathcal{Q}(2, a) = 0 + \gamma V(1)$, restart [for this to happen, a had to be “left”]
  - if $s_{t+2} > 1$ then $\mathcal{Q}(s_{t+1}, a)$ remains 0

After ε decayed, we arrive at the value function

for $N > k > 1$: $\mathcal{Q}(k, \text{left}) = \gamma^{k-1}$, $\mathcal{Q}(k, \text{right}) = \gamma^{k-3}$

For the ends of the track: $\mathcal{Q}(1, \text{right}) = 1$, $\mathcal{Q}(N, \text{left}) = \gamma^{N-1}$.

Non-admissible moves are never updated: $\mathcal{Q}(1, \text{left}) = 0$, $\mathcal{Q}(N, \text{right}) = 0$

Homework: Write a simulation for this example

Example 2: MAB as a Special Case

$\mathcal{Q}$-learning for the N-armed bandit

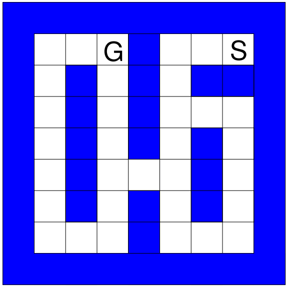
- States: $s \in \{C\}$ (just one state, namely the *C*asino)
- Actions: $a \in \{1, \ldots, N\}$
- state transitions $C \to C, \forall a$
- Reward: $r = r_a$ with $r_a \sim \mathcal{N}\left(\mu_a, \sigma_a^2\right)$
- Initialisation: $\mathcal{Q}_0(a, s) = 5 \times \max_a \{\mu_a + \sigma_a\}$ (optimistic)

$$\begin{aligned}
\mathcal{Q}_{t+1}(s_t, a_t) &= \mathcal{Q}_t(s_t, a_t) + \eta\left(r(s_t, a_t) + \gamma V_t(s_{t+1}) - \mathcal{Q}_t(s_t, a_t)\right) \\
&= \mathcal{Q}_t(a_t) + \eta\left(r(a_t) + \gamma V_t - \mathcal{Q}_t(a_t)\right) \\
\tilde{\mathcal{Q}}_{t+1}(a_t) &= \tilde{\mathcal{Q}}_t(a_t) + \eta\left(r(a_t) - \tilde{\mathcal{Q}}_t(a_t)\right)
\end{aligned}$$

- $a_t = \arg\max_a \mathcal{Q}_{t+1}(a) = \arg\max_a \tilde{\mathcal{Q}}_{t+1}(a)$

Example 3: Another simple maze

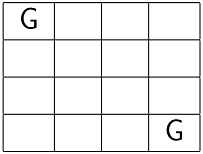
- States: $s \in \{7 \times 7 \text{ squares}\}$ with obstacles (see previous lecture)
- Actions: $a \in \{E, W, N, S\}$
- Reward: $r = 1$ if $s \equiv G$, else $\{r = 0$ for any admissible move and else $r = -100\}$
- Initialisation: $\mathcal{Q}_0(a, s) \sim \mathcal{N}(0, 0.01)$
- Soft-max exploration (Boltzmann)



Result: Similar to the 1-D example.

Example 4: Navigation in a grid world

- States: $s \in \{4 \times 4 \text{ squares}\}$ without obstacles
- Actions: $a \in \{E, W, N, S\}$
- Reward: $r = 0$ if $s \equiv G$ (two of the corners),
  $r = -1$ for each step taken
- discount factor: $\gamma = 1$ (no discount)



- Initialisation: $\mathcal{Q}(a, s) \sim \mathcal{N}(0, 0.01)$
- Reset to random position after reaching the goal

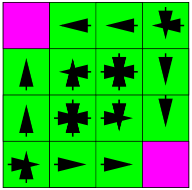
Example 4: A fully explorable simple maze

0	−14	−20	−22
−14	−18	−22	−20
−20	−22	−18	−14
−22	−20	−14	0

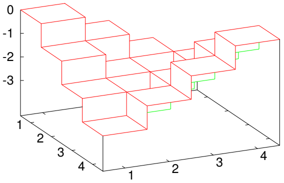
average path length
for random exploration

0	−1	−2	−3
−1	−2	−3	−2
−2	−3	−2	−1
−3	−2	−1	0

minimal path length
using optimal policy



preferred actions
(may stay undecided)



non-discounted
value function
over state
space

Adapted from
Mance E. Harmon:
Reinforcement Learning:
A Tutorial (1996)

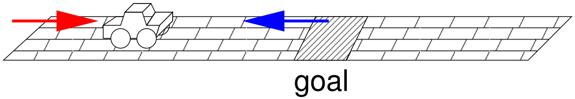
Navigation in a grid world: Discussion of Example 4

- Random exploration (i.e. ε-greedy with $\varepsilon = 1$) provides (in this example; in general it's better to slowly decrease ε) the information about optimal policy.
- If $Q(s, a)$ is randomly initialised (and ε is small) then the agent may easily get stuck.
- The values at the goals are due to a special treatment of these states. The reward is immediate reward plus value of next state:
    - if the agent is reset to a random position for the next episode then the next state may have a very low value
    - if the agent stays at G then another step is taken which has a cost of -1, but is this not counted.
    - Practically, we are not using $Q(G, a)$, only $V(G)$ which is defined to be zero an used to update $Q(s, a)$ of the previous state that led the agent to the goal.
    - If we did update $Q(G, a)$ we should (make sure that we) obtain $Q(G, a) = 0$ for a towards the wall, and $Q(G, a) = -2$ for the a's back to the maze.

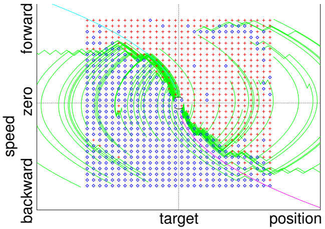
Example 5: Car Positioning
(A simplified version of the Mountain Car problem)

- States include position and speed: $s \in [-P, P] \times [-V, V]$ discretrised into 40×40 squares
- Actions: $a \in \{\text{accelerate forward, accelerate backward}\}$
- Reward: $r = 1$ if stopping in a small region near the goal, i.e. both speed and position close to zero, $r = 0$ otherwise
- Initialisation: $\mathcal{Q}_0(a, s) \sim \mathcal{N}(0, 0.01)$
- Discount factor: $\gamma = 0.95$
- Exploration: initially $\varepsilon = 0.25$, decaying over 100,000,000 time steps
- Learning rate $\eta = 0.1$

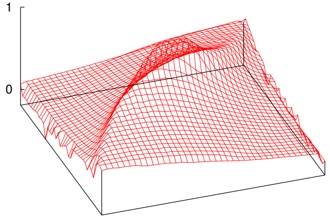
Example 5: Cart positioning by Q-learning



Accelerate cart such that it stops at a given position in min. time
Only one level of acceleration, action is to choose the sign
Size of goal region determines minimal state space resolution
Problems: relevant part of state space, slow convergence



preferred actions



value function

Example 5: Discussion

- Extremely long learning time
- How many time step does is take until the agent reaches a new state?
- Neighbouring states are doing largely the same
- The boundary between the region is not very well resolved even for a fine democratisation
- Try (using information available during the learning process)
  - success stories: update not only previous time steps, but also everything that lead to the final success
  - adaptive partitioning of the state space: In more homogeneous regions use few states, whereas near critical boundaries more state are needed (e.g. based on a weighted k-means algorithm)
  - use options: only update once new information becomes available

Examples and Applications

- Higher dimensional mazes, grid worlds, search trees etc.
- Games, decision making, modelling the control of behaviour
- Cart-pole, pendulum swing-up, multiple pendula, mountain-car
- Chaos control, robot control
- Industrial control, production control, automotive control, autonomous vehicles control, logistics, telecommunication networks, sensor networks, ambient intelligence, robotics, finance (s. Real World Applications of Reinforcement Learning at *International Joint Conference on Neural Networks* 2012)

Discussion and outlook

- $\mathcal{Q}$-learning is an *off-policy* algorithm: Learning the value of state-action pairs independently of their position in a policy, i.e. learning generalises across policies
  - Policies are not factorisable in general
  - If the algorithm contains exploration it is not performing optimally and therefore is does not know the value of the currently optimal policy
  - We will later compare on- and off-policy algorithms
- Look-up table representation of the $\mathcal{Q}$-function ($\mathcal{Q} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$) is not very efficient. We will later use function approximation
- Other exploration schemes are clearly possible
- Convergence (Watkins and Dayan, 1992): We need a solid theoretical basis for this
- Complexity: [will take time]

Reading

Ch 6.4 of Sutton & Barto book deals with $\mathcal{Q}$-learning only briefly. Please check wikipedia and some of the papers on $\mathcal{Q}$-learning, such as:

- Watkins, C. J., Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292.
- Ribeiro, C. H. C. (1999). A tutorial on reinforcement learning techniques. In: Supervised Learning track tutorials of the 1999 *International Joint Conference on Neuronal Networks*.