Reinforcement learning

Philippe Preux
philippe.preux@univ-lille.fr
Scool

This presentation:

`https://philippe-preux.github.io/talks/AISS-Insa-Rouen/AISS.pdf`

There is a companion for practical work (on your own) at

`https://philippe-preux.github.io/talks/AISS-Insa-Rouen/AISS-companion.pdf`

Reinforcement learning in 2023

$1/2/++$ slides de propagande pour mettre en appétit.

The roots of reinforcement learning

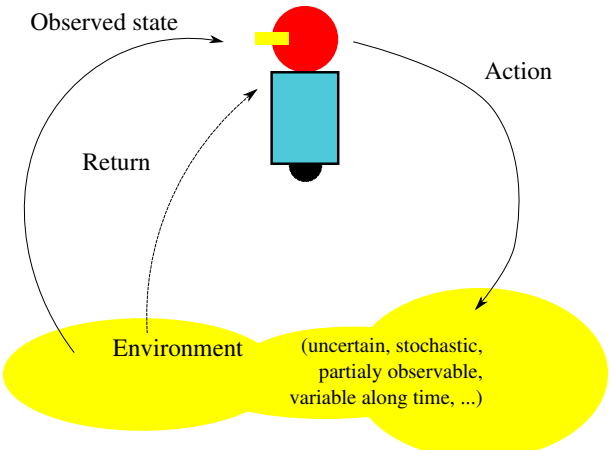
- Law of effect: Thorndike, 1896.
- Classical conditioning: Pavlov, 19..
- Operant conditioning: Skinner *et al.* from 19.. on.
- Rescorla-Wagner law: 1972.
-
- Sutton's Ph.D. defended in Feb. 1984.

frise avec dates importantes

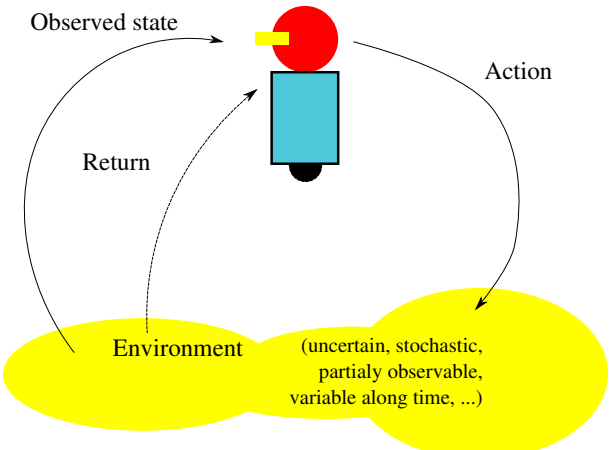
Outline

- Introduction
- Markov decision processes and Markov decision problems
- Reinforcement learning: definition and algorithms
- RL in practice
- How could you contribute to RL?

Markov decision problems



Markov decision problems



Learn an optimal behavior.

Markov decision process

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
  For the sake of simplicity, $\mathcal{T}$ is usually the sequence of positive integers: $0, 1, 2, ...$

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
  For the sake of simplicity, the states are usually numbered from 0 to $N-1$, with $N = |\mathcal{X}|$.

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$.
  For the sake of simplicity, actions are usually numbered from 0 to $P-1$, with $P = |\mathcal{A}|$.

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$.
- $\mathcal{P}$ is the transition function.
  Assume at time t, the environment is in state $x_t = x$ and performs action $a_t = a$, then $\mathcal{P}(x, a, x')$ is the probability that the environment will be in state x' at time of decision $t + 1$.
  That is: $\mathcal{P}(x, a, x') = Pr[x_{t+1} = x' | x_t = x, a_t = a]$.

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$.
- $\mathcal{P}$ is the transition function.
- $\mathcal{R}$ is the return function, $r \in \mathbb{R}$.
  With the same assumption as for $\mathcal{P}$: $\mathcal{R}(x, a, x')$ is the expected return for the transition from state x to x' following action a.
  That is: $\mathcal{R}(x, a, x') = \mathbb{E}[r_t | x_{t+1} = x', x_t = x, a_t = a]$.

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$.
- $\mathcal{P}$ is the transition function.
- $\mathcal{R}$ is the return function, $r \in \mathbb{R}$.

Remarks:

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$.
- $\mathcal{P}$ is the transition function.
- $\mathcal{R}$ is the return function, $r \in \mathbb{R}$.

Remarks:

1. items depend on t and do not depend on $t-1$, $t-2$, ...: Markov property.

Markov decision process

describes a dynamical decision system

Definition

A Markov decision process is defined by the tuple $(\mathcal{T}, \mathcal{X}, \mathcal{X}_0, \mathcal{X}_f, \mathcal{A}, \mathcal{P}, \mathcal{R})$ where:

- $\mathcal{T}$ is the set of instants of decision, $t \in \mathcal{T}$.
- $\mathcal{X}$ is a finite set of states, $x \in \mathcal{X}$.
- $\mathcal{X}_0 \subset \mathcal{X}$: set of initial states.
- $\mathcal{X}_f \subset \mathcal{X}$: set of terminal states (may be empty).
- $\mathcal{A}$ is a finite set of actions, $a \in \mathcal{A}$.
- $\mathcal{P}$ is the transition function.
- $\mathcal{R}$ is the return function, $r \in \mathbb{R}$.

Remarks:

1. items depend on t and do not depend on $t-1$, $t-2$, ...: Markov property.
2. None of these items depend on $\mathcal{T}$: stationary system (= non autonomous).

Why would we take an action or an other? What's the point?

Why would we take an action or an other? What's the point?

An MD process only specifies the dynamics of a system that takes decision: it describes the "how", not the "why".

Markov decision process

Example

Let's play "21 with a dice".

Rules:

- you need 1 "standard" dice with 6 faces numbered from 1 to 6.
- You roll the dice, and note the value on its upper face: that's your initial score.
- Now repeatedly, you decide whether you roll it again or you stop the game.
- If you roll the dice, you add the value on its upper face to your current score.

Define a Markov decision process that models this dynamical system.

Markov decision problem

The “why”.

Markov decision problem

- A Markov decision problem describes a sequential decision problem on top of a Markov decision process.

Markov decision problem

- A Markov decision problem describes a sequential decision problem on top of a Markov decision process.
- The MD process defines how the dynamical system behaves in reaction to actions.

Markov decision problem

- A Markov decision problem describes a sequential decision problem on top of a Markov decision process.
- The MD process defines how the dynamical system behaves in reaction to actions.
- The MD problem defines the objective of the agent:
  $\rightarrow$ we need to define an objective function.

Markov decision problem

- A Markov decision problem describes a sequential decision problem on top of a Markov decision process.
- The MD process defines how the dynamical system behaves in reaction to actions.
- The MD problem defines the objective of the agent:
  $\rightarrow$ we need to define an objective function.
- Very often in the RL academic litterature, it is defined as maximizing $\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$, where $\gamma \in [0, 1)$.

Markov decision problem

- A Markov decision problem describes a sequential decision problem on top of a Markov decision process.
- The MD process defines how the dynamical system behaves in reaction to actions.
- The MD problem defines the objective of the agent:
  $\rightarrow$ we need to define an objective function.
- Very often in the RL academic litterature, it is defined as maximizing $\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$, where $\gamma \in [0, 1)$.
- Solution of an MD problem: a policy π that specifies the probability to perform any action a in any state x in order to optimize ζ.
  $\pi(x, a) = Pr[a_t = a | x_t = x], \forall a \in \mathcal{A}$.

Markov decision problem

- A Markov decision problem describes a sequential decision problem on top of a Markov decision process.
- The MD process defines how the dynamical system behaves in reaction to actions.
- The MD problem defines the objective of the agent:
  $\rightarrow$ we need to define an objective function.
- Very often in the RL academic litterature, it is defined as maximizing $\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$, where $\gamma \in [0, 1)$.
- Solution of an MD problem: a policy π that specifies the probability to perform any action a in any state x in order to optimize ζ.
  $\pi(x, a) = Pr[a_t = a | x_t = x], \forall a \in \mathcal{A}$.
- Blackwell's theorem tells us that an optimal policy (for this ζ) is determinitic: $\pi(x)$.
  More than 1 action may be optimal is a state.

Markov decision problem

Remarks about the definition

γ?

Markov decision problem

Remarks about the definition

γ?

$\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$

Markov decision problem

Remarks about the definition

γ?

$\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$

$\rightarrow \zeta(x_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$

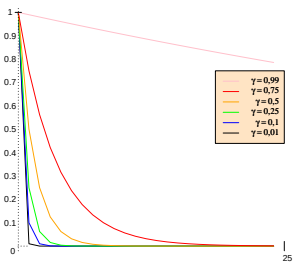
Markov decision problem

Remarks about the definition

γ?

$\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$

$\rightarrow \zeta(x_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$



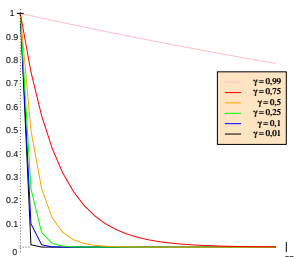
Markov decision problem

Remarks about the definition

γ?

$\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$

$\rightarrow \zeta(x_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$



- Nice property: If we assume $\mathcal{R}$ is bounded, then ζ converges.

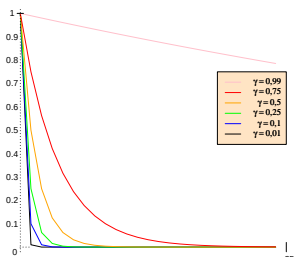
Markov decision problem

Remarks about the definition

γ?

$\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$

$\rightarrow \zeta(x_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$



- Nice property: If we assume $\mathcal{R}$ is bounded, then ζ converges.
- This definition makes the maths easier.

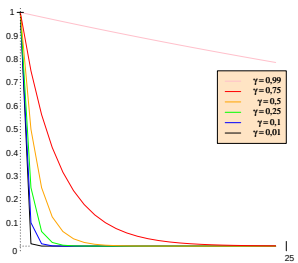
Markov decision problem

Remarks about the definition

γ?

$\zeta(x_t) = \sum_{k \geq 0} \gamma^k r_{t+k}$

$\rightarrow \zeta(x_t) = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + ...$



- Nice property: If we assume $\mathcal{R}$ is bounded, then ζ converges.
- This definition makes the maths easier.
- Short term consequences vs. longer consequences of actions.

Markov decision problem

A Markov decision process defines the "how": how does the system evolve in time?

A Markov decision problem defines the "why": why would the agent choose one or an other action?
Answer: to maximize ζ.

Pending questions:

- Does this problem have a solution?
- Can we compute it? In practice?
- How?

- An "episode" is a sequence of interactions starting from an initial state to a final state.

- An "episode" is a sequence of interactions starting from an initial state to a final state.
- A "trajectory" is the set of (x_t, a_t, r_t, x_{t+1}) for all t during an episode.

- An "episode" is a sequence of interactions starting from an initial state to a final state.
- A "trajectory" is the set of (x_t, a_t, r_t, x_{t+1}) for all t during an episode.
- Let $x_0 \in \mathcal{X}_0$.

- An "episode" is a sequence of interactions starting from an initial state to a final state.
- A "trajectory" is the set of (x_t, a_t, r_t, x_{t+1}) for all t during an episode.
- Let $x_0 \in \mathcal{X}_0$.
- Two episodes starting in x_0 will usually follow different trajectories.

Markov decision problem

The value function

- An "episode" is a sequence of interactions starting from an initial state to a final state.
- A "trajectory" is the set of (x_t, a_t, r_t, x_{t+1}) for all t during an episode.
- Let $x_0 \in \mathcal{X}_0$.
- Two episodes starting in x_0 will usually follow different trajectories.
- $\zeta(x), x \in \mathcal{X}_0$ is a random variable.

- An "episode" is a sequence of interactions starting from an initial state to a final state.
- A "trajectory" is the set of (x_t, a_t, r_t, x_{t+1}) for all t during an episode.
- Let $x_0 \in \mathcal{X}_0$.
- Two episodes starting in x_0 will usually follow different trajectories.
- $\zeta(x), x \in \mathcal{X}_0$ is a random variable.
- We define $V^\pi(x)$, the value of a state x according to a policy π by:

$$V^\pi(x) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x) | x_0 = x, a_{t \geq 0} \sim \pi]$$

the actions being chosen according to policy π.

Markov decision problem

The value function: intuition

$$V^\pi(x) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t\geq 0} \sim \pi]$$

$V^\pi(x)$ simply quantifies how good it is to be in state x while behaving according to policy π in order to optimize ζ.

$$V^\pi(x) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t\geq 0} \sim \pi]$$

$V^\pi(x)$ simply quantifies how good it is to be in state x while behaving according to policy π in order to optimize ζ.

We will soon see that given V^π, we can improve the policy and obtain a policy which value is better.

Markov decision problem

Evaluation of the value of a policy

$$V^\pi(x) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t\geq 0} \sim \pi]$$

...

$$V^\pi(x) = \sum_{a\in\mathcal{A}} \pi(x,a) \sum_{x'\in\mathcal{X}} \mathcal{P}(x,a,x')[\mathcal{R}(x,a,x') + \gamma V^\pi(x')]$$

$$V^\pi(x) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t\geq 0} \sim \pi]$$

...

$$V^\pi(x) = \sum_{a\in\mathcal{A}} \pi(x,a) \sum_{x'\in\mathcal{X}} \mathcal{P}(x,a,x')[\mathcal{R}(x,a,x') + \gamma V^\pi(x')]$$

This may look complicated but all terms are known except the vector V.

Markov decision problem

Evaluation of the value of a policy

$V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x,a) \sum_{x' \in \mathcal{X}} \mathcal{P}(x,a,x')[\mathcal{R}(x,a,x') + \gamma V^\pi(x')]$

Development $\rightarrow$ a system of N linear equations with N unknowns, the $V^\pi(x), \forall x \in \mathcal{X}$.

In principle, an easy problem.
In practice, when N is large (*e.g.* 10^9), this is a challenging problem.

$V^\pi = \mathcal{M} V^\pi + Q$, where $\mathcal{M} \in \mathbb{R}^{N \times N}$ is an $N \times N$ matrix, and $Q \in \mathbb{R}^N$.
$\rightarrow (Id - \mathcal{M}) V^\pi = Q$

Thanks to contraction properties of $\mathcal{M}$, this can be solved iteratively.

$$V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x, a) \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma V^\pi(x')]$$

Thanks to contraction properties of $\mathcal{M}$, this can be solved iteratively.

This algorithm provides an estimation of V^π at most ϵ away from its true value.

It is very easy to implement and very fast, even when $N = 10^9$.

Markov decision problem

Evaluation of the value of a policy

$$V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x,a) \sum_{x' \in \mathcal{X}} \mathcal{P}(x,a,x')[\mathcal{R}(x,a,x') + \gamma V^\pi(x')]$$

is known as a "Bellman equation".

[Bellman, *Dynamic Programming*, Princeton U. Press, 1957]

Markov decision problem

Example (continued)

- Once the value of a policy has been estimated, the policy can be improved.

- Once the value of a policy has been estimated, the policy can be improved.
- Let us assume we estimated V^π. Then, we compute:

$$\pi'(x) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{x'} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma V^\pi(x')]$$

for each state $x \in \mathcal{X}$.

- Once the value of a policy has been estimated, the policy can be improved.
- Let us assume we estimated V^π. Then, we compute:

$$\pi'(x) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{x'} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma V^\pi(x')]$$

  for each state $x \in \mathcal{X}$.
- Then π' is either better than π, or they are equivalent, that is: $V^{\pi'} \geq V^\pi$.

- Start with a random policy.
- Then, alternate:
  - estimate the value of the current policy
  - improve the current policy

  until the value of the policy no longer improves.

This is the "policy iteration" algorithm [Howard, 1958].

Markov decision problem

Value iteration

- The value V^* of the optimal policy π^* is:
  $V^*(x) \stackrel{\text{def}}{=} \max_\pi V^\pi(x), \forall x \in \mathcal{X}$

Markov decision problem

Value iteration

- The value V^* of the optimal policy π^* is:
  $V^*(x) \stackrel{\text{def}}{=} \max_\pi V^\pi(x), \forall x \in \mathcal{X}$
- It follows:
  $V^*(x) = \max_{a \in \mathcal{A}} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma V^*(x')]$
  The "Bellman optimality equation".

Markov decision problem

Value iteration

- The value V^* of the optimal policy π^* is:
  $V^*(x) \stackrel{\text{def}}{=} \max_\pi V^\pi(x), \forall x \in \mathcal{X}$
- It follows:
  $V^*(x) = \max_{a \in \mathcal{A}} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma V^*(x')]$
  The "Bellman optimality equation".
- and an algorithm to obtain V^* directly:
  - $k \leftarrow 0$
  - initialize V_k
    - compute $V_{k+1} \leftarrow \max_{a \in \mathcal{A}} \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma V^*(x')]$ for all x
    - $k \leftarrow k + 1$
  - repeat
  - until $||V_k - V_{k-1}||_\infty \leq \frac{\epsilon(1-\gamma)}{2\gamma}$

  named "value iteration".

Markov decision problem

As a linear program

We can frame a Markov decision problem as a linear program:

$$
\begin{aligned}
&min \sum_x V[x] \\
&\text{s.t. } V[x] - \sum_{x'} \mathcal{P}(x, a, x')(\mathcal{R}(x, a, x')) + \gamma V[x']) \geq 0 \ \forall (x, a) \in \mathcal{X} \times \mathcal{A}
\end{aligned}
$$

There one constraint for each (state, action) pair and one variable per state.
The dual is:

$$
\begin{aligned}
&max \sum_{(x,a)} \sum_{x'} \mathcal{P}(x, a, x')\mathcal{R}(x, a, x')\xi(x, a) \\
&\text{s.t. } \sum_a \xi(x', a) - \sum_{x,a} \gamma \mathcal{P}(x, a, x')\xi(x, a) \leq 1, \forall x' \in \mathcal{X} \\
&\text{and } \xi(x, a) \geq 0, \forall (x, a) \in \mathcal{X} \times \mathcal{A}
\end{aligned}
$$

Once solved, $\xi(x, a)$ is non zero if action a is optimal in x.

Very interesting theoretically speaking.
In practice, policy iteration is (usually) the best way to go.

Markov decision problem

Example (continued)

From MDPs to Reinforcement Learning

What if $\mathcal{P}$ and $\mathcal{R}$ are unknown?

- Reminder: $V^\pi \overset{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t>0} \sim \pi]$

  Let us define $Q^\pi(x, a) \overset{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_0 = a, a_{t>0} \sim \pi]$

  named the "quality" of the (state, action) pair (x, a).

- Reminder: $V^\pi \overset{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t>0} \sim \pi]$
  Let us define $Q^\pi(x,a) \overset{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_0 = a, a_{t>0} \sim \pi]$

  named the "quality" of the (state, action) pair (x,a).
- We have: $V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x,a) Q^\pi(x,a)$.

- Reminder: $V^\pi \overset{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t>0} \sim \pi]$
  Let us define $Q^\pi(x,a) \overset{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_0 = a, a_{t>0} \sim \pi]$

  named the "quality" of the (state, action) pair (x,a).
- We have: $V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x,a) Q^\pi(x,a)$.
- Bellman equation for Q:
  $Q^\pi(x,a) =$
  $\sum_{x' \in \mathcal{X}} \mathcal{P}(x,a,x')[\mathcal{R}(x,a,x') + \gamma \sum_{a' \in \mathcal{A}} \pi(x',a') Q^\pi(x',a')]$.

- Reminder: $V^\pi \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t>0} \sim \pi]$
  Let us define $Q^\pi(x,a) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_0 = a, a_{t>0} \sim \pi]$

  named the "quality" of the (state, action) pair (x,a).
- We have: $V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x,a) Q^\pi(x,a)$.
- Bellman equation for Q:
  $Q^\pi(x,a) =$
  $\sum_{x' \in \mathcal{X}} \mathcal{P}(x,a,x')[\mathcal{R}(x,a,x') + \gamma \sum_{a' \in \mathcal{A}} \pi(x',a') Q^\pi(x',a')]$.
- As for V, we may define the optimal quality that is the quality of the optimal policy: $Q^* \stackrel{\text{def}}{=} max_\pi Q^\pi$.

- Reminder: $V^\pi \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_{t>0} \sim \pi]$
  Let us define $Q^\pi(x, a) \stackrel{\text{def}}{=} \mathbb{E}[\zeta(x)|x_0 = x, a_0 = a, a_{t>0} \sim \pi]$

  named the "quality" of the (state, action) pair (x, a).
- We have: $V^\pi(x) = \sum_{a \in \mathcal{A}} \pi(x, a) Q^\pi(x, a)$.
- Bellman equation for Q:
  $Q^\pi(x, a) =$
  $\sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma \sum_{a' \in \mathcal{A}} \pi(x', a') Q^\pi(x', a')]$.
- As for V, we may define the optimal quality that is the quality of the optimal policy: $Q^* \stackrel{\text{def}}{=} max_\pi Q^\pi$.
- Bellman optimality equation for Q^*:
  $Q^*(x, a) = \sum_{x' \in \mathcal{X}} \mathcal{P}(x, a, x')[\mathcal{R}(x, a, x') + \gamma \max_{a' \in \mathcal{A}} Q^*(x', a')]$.

Bellman approach:

- $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$

Bellman approach:

- $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- the value of a state is: $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  Reminder: This quantifies what will happen to the agent in its future if it behaves according to π.

Bellman approach:

- $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- the value of a state is: $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  Reminder: This quantifies what will happen to the agent in its future if it behaves according to π.
- re-written as: $V(x_t|\pi) = \mathbb{E}(r_t) + \gamma \mathbb{E}(V(x_{t+1}|\pi))$

  sum of what will happen immediately $+ \gamma \times$ what will happen then.

Reinforcement Learning

Bellman equation and the TD error [Sutton, 1988]

Bellman approach:

- $\zeta = \sum_{t \geq 0} \gamma^t r_t, \gamma \in [0, 1[$
- the value of a state is: $V(x) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x|\pi))$

  Reminder: This quantifies what will happen to the agent in its future if it behaves according to π.
- re-written as: $V(x_t|\pi) = \mathbb{E}(r_t) + \gamma \mathbb{E}(V(x_{t+1}|\pi))$

  sum of what will happen immediately $+ \gamma \times$ what will happen then.
- at time t, $r_t + \gamma(V(x_{t+1}|\pi)) - V(x_t|\pi)$

  is an estimation of the error of estimation of V: TD-error

  This TD-error may be used to learn the optimal behavior.

- computing V by gradient descent:

  $V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$
  where η is a learning rate, adequately decreasing along time.

- computing V by gradient descent:

  $V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$
  where η is a learning rate, adequately decreasing along time.
- We may also consider the quality of an (x, a) pair:
  $Q(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi))$

- computing V by gradient descent:

  $V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$
  where η is a learning rate, adequately decreasing along time.
- We may also consider the quality of an (x, a) pair:
  $Q(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi))$
- At t, we may consider $r_t + \gamma \max_{a'} Q(x_{t+1}, a') - Q(x_t, a_t)$ as a correction to $Q(x_t, a_t)$.
  $\rightsquigarrow$
  $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \eta[r_t + \gamma \max_b Q(x_{t+1}, b) - Q(x_t, a_t)]$

- computing V by gradient descent:

  $V(x_{t+1}) \leftarrow V(x_t) - \eta[r_t + \gamma(V_t(x_{t+1})) - V(x_t)]$
  where η is a learning rate, adequately decreasing along time.
- We may also consider the quality of an (x, a) pair:
  $Q(x_t, a_t) \stackrel{\text{def}}{=} \mathbb{E}(\zeta(x_t | a_t = a, \pi))$
- At t, we may consider $r_t + \gamma \max_{a'} Q(x_{t+1}, a') - Q(x_t, a_t)$ as a correction to $Q(x_t, a_t)$.

  $\rightsquigarrow$

  $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \eta[r_t + \gamma \max_b Q(x_{t+1}, b) - Q(x_t, a_t)]$

- $\rightsquigarrow$ learning π^* algorithm.

The goal of this algorithm is to estimate Q^* by interacting with the environment and correcting its estimate of Q^*.

1. initialize Q.
2. set the agent in some random initial state $\in \mathcal{X}_0$.
3. run the agent in the environment:
   at each step, record the state x_t, the action performed a_t, the reward collected r_t, and the next state x_{t+1}.
4. correct Q.
5. continue until a terminal state is reached.
6. do it again and again (re-starting at step 2).

Reinforcement Learning

Q-Learning [Watkins, 1989]

- $Q(x, a) \leftarrow$ some value (0, random, ...)
- **Repeat**
  - $t \leftarrow 0$
  - Initialize the state of the agent x_t
  - **while** episode not completed, **do:**
    - choose an action to perform in state x_t: a_t
    - perform this action and observe r_t and x_{t+1}
    - update $Q(x, a)$:
      $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \alpha[r_t + \max Q(x_{t+1}, b_b - Q(x_t, a_t)]$
    - $t++$
- **Until** some condition is met.

At the completion of this algorithm (if you looped enough):
$\pi^*(x) = \arg\max_a Q(x, a), \forall x$

At the completion of this algorithm (if you looped enough):
$\pi^*(x) = \arg\max_a Q(x, a), \forall x$

Remark: you never know if you looped enough!

Asymptotic convergence to Q^*.

- "choose an action to perform in state x_t: a_t"
  How do we do that?

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.
- Intuition:

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.
- Intuition:
  - initially, we do not know anything about Q^*. We have to test the effect of the actions: we have to explore.

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.
- Intuition:
  - initially, we do not know anything about Q^*. We have to test the effect of the actions: we have to explore.
  - step by step, we acquire some knowledge about which actions are good, and which are bad. We can exploit this knowledge.

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.
- Intuition:
  - initially, we do not know anything about Q^*. We have to test the effect of the actions: we have to explore.
    Select action at random
  - step by step, we acquire some knowledge about which actions are good, and which are bad. We can exploit this knowledge.

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.
- Intuition:
  - initially, we do not know anything about Q^*. We have to test the effect of the actions: we have to explore.
    Select action at random
  - step by step, we acquire some knowledge about which actions are good, and which are bad. We can exploit this knowledge.
    Select the "best" action: arg $\max_a Q(x_t, a)$.

- "choose an action to perform in state x_t: a_t"
  How do we do that?
- No perfect solution.
- Intuition:
  - initially, we do not know anything about Q^*. We have to test the effect of the actions: we have to explore.
    Select action at random
  - step by step, we acquire some knowledge about which actions are good, and which are bad. We can exploit this knowledge.
    Select the "best" action: $\arg\max_a Q(x_t, a)$.
- Exploration and exploitation should be carefully balanced.

- "choose an action to perform in state x_t: a_t".

Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.

Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.

Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:
  - $\tau \leftarrow 1000$.

Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:
  - $\tau \leftarrow 1000$.
  - Each time you need to select an action:

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:
  - $\tau \leftarrow 1000$.
  - Each time you need to select an action:
    - compute $p(a) \leftarrow \frac{e^{\frac{Q(x_t,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s_t,a')}{\tau}}}$,

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:
  - $\tau \leftarrow 1000$.
  - Each time you need to select an action:
    - compute $p(a) \leftarrow \frac{e^{\frac{Q(x_t,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s_t,a')}{\tau}}}$,
    - draw an action at random according to p.

Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:
  - $\tau \leftarrow 1000$.
  - Each time you need to select an action:
    - compute $p(a) \leftarrow \frac{e^{\frac{Q(x_t,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s_t,a')}{\tau}}}$,
    - draw an action at random according to p.
  - Slowly decrease τ along the episodes.

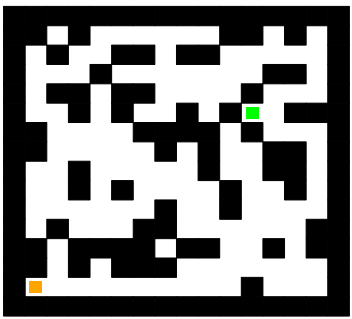
Reinforcement Learning

About Q-Learning

- "choose an action to perform in state x_t: a_t".
- explore and progressively exploit more and more.

- The ϵ-decreasing greedy strategy:
  - $\epsilon \leftarrow 1$.
  - Select action $\arg\max_a Q(x_t, a)$ with probability ϵ and a random action otherwise.
  - Slowly decrease ϵ along the episodes.

- The softmax strategy:
  - $\tau \leftarrow 1000$.
  - Each time you need to select an action:
    - compute $p(a) \leftarrow \frac{e^{\frac{Q(x_t,a)}{\tau}}}{\sum_{a' \in \mathcal{A}} e^{\frac{Q(s_t,a')}{\tau}}}$,
    - draw an action at random according to p.
  - Slowly decrease τ along the episodes.
  - Rationale: when τ is large, uniformly random selection. τ close to 0, greedy selection.

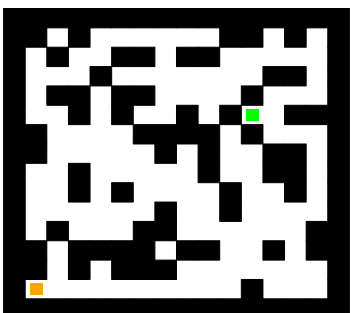


Question 1: propose a Markov decision process: what are ($\mathcal{T}$, $\mathcal{X}$, $\mathcal{X}_0$, $\mathcal{X}_f$, $\mathcal{A}$, $\mathcal{P}$, $\mathcal{R}$)?
Question 2: model this task as a Markov decision problem: what is ζ?

Reinforcement Learning

Q-Learning in action: escaping a maze

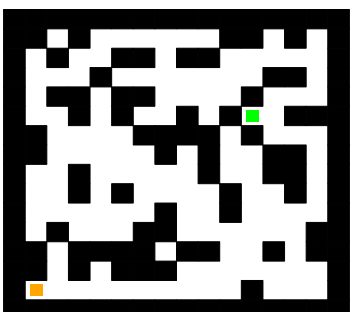
We use an extremely basic Q-Learning.
Has a very local perception: sees only the 4 neighboring cells.

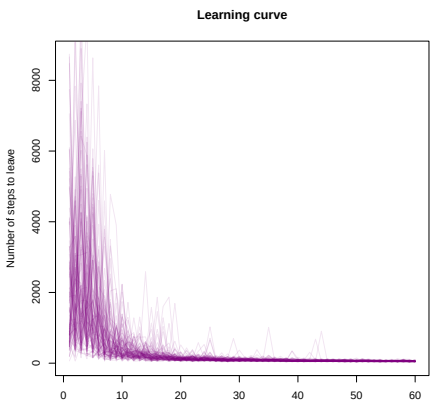


Reinforcement Learning

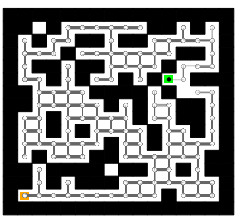
Q-Learning in action: escaping a maze

We use an extremely basic Q-Learning.
Has a very local perception: sees only the 4 neighboring cells.





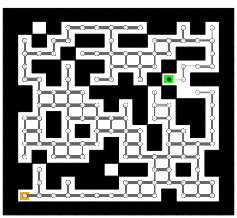
1st reach



Reinforcement Learning

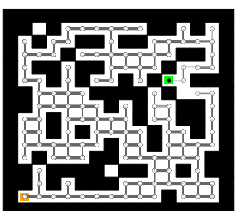
Q-Learning in action

1st reach

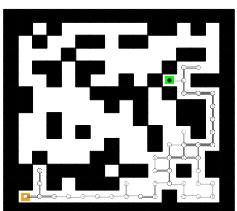


10th reach

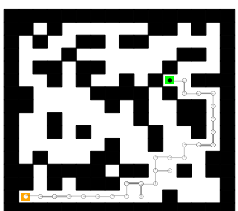
1st reach



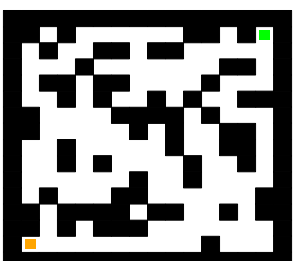
10th reach

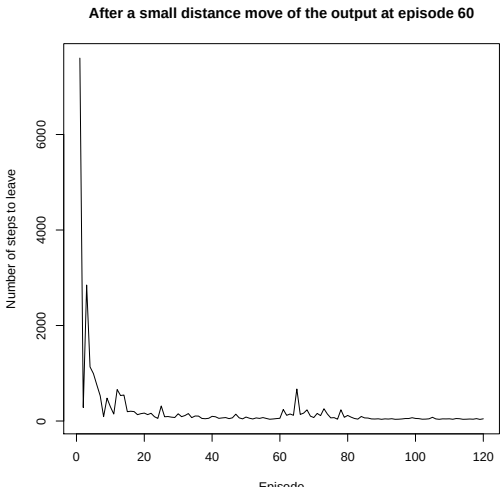


60th reach

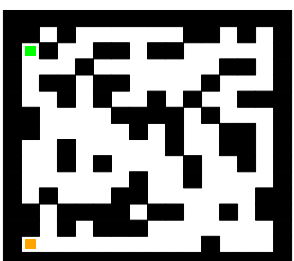


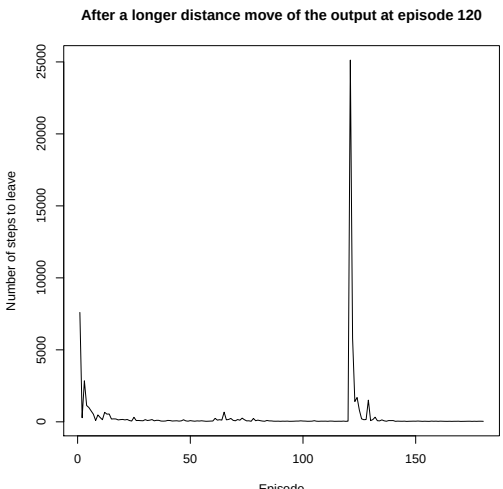
The goal state moves nearby:



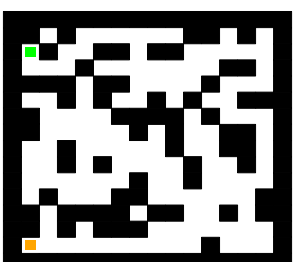


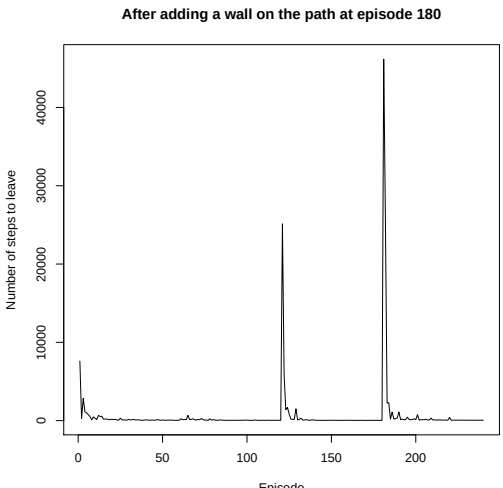
The goal state moves farther away:





Blocking the path:





- This is the "tabular" Q-Learning: Q is represented in a "table".

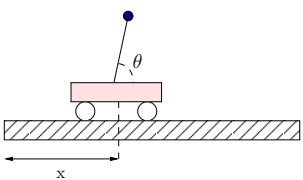
- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.
- Use a function approximator, that is, replace the table `Q [x, a]` by a function `Q (x, a)`.

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.
- Use a function approximator, that is, replace the table `Q [x, a]` by a function `Q (x, a)`.
- `Q (x, a)` returns an estimate of $Q(x, a)$.

- This is the "tabular" Q-Learning: Q is represented in a "table".
- What about large $\mathcal{X}$?
- Impossible to store Q in a table.
- Use a function approximator, that is, replace the table `Q [x, a]` by a function `Q (x, a)`.
- `Q (x, a)` returns an estimate of $Q(x, a)$.
- This estimate may be updated/improved by learning.

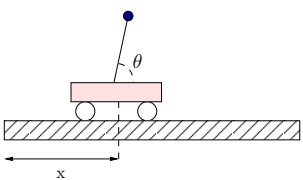


State is $(\theta, \dot{\theta})$

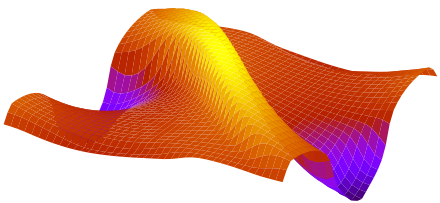
Action is $\ddot{\theta}$

Reinforcement Learning

Value function



State is $(\theta, \dot{\theta})$
Action is $\ddot{\theta}$



$(\theta, \dot{\theta})$ plane
z is $V(x)$
Maximize value $\rightsquigarrow$ reach the top of V

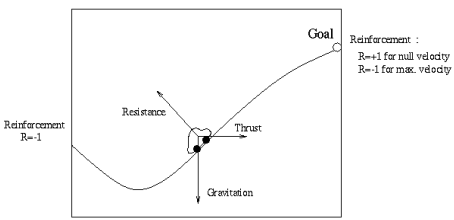
Reinforcement Learning

Handling large $\mathcal{X}$: the function approximator zoo

- neural network [Lin, 1991; Riedmiller, 2005; ...],
- random forest [Geurts *et al.*, 2006],
- SVM and kernels,
- and many other ideas from statistical learning (supervised learning).
- Tabular with progressive and adaptive state partitioning.

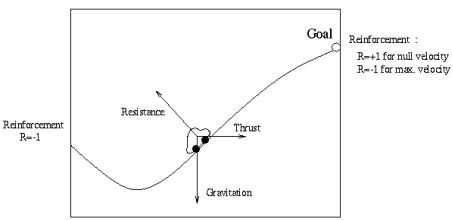
Reinforcement Learning

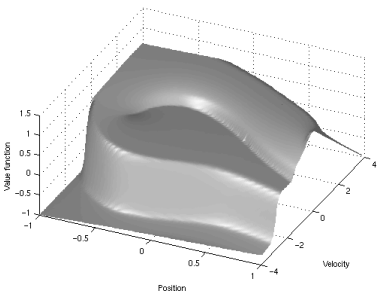
Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]



Reinforcement Learning

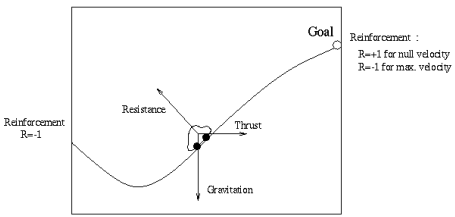
Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

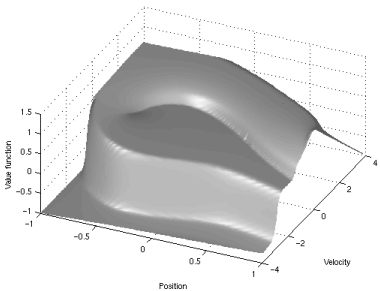


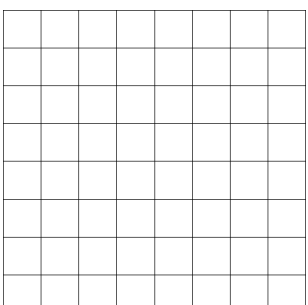


Reinforcement Learning

Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

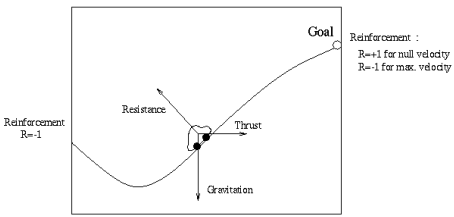


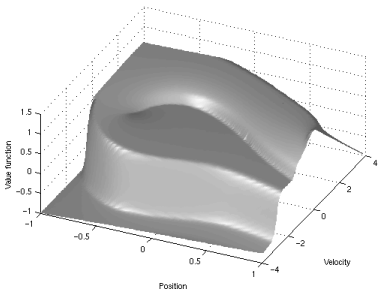


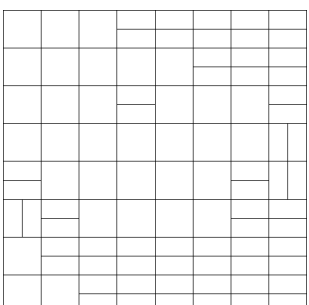


Reinforcement Learning

Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]

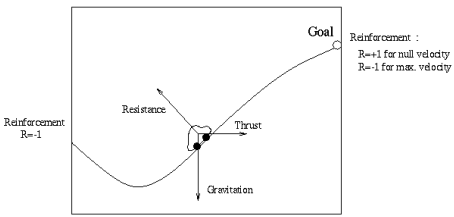


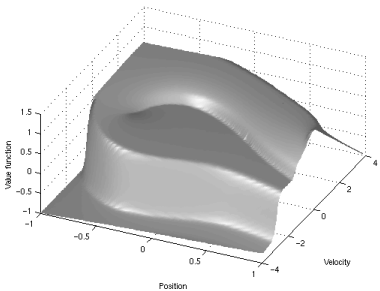


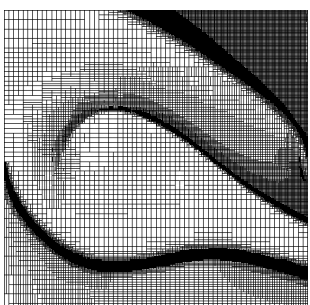


Reinforcement Learning

Progressive and adaptive state partitioning [Munos, Moore, MLJ, 2001]







Trendy people call that "deep RL" though there is only shallow networks most of the times.

- Use of a neural network to represent the Q table.
- Input = the current state
- Output = either an estimate of $Q(x, a), \forall a$, or an estimate of $\pi(x)$.

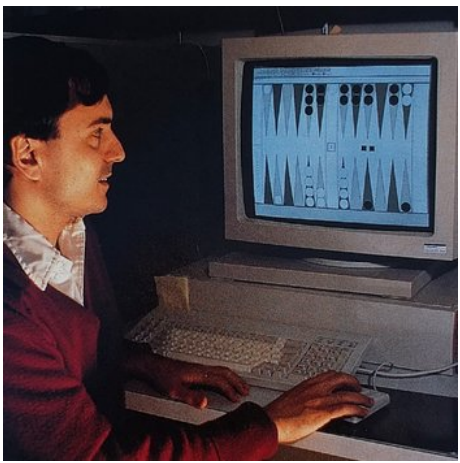
Reinforcement Learning

Neural RL

-
-

Reinforcement Learning

Application: TD-Gammon



Reinforcement Learning

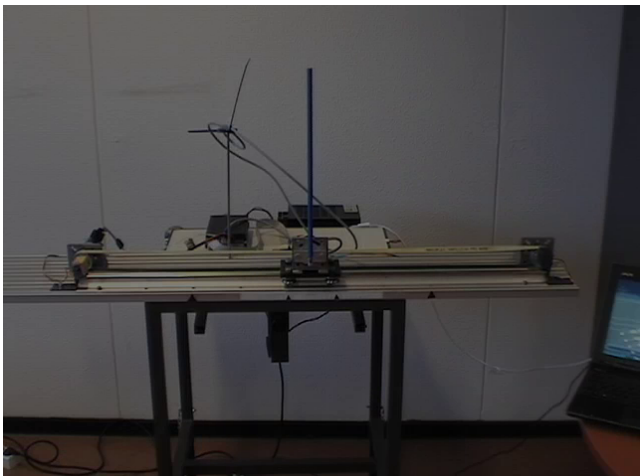
Application: TD-Gammon

- Backgammon is studied at least since 1974
- Branching factor: 800
- TD-Gammon: "successor of Neurogammon, trained by superviser learning. NeuroGammon won the 1st Computer Olympiad in London in 1989, handily defeating all opponents. Its level of play was that of an intermediate-level human player." (Source: wikipedia)
- raw representation of the board position
- trained with TD(λ) algorithm
- no knowledge, self-play
- hand-crafted features
- 3-plies in v3

Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, 1995

Reinforcement Learning

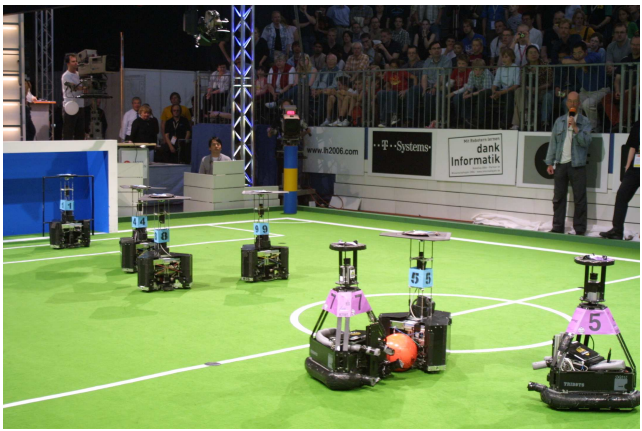
Application to robotics



Riedmiller, Neural reinforcement learning to swing-up and balance a real pole, *Proc. 2005 IEEE International Conference on Systems, Man and Cybernetics*

Reinforcement Learning

Application to robotics



Lauer et al., Cognitive Concepts in Autonomous Soccer Playing Robots, *Cognitive Systems Research*, **11**(3), 287:309, September, 2010
(No Deep Learning! only shallow multi-layer perceptron)

Reinforcement Learning

Application: board games

- Learning to play board games using only the rules of the game
- Alpha Go learned to play Go by using games played by humans
- Alpha Zero learned to play even better by itself by RL.
- then other board games (chess, draughts, reversi, ...)
- then Starcraft II

Reinforcement Learning

Alpha Zero type of algorithms

- RL
- + various tricks to stabilize learning and make it more efficient
- MCTS as a key component
- moderately deep network as function approximator

Outro

- learning options
- learning representation
- generalization in RL
- time varying environments
- transfer learning
- life-long learning
- explaination/accountability of the learned behavior

Take home message

- Many problems can benefit from a sequential decision making point of view.




Not only games.

Take home message

- Many problems can benefit from a sequential decision making point of view.




  Not only games.

- Reinforcement learning outperforms supervised learning.

Bibliography

- Bertsekas, A Course in Reinforcement Learning, Athena Scientific, 2023, available on the web for free `http://web.mit.edu/dimitrib/www/RLbook.html`
- Lapan, *Practical Deep Reinforcement Learning*, Packt, 2018
- Sutton and Barto, *Reinforcemnt Learning*, 2nd ed, MIT Press, 2018, `http://incompleteideas.net/book/the-book-2nd.html`
- Silver *et al.*, Mastering the game of Go without human knowledge, Nature, **550**, 2017
- Silver *et al.*, Mastering the game of Go with deep neural networks and tree search, Nature, **529**, 2016
- Tesauro, Temporal Difference Learning and TD-Gammon, *Communications of the ACM*, 1995