Data-driven and Learning-based Control

Learning in Control

Erica Salvato

UNIVERSITÀ DEGLI STUDI DI TRIESTE

Table of Contents

- A brief recap
- Machine-Learning
- Introduction to Reinforcement Learning
- Monte Carlo Learning
- Temporal difference learning

What we know so far?

We are now able to solve optimal control problems in the case of dynamics that are:

- Deterministic (linear or non-linear)
  - Dynamic Programming
  - Policy iteration
  - Value-iteration
- Stochastic $\longrightarrow$ **Markov Decision Process**
  - Policy iteration
  - Value-iteration
- Known.

What we know so far?

We are now able to solve optimal control problems in the case of dynamics that are:

- Deterministic (linear or non-linear)
  - Dynamic Programming
  - Policy iteration
  - Value-iteration
- Stochastic $\rightarrow$ **Markov Decision Process**
  - Policy iteration
  - Value-iteration
- Known.

Now we need to address the following:

2. What if **we don't know** the model?

What we know so far?

We are now able to solve optimal control problems in the case of dynamics that are:

- Deterministic (linear or non-linear)
  - Dynamic Programming
  - Policy iteration
  - Value-iteration
- Stochastic $\rightarrow$ **Markov Decision Process**
  - Policy iteration
  - Value-iteration
- Known.

Now we need to address the following:

2. What if **we don't know** the model? $\rightarrow$ **Learning**

Table of Contents

2 Machine-Learning

- A brief recap
- Machine-Learning
- Introduction to Reinforcement Learning
- Monte Carlo Learning
- Temporal difference learning

Machine-Learning concept

The **machine-learning paradigms** empower machines to autonomously acquire knowledge from data and takes place as a result of interaction with a physical system.

Machine-Learning concept

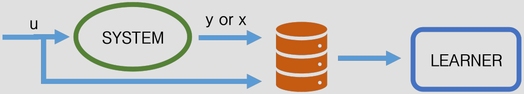
The **machine-learning paradigms** empower machines to autonomously acquire knowledge from data and takes place as a result of interaction with a physical system.

Knowledge gained from data can be expressed in terms of:

Supervised and Unsupervised Learning

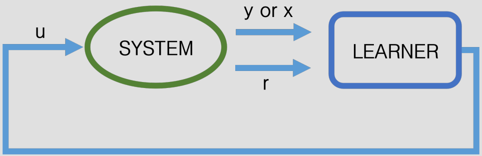
- understanding
- interpreting
- predicting

system behavior.

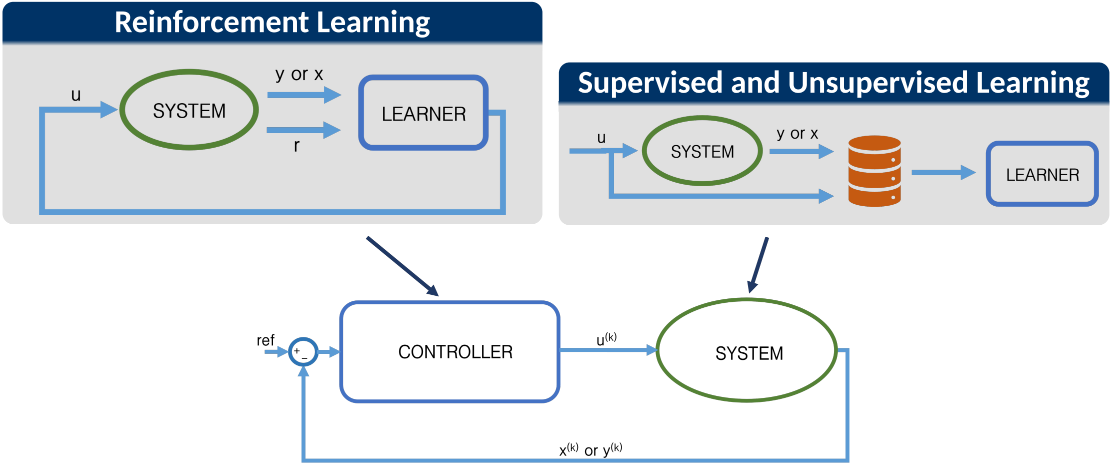


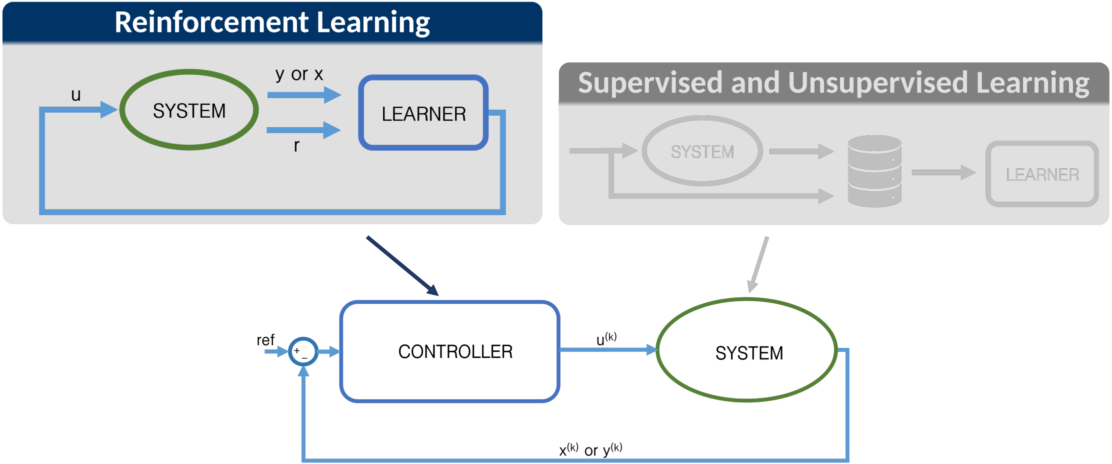
Reinforcement Learning

- learning decision-making strategies



Machine-Learning in control





Reinforcement Learning
u
SYSTEM
y or x
r
LEARNER
Supervised and Unsupervised Learning
SYSTEM
LEARNER
ref
+
_
CONTROLLER
u(k)
SYSTEM
x(k) or y(k)

Table of Contents

3 Introduction to Reinforcement Learning

- A brief recap
- Machine-Learning
- Introduction to Reinforcement Learning
- Monte Carlo Learning
- Temporal difference learning

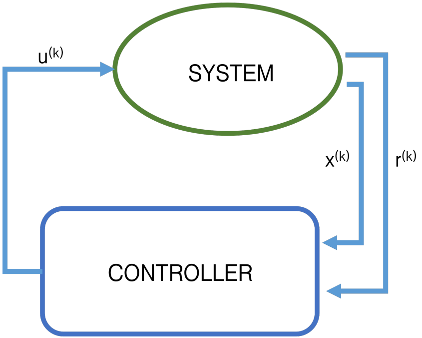
Reinforcement learning is a way to solve discrete-time, stochastic or deterministic optimal control problems based only on the data perceived by a system whose dynamical model is unknown $\rightarrow$ **Model-free control**

Reinforcement learning is a way to solve discrete-time, stochastic or deterministic optimal control problems based only on the data perceived by a system whose dynamical model is unknown $\rightarrow$ **Model-free control**

A typical terminology in RL is to call:

- the controller $\pi \rightarrow$ the **policy**
- the control input $u \rightarrow$ the **action**.

The only essential requirement is the **reward function** $\rightarrow$ no transition functions or transition probabilities are needed; data are acquired directly from the real system.



RL as Markov decision problem

- $x^{(k)} \in \mathcal{X} \subseteq \mathbb{R}^n$ the n-dimensional state
- $u^{(k)} \in \mathcal{U} \subseteq \mathbb{R}^m$ the m-dimensional input
- $k \in \mathbb{Z}_0^+$ the time-step index
- $h : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$ the reward function
- $\gamma \in [0, 1)$ the discount factor

- $x^{(k)} \in \mathcal{X} \subseteq \mathbb{R}^n$ the n-dimensional state
- $u^{(k)} \in \mathcal{U} \subseteq \mathbb{R}^m$ the m-dimensional input
- $k \in \mathbb{Z}_0^+$ the time-step index
- $h : \mathcal{X} \times \mathcal{U} \times \mathcal{X} \to \mathbb{R}$ the reward function
- $\gamma \in [0, 1)$ the discount factor

Reinforcement Learning (RL)

Solves the optimal control problem of finding the optimal controller $\pi^*\left(x^{(k)}\right)$ such that

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k h\left(x^{(k)}, \pi\left(x^{(k)}\right), x^{(k+1)}\right)\right]$$

The system is an MDP. However, we don't know T.
We directly receive the measure of $x^{(k+1)}$ once we apply $u^{(k)}$ when the measure of the previous state is $x^{(k)}$.

If you think of the policy iteration algorithm for MDPs:

Initialization. Select a guess $\pi_i = \pi_0$

Policy evaluation (PE). Determine the value of the current policy

- Select a guess $Q_j = Q_0$
- Repeat until $Q_{j+1} = Q_j$

  $\forall x^{(k)} \in \mathcal{X}$

  $Q_{j+1}\left(x^{(k)}, \pi_i\left(x^{(k)}\right)\right) =$

  $\sum_{x^{(k+1)}} T\left(x^{(k)}, \pi_i\left(x^{(k)}\right)\right)\left[h\left(x^{(k)}, \pi_i\left(x^{(k)}\right), x^{(k+1)}\right) + \gamma \max Q_j\left(x^{(k+1)}, \pi_i\left(x^{(k+1)}\right)\right)\right]$
- $Q_{\pi_i} = Q_{j+1} = Q_j$

Policy improvement (PI). Determine an improved policy

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i}$$

Terminal condition. PE and PI are repeated until $\pi_{i+1} = \pi_i$.

How to solve RL?

If you think of the policy iteration algorithm for MDPs:

Initialization. Select a guess $\pi_i = \pi_0$

Policy evaluation (PE). Determine the value of the current policy

- Select a guess $Q_j = Q_0$
- Repeat until $Q_{j+1} = Q_j$
  $\forall x^{(k)} \in \mathcal{X}$
  $Q_{j+1}\left(x^{(k)}, \pi_i\left(x^{(k)}\right)\right) = \sum_{x^{(k+1)}} T\left(x^{(k)}, \pi_i\left(x^{(k)}\right)\right)\left[h\left(x^{(k)}, \pi_i\left(x^{(k)}\right), x^{(k+1)}\right) + \gamma \max Q_j\left(x^{(k+1)}, \pi_i\left(x^{(k+1)}\right)\right)\right]$
- $Q_{\pi_i} = Q_{j+1} = Q_j$

Policy improvement (PI). Determine an improved policy

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i}$$

Terminal condition. PE and PI are repeated until $\pi_{i+1} = \pi_i$.

If you think of the policy iteration algorithm for MDPs:

Initialization. Select a guess $\pi_i = \pi_0$

Policy evaluation (PE). Determine the value of the current policy

- Select a guess $Q_j = Q_0$
- Repeat until $Q_{j+1} = Q_j$

  $\forall x^{(k)} \in \mathcal{X}$

  $Q_{j+1}\left(x^{(k)}, \pi_i\left(x^{(k)}\right)\right) =$

  $\sum_{x^{(k+1)}} T\left(x^{(k)}, \pi_i\left(x^{(k)}\right)\right)\left[h\left(x^{(k)}, \pi_i\left(x^{(k)}\right), x^{(k+1)}\right) + \gamma \max Q_j\left(x^{(k+1)}, \pi_i\left(x^{(k+1)}\right)\right)\right]$
- $Q_{\pi_i} = Q_{j+1} = Q_j$

Policy improvement (PI). Determine an improved policy

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i}$$

Terminal condition. PE and PI are repeated until $\pi_{i+1} = \pi_i$.

We need to find a way to solve the same problem without the need of T.

Trial and error paradigm

Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

At all k-th time-step:

Trial and error paradigm

Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

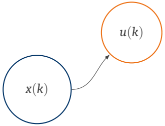
At all k-th time-step:

$x(k)$

Trial and error paradigm

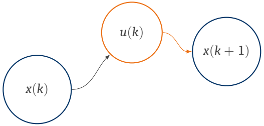
Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

At all k-th time-step:



Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

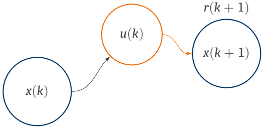
At all k-th time-step:



Trial and error paradigm

Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

At all k-th time-step:

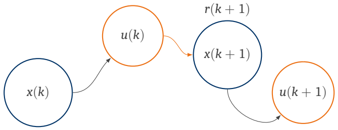


the reward determines whether or not the chosen action is good for achieving the goal and leverages this information for feature selection.

Trial and error paradigm

Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

At all k-th time-step:



the reward determines whether or not the chosen action is good for achieving the goal and leverages this information for feature selection.

However, RL cannot execute $\forall x$ policy evaluation in the already defined manner, but must execute a strategy to:

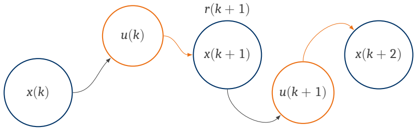
Exploration vs. exploitation

- **explore** new actions or states to discover their effects and potentially find better strategies
- **exploit** the information already gathered to maximize short-term gains

Trial and error paradigm

Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

At all k-th time-step:



the reward determines whether or not the chosen action is good for achieving the goal and leverages this information for feature selection.

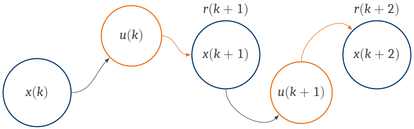
However, RL cannot execute $\forall x$ policy evaluation in the already defined manner, but must execute a strategy to:

Exploration vs. exploitation

- **explore** new actions or states to discover their effects and potentially find better strategies
- **exploit** the information already gathered to maximize short-term gains

Trial-and-error is a universal strategy for establishing which actions are beneficial or harmful in task achievement. The way in which it works can be summarized as follows:

At all k-th time-step:



the reward determines whether or not the chosen action is good for achieving the goal and leverages this information for feature selection.

However, RL cannot execute $\forall x$ policy evaluation in the already defined manner, but must execute a strategy to:

Exploration vs. exploitation

- **explore** new actions or states to discover their effects and potentially find better strategies
- **exploit** the information already gathered to maximize short-term gains

Table of Contents

4 Monte Carlo Learning

- A brief recap
- Machine-Learning
- Introduction to Reinforcement Learning
- Monte Carlo Learning
- Temporal difference learning

Episode

We define **episode** a finite-time simulation of a system, subject to a controller π, that starts from an initial condition $x^{(0)}$ and ends either when a **terminal condition** is met:

- $x^{(k)} = x_{\text{des}}$ is reached
- the maximum number of time-steps have been executed

Episode and Return

Episode

We define **episode** a finite-time simulation of a system, subject to a controller π, that starts from an initial condition $x^{(0)}$ and ends either when a **terminal condition** is met:

- $x^{(k)} = x_{\text{des}}$ is reached
- the maximum number of time-steps have been executed

Return

Given an episode of T steps, we define **return** the total discounted reward

$$G_k = \sum_{k \geq 0}^{T-1} \gamma^k \boldsymbol{r}^{(k+1)}$$

Episodic vs. Non-episodic tasks

In view of what has been studied so far:

Episodic RL task $\rightarrow$ Finite-horizon optimal control

Non-episodic RL task $\rightarrow$ Infinite-horizon optimal control

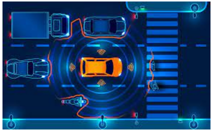
In view of what has been studied so far:

Episodic RL task $\rightarrow$ Finite-horizon optimal control

Non-episodic RL task $\rightarrow$ Infinite-horizon optimal control

Identify the nature of the tasks represented by the following examples:

Autonomous vehicle navigation:
learning effective strategies that adapt to continuous, evolving scenarios



Episodic vs. Non-episodic tasks

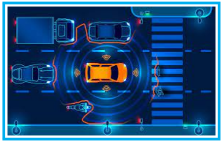
In view of what has been studied so far:

Episodic RL task $\rightarrow$ Finite-horizon optimal control

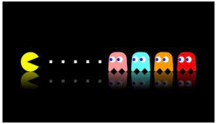
Non-episodic RL task $\rightarrow$ Infinite-horizon optimal control

Identify the nature of the tasks represented by the following examples:

Autonomous vehicle navigation:
learning effective strategies that adapt to continuous, evolving scenarios



Playing a video game:
learning effective strategies to win the game



Episodic vs. Non-episodic tasks

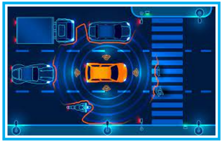
In view of what has been studied so far:

Episodic RL task $\rightarrow$ Finite-horizon optimal control

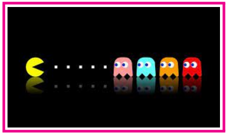
Non-episodic RL task $\rightarrow$ Infinite-horizon optimal control

Identify the nature of the tasks represented by the following examples:

Autonomous vehicle navigation:
learning effective strategies that adapt to continuous, evolving scenarios



Playing a video game:
learning effective strategies to win the game



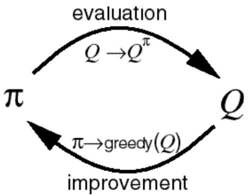
Monte Carlo is a model-free learning method that solves RL problems.
It relies on complete episode trajectories $(x^{(k)}, u^{(k)})$, $k = 0, \dots, T-1$ and their returns G_k, and on the simplest possible idea:

Value = mean of return

Monte Carlo is a model-free learning method that solves RL problems.
It relies on complete episode trajectories $(x^{(k)}, u^{(k)})$, $k = 0, \ldots, T-1$ and their returns G_k, and on the simplest possible idea:

Value = mean of return

Therefore, it is only defined for **episodic tasks** and uses the mean of return to perform a policy iteration approach:



- policy evaluation
- policy improvement

Here instead of computing value functions from the MDP knowledge, we **learn value functions from the sample returns**.

Exploration vs. exploitation

Recall that in RL, due to the absence of the model knowledge, we need to ensure a **trade-off** between exploration and exploitation.

In the policy iteration approach, we can observe that the policy improvement step is in a certain sense **exploiting** knowledge:

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i}$$

Therefore we need to understand how to include **exploration**

Monte Carlo Exploring starts

Exploring starts: every state-action pair has a non-zero probability of being the starting pair. Therefore, if we perform ∞ episodes, state-action pairs will be visited infinite time.

Initialization. Select a guess $\pi_i = \pi_0$, $Q_i = Q_0$. Initialize an empty list Return $(\cdot, \cdot)$

Policy evaluation (PE). Determine the value of the current policy

Choose $x^{(0)} \in \mathcal{X}$, $u^{(0)} \in \mathcal{U}$

Perform an episode following π_i

$\forall\, x^{(k)}, u^{(k)}$ in the episode:

- consider the first occurrence of each state $x^{(k)}, u^{(k)}$ and compute G_k
- append G_k to the list of return of $x^{(k)}, u^{(k)}$: Return $\left(x^{(k)}, u^{(k)}\right)$
- $Q\left(x^{(k)}, u^{(k)}\right) = \frac{1}{|\text{Return}\left(x^{(k)}, u^{(k)}\right)|} \sum_{l=1}^{|\text{Return}\left(x^{(k)}, u^{(k)}\right)|} \text{Return}\left(x^{(k)}, u^{(k)}\right)_l$

Monte Carlo Exploring starts

Exploring starts: every state-action pair has a non-zero probability of being the starting pair. Therefore, if we perform ∞ episodes, state-action pairs will be visited infinite time.

Initialization. Select a guess $\pi_i = \pi_0$, $Q_i = Q_0$. Initialize an empty list Return $(\cdot, \cdot)$

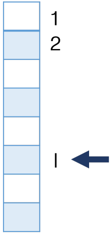
Policy evaluation (PE). Determine the value of the current policy

Choose $x^{(0)} \in \mathcal{X}$, $u^{(0)} \in \mathcal{U}$

Perform an episode following π_i

$\forall\, x^{(k)}, u^{(k)}$ in the episode:

- consider the first occurrence of each state $x^{(k)}, u^{(k)}$ and compute G_k
- append G_k to the list of return of $x^{(k)}, u^{(k)}$: Return $\left(x^{(k)}, u^{(k)}\right)$
- $Q\left(x^{(k)}, u^{(k)}\right) = \frac{1}{\left|\text{Return}\left(x^{(k)}, u^{(k)}\right)\right|} \sum_{l=1}^{\left|\text{Return}\left(x^{(k)}, u^{(k)}\right)\right|} \text{Return}\left(x^{(k)}, u^{(k)}\right)_l$



Policy improvement (PI). Determine an improved policy

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i}$$

PE and PI are repeated forever.

Policy improvement (PI). Determine an improved policy

$$\pi_{i+1} = \arg\max_{\pi} Q_{\pi_i}$$

PE and PI are repeated forever.

Notice that policy improvement is performed by making the policy (or controller) **greedy**, i.e., a policy that only exploits the already visited results. Formally speaking:

Greedy controller

A greedy controller is the one that, for each $x^{(k)} \in \mathcal{X}$, deterministically chooses

$$\pi\left(x^{(k)}\right) = \arg\max_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right)$$

Monte Carlo Exploring starts drawbacks

In order to be performed it needs:

- to use exploring starts to ensure exploration
- to perform infinite episodes

These are two unlikely assumptions in practice.

Soft policy

A soft policy is a stochastic policy that assigns a probability distribution over the set of possible actions for a given state

$$\pi\left(u^{(k)}|x^{(k)}\right) > 0 \quad \forall x^{(k)} \in \mathcal{X},\, u^{(k)} \in \mathcal{U}$$

Monte Carlo Exploring starts drawbacks

In order to be performed it needs:

- to use exploring starts to ensure exploration
- to perform infinite episodes

These are two unlikely assumptions in practice.

How can we avoid exploring starts and at the same time ensure exploration?

Soft policy

A soft policy is a stochastic policy that assigns a probability distribution over the set of possible actions for a given state

$$\pi\left(u^{(k)}|x^{(k)}\right) > 0 \quad \forall x^{(k)} \in \mathcal{X},\, u^{(k)} \in \mathcal{U}$$

Monte Carlo Exploring starts drawbacks

Among soft policies, we are interested in ϵ-soft policies, defined as soft policies for which each action has at least the probability $frac\epsilon|\mathcal{U}|$ of being selected:

$$\pi\left(\boldsymbol{u}^{(k)}|\boldsymbol{x}^{(k)}\right) \geq \frac{\epsilon}{|\mathcal{U}|} \quad \forall \boldsymbol{x}^{(k)} \in \mathcal{X},\, \boldsymbol{u}^{(k)} \in \mathcal{U}$$

Among soft policies, we are interested in ϵ-soft policies, defined as soft policies for which each action has at least the probability $frac\epsilon|\mathcal{U}|$ of being selected:

$$\pi\left(u^{(k)}|x^{(k)}\right) \geq \frac{\epsilon}{|\mathcal{U}|} \quad \forall x^{(k)} \in \mathcal{X},\, u^{(k)} \in \mathcal{U}$$

The **ϵ-greedy** policy is a special case of ϵ-soft policies that are characterized by the ability to select the greediest actions in most cases, but with probability ϵ can also select the random actions.

ϵ-greedy policy

$$\pi\left(u^{(k)}|x^{(k)}\right) = \begin{cases} \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} \neq \arg\max\limits_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right) \\ 1-\epsilon+\frac{\epsilon}{|\mathcal{U}|} & u^{(k)} = \arg\max\limits_{u^{(k)}} Q\left(x^{(k)}, u^{(k)}\right) \end{cases}$$

Monte Carlo with ϵ-greedy policy

Initialization. Select a guess $\pi_i = \pi_0$, $Q_i = Q_0$. Initialize an empty list Return $(\cdot, \cdot)$

Policy evaluation (PE). Determine the value of the current policy as in MC exploring starts

Policy improvement (PI). Determine an improved policy

$$u^{k^*} = \arg\max_{\pi} Q_{\pi_i}$$

$\forall\, u^{(k)} \in \mathcal{U}$

$$\pi\left(u^{(k)} | x^{(k)}\right) = \begin{cases} \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} \neq u^{k^*} \\ 1 - \epsilon + \frac{\epsilon}{|\mathcal{U}|} & u^{(k)} = u^{k^*} \end{cases}$$

PE and PI are repeated forever.

Incremental Monte Carlo evaluation

We observed that the MC method estimates the action-value function of a policy based on sample averages of observed rewards.

How these averages can be computed in a computationally efficient manner?

Notice that, if we denote by N the cardinality of Return $\left(x^{(k)}, u^{(k)}\right)$ we get:

$$
\begin{aligned}
Q_{N+1}\left(x^{(k)}, u^{(k)}\right) &= \frac{1}{N} \sum_{l=1}^{N} \text{Return}\left(x^{(k)}, u^{(k)}\right)_l \\
&= \frac{1}{N} \left(\text{Return}\left(x^{(k)}, u^{(k)}\right)_N + \sum_{l=1}^{N-1} \text{Return}\left(x^{(k)}, u^{(k)}\right)_l \right)
\end{aligned}
$$

Incremental Monte Carlo evaluation

We observed that the MC method estimates the action-value function of a policy based on sample averages of observed rewards.
How these averages can be computed in a computationally efficient manner?
Notice that, if we denote by N the cardinality of Return $\left(x^{(k)}, u^{(k)}\right)$ we get:

$$
\begin{aligned}
Q_{N+1}\left(x^{(k)}, u^{(k)}\right) &= \frac{1}{N}\sum_{l=1}^{N} \text{Return}\left(x^{(k)}, u^{(k)}\right)_l \\
&= \frac{1}{N}\left(\text{Return}\left(x^{(k)}, u^{(k)}\right)_N + \frac{N-1}{N-1}\sum_{l=1}^{N-1} \text{Return}\left(x^{(k)}, u^{(k)}\right)_l\right) \\
&= \frac{1}{N}\left(\text{Return}\left(x^{(k)}, u^{(k)}\right)_N + (N-1)Q_N\left(x^{(k)}, u^{(k)}\right)\right) \\
&= Q_N + \frac{1}{N}\left[\text{Return}\left(x^{(k)}, u^{(k)}\right)_N - Q_N\right]
\end{aligned}
$$

Incremental Monte Carlo evaluation

In the case of non-stationary systems, it makes sense to track a running mean to forget old (and less relevant) episodes.
It can be obtained by using a stepsize $\alpha \in (0, 1]$:

$$Q_{N+1} = Q_N + \alpha \left[\text{Return} \left(x^{(k)}, u^{(k)} \right)_N - Q_N \right]$$

Incremental Monte Carlo evaluation

In the case of non-stationary systems, it makes sense to track a running mean to forget old (and less relevant) episodes.
It can be obtained by using a stepsize $\alpha \in (0, 1]$:

$$Q_{N+1} = Q_N + \alpha \left[\text{Return}\left(x^{(k)}, u^{(k)}\right)_N - Q_N \right]$$

Convergence is guaranteed if:

$$\sum \alpha = \infty \qquad \sum \alpha^2 < \infty \qquad \rightarrow$$ **Robins and Monroe conditions**[1]

[1] Robbins, H., & Monro, S. (1951). A stochastic approximation method. The annals of mathematical statistics, 400-407.

Monte Carlo vs Dynamic Programming

There are three main advantages in using Monte Carlo methods instead of Dynamic Programming

- Can learn directly from interaction with the system
- No need for full model knowledge
- No need to learn about ALL states

Monte Carlo vs Dynamic Programming

There are three main advantages in using Monte Carlo methods instead of Dynamic Programming

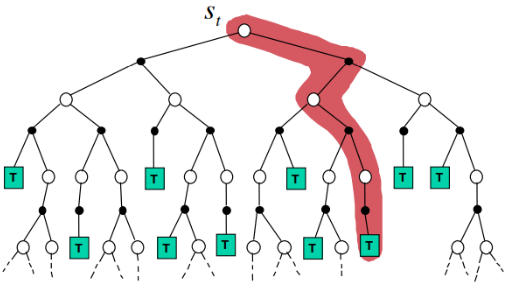
- Can learn directly from interaction with the system
- No need for full model knowledge
- No need to learn about ALL states

However:

- Monte Carlo must perform a full episode before performing estimate updates
- Dynamic Programming only needs to wait for the next time step to determine the increment of the estimate

Monte Carlo vs Dynamic Programming

Monte Carlo updates



Dynamic Programming updates

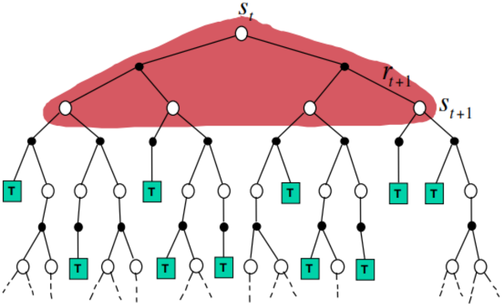


Table of Contents

5 Temporal difference learning

- A brief recap
- Machine-Learning
- Introduction to Reinforcement Learning
- Monte Carlo Learning
- Temporal difference learning

Temporal-difference

Temporal-difference is a combination of Monte Carlo and Dynamic Programming ideas:

- it can learn from data and does not need model dynamics
- it does not need to run an entire episode to update the estimate

Temporal-difference is a combination of Monte Carlo and Dynamic Programming ideas:

- it can learn from data and does not need model dynamics
- it does not need to run an entire episode to update the estimate

Consider the Monte Carlo updates for non-stationary systems:

$$Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[G_k - Q\left(x^{(k)}, u^{(k)}\right)\right]$$

It can also be written in value function form:

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[G_k - V\left(x^{(k)}\right)\right]$$

Considering the definition of G_k we can write the following:

$$
\begin{aligned}
V\left(x^{(k)}\right) &= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma\sum_{k=0}^{\infty}\gamma^k r^{(k+2)} - V\left(x^{(k)}\right)\right] \\
&= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]
\end{aligned}
$$

Considering the definition of G_k we can write the following:

$$
\begin{aligned}
V\left(x^{(k)}\right) &= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma \sum_{k=0}^{\infty} \gamma^k r^{(k+2)} - V\left(x^{(k)}\right)\right] \\
&= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]
\end{aligned}
$$

temporal-difference target

Considering the definition of G_k we can write the following:

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma \sum_{k=0}^{\infty} \gamma^k r^{(k+2)} - V\left(x^{(k)}\right)\right]$$

$$= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]$$

temporal-difference error

- The temporal-difference error (**TD error**) δ_k measures the difference between the estimated value of $x^{(k)}$ and the better estimate

Considering the definition of G_k we can write the following:

$$
\begin{aligned}
V\left(x^{(k)}\right) &= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma\sum_{k=0}^{\infty}\gamma^k r^{(k+2)} - V\left(x^{(k)}\right)\right] \\
&= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]
\end{aligned}
$$

temporal-difference error

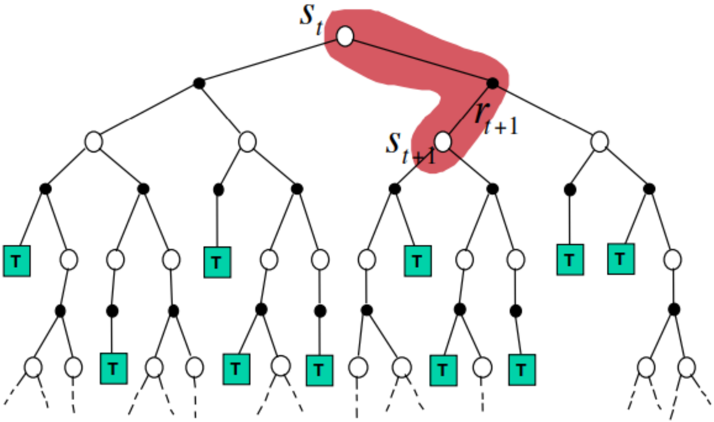
- The temporal-difference error (**TD error**) δ_k measures the difference between the estimated value of $x^{(k)}$ and the better estimate
- The update of the estimate is obtained after performing one step of the episode

Considering the definition of G_k we can write the following:

$$
\begin{aligned}
V\left(x^{(k)}\right) &= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma \sum_{k=0}^{\infty} \gamma^k r^{(k+2)} - V\left(x^{(k)}\right)\right] \\
&= V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]
\end{aligned}
$$

- The temporal-difference error (**TD error**) δ_k measures the difference between the estimated value of $x^{(k)}$ and the better estimate
- The update of the estimate is obtained after performing one step of the episode
- Given the presence of an existing estimate in the updating rule it is defined **bootstrapping** method

Temporal-difference



Temporal-difference advantages

TD methods do not require a model of the environment, only experience, i.e., we can learn without knowing the final outcome

- Less memory
- Less computation
- Incomplete sequence

Therefore, it is only defined for **episodic tasks**

Temporal-difference for control

Again, to apply TD-learning strategy for control purposes, we need to develop a policy iteration strategy that consists of the following interactive processes:

- Policy evaluation
- Policy iteration

Temporal-difference for control

Again, to apply TD-learning strategy for control purposes, we need to develop a policy iteration strategy that consists of the following interactive processes:

- Policy evaluation
- Policy iteration

In Monte Carlo method we already observe that in order to ensure a proper exploration/exploitation trade-off we need to apply non an only greedy policy, but an ϵ-greedy policy:

- Select greedy action with probability $1 - \epsilon$
- Select random action with probability ϵ

Initialization. $Q\ \forall x \in \mathcal{X}\ u \in \mathcal{U}$

Repeat (for each episode)

- Set $x^{(0)}$
- Select an action $u^{(k)}$ with ϵ-greedy policy
- Repeat for each step of the episode until terminal condition is met
  - Perform the action $u^{(k)}$, observe $x^{(k+1)}$ and $r^{(k+1)}$
  - Select an action $u^{(k+1)}$ with ϵ-greedy policy
  - $Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma Q\left(x^{(k+1)}, u^{(k+1)}\right) - Q\left(x^{(k)}, u^{(k)}\right)\right]$
  - Update $x^{(k)} = x^{(k+1)}$, $u^{(k)} = u^{(k+1)}$

SARSA algorithm

Initialization. $Q \, \forall x \in \mathcal{X} \; u \in \mathcal{U}$

Repeat (for each episode)

- Set $x^{(0)}$
- Select an action $u^{(k)}$ with ϵ-greedy policy
- Repeat for each step of the episode until terminal condition is met
  - Perform the action $u^{(k)}$, observe $x^{(k+1)}$ and $r^{(k+1)}$
  - Select an action $u^{(k+1)}$ with ϵ-greedy policy
  - $Q\left(x^{(k)}, u^{(k)}\right) = Q\left(x^{(k)}, u^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma Q\left(x^{(k+1)}, u^{(k+1)}\right) - Q\left(x^{(k)}, u^{(k)}\right)\right]$
  - Update $x^{(k)} = x^{(k+1)}$, $u^{(k)} = u^{(k+1)}$

Notice that at each step the ϵ-greedy policy is used we first have updated $Q \rightarrow$ **Policy improvement**

Convergence of Sarsa

Sarsa converges to the optimal action-value function, $Q(x, u) \to Q^*(x, u)$, under the following conditions:

- All state-action pairs are explored infinitely many times

$$\lim_{k \to \infty} N_k(x, u) = \infty$$

- The policy converges on a greedy policy. It can be reached by imposing a decaying ϵ
- Robins and Monroe conditions are satisfied

We can notice that in the Monte Carlo update the term:

$$G_k - V\left(x^{(k)}\right)$$

can be written as follows:

$$\begin{aligned}
& r^{(k+1)} + \gamma G_{k+1} - V\left(x^{(k)}\right) + \gamma V\left(x^{(k+1)}\right) - \gamma V\left(x^{(k+1)}\right) \\
=& \delta_k + \gamma\left(G_{k+1} - V\left(x^{(k+1)}\right)\right) \\
=& \delta_k + \gamma\delta_{k+1} + \gamma^2\left(G_{k+2} - V\left(x^{(k+2)}\right)\right) \\
=& \delta_k + \gamma\delta_{k+1} + \gamma^2\left(G_{k+2} - V\left(x^{(k+2)}\right)\right) = \sum_{i=0}^{T-k-1} \gamma^i \delta_{k+i}
\end{aligned}$$

n-step Temporal-difference

- One step TD learning
  - target TD:

$$G_{k:k+1} = r^{(k+1)} + \gamma V\left(x^{(k+1)}\right)$$

  - bellman equation:

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]$$

- One step TD learning
  - target TD:

$$G_{k:k+1} = r^{(k+1)} + \gamma V\left(x^{(k+1)}\right)$$

  - bellman equation:

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[r^{(k+1)} + \gamma V\left(x^{(k+1)}\right) - V\left(x^{(k)}\right)\right]$$

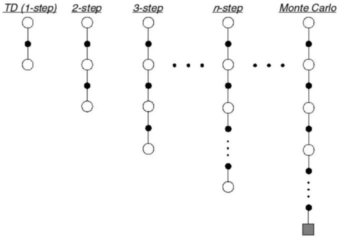
- Monte Carlo:
  - target:

$$G_k = r^{(k+1)} + \gamma r^{(k+2)} + \ldots + \gamma^{T-k-1} r^T$$

  - bellman equation:

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[G_k - V\left(x^{(k)}\right)\right]$$

n-step Temporal-difference



- n-step TD learning
  - target TD:

$$G_{k:k+n} = r^{(k+1)} + \gamma r^{(k+2)} + \gamma^{n-1} r^{(k+n)} + \gamma^n V\left(x^{(k+n)}\right)$$

  - bellman equation:

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\left[G_{k:k+n} + \gamma V\left(x^{(k)}\right) - V\left(x^{(k)}\right)\right]$$

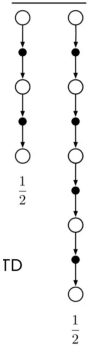
Temporal-difference(λ)

A simple way to combine n-step TD returns is to average n-step return as òong as the weights on component returns are positive and sum to 1:

$$G_{k:k+2} = r^{(k+1)} + \gamma r^{(k+2)} + \gamma^2 V\left(x^{(k+2)}\right)$$

$$G_{k:k+4} = r^{(k+1)} + \gamma r^{(k+2)} + \gamma^2 r^{(k+3)} + \gamma^3 r^{(k+4)} + \gamma^4 V\left(x^{(k+4)}\right)$$

$$G_{\text{avg}} = \frac{1}{2} G_{k:k+2} + \frac{1}{2} G_{k:k+4}$$



Temporal-difference(λ)

A simple way to combine n-step TD returns is to average n-step return as òong as the weights on component returns are positive and sum to 1:

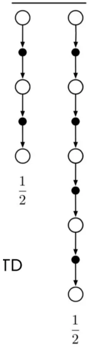
$$G_{k:k+2} = r^{(k+1)} + \gamma r^{(k+2)} + \gamma^2 V\left(x^{(k+2)}\right)$$

$$G_{k:k+4} = r^{(k+1)} + \gamma r^{(k+2)} + \gamma^2 r^{(k+3)} + \gamma^3 r^{(k+4)} + \gamma^4 V\left(x^{(k+4)}\right)$$

$$G_{\text{avg}} = \frac{1}{2} G_{k:k+2} + \frac{1}{2} G_{k:k+4}$$

$$\downarrow$$

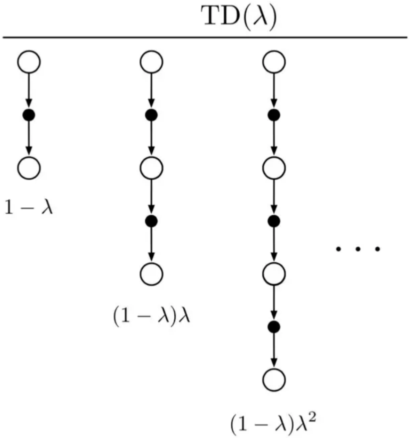
compound return



TD(λ) is one particular way of averaging n-steps updates.

This average contains all the n-steps updates, each weighted proportional to λ^{n-1}

Besides, each term of n-step return are normalized by a factor of $1 - \lambda$ to ensure that the weights sum to 1.



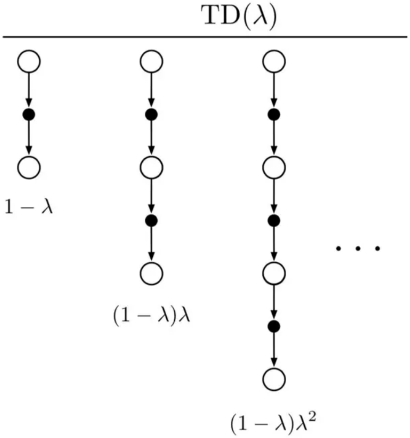
The return is defined as follows

$$G_k^\lambda = (1-\lambda)\sum_{n=1}^{T-k-1} \lambda^{n-1} G_{k:k+n} + \lambda^{T-k-1} G_k$$

Where:

- $\lambda = 0$ is TD(0) (one-step TD)
- $\lambda = 1$ is Monte Carlo

Besides, each term of n-step return are normalized by a factor of $1 - \lambda$ to ensure that the weights sum to 1.

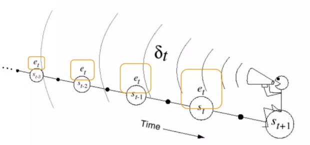


Eligibility trace

Is a variable that keeps track of the weights of the updating value function for every state

$$e_0(x^{(k)}) = 0$$

$$e_k(x^{(k)}) = \gamma\lambda e_{k-1}(x^{(k)}) + 1$$



Conceptually it is like the strength of your voice that decreases with temporal distance by $\gamma\lambda$

Backward TD(λ)

The algorithm for solving TD(λ) is:

1. Keep an eligibility trace for every state $x^{(k)}$
2. Update values $V(x^{(k)})$ for every state $x^{(k)}$ in the single step in proportion to TD-error δ_k and eligibility trace

$$e_k(x^{(k)}) = 0$$

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\delta_k e_k(x^{(k)})$$

Notice that:

- When $\lambda = 0$ only the current state is updated

$$e_k(x^{(k)}) = \gamma\lambda e_{k-1}(x^{(k)}) + 1 = 1$$

$$V\left(x^{(k)}\right) = V\left(x^{(k)}\right) + \alpha\delta_k$$

We are performing TD(0).

Questions' time!

UNIVERSITÀ DEGLI STUDI DI TRIESTE