Reinforcement Learning

Introduction: Framework, concepts and definitions

Mario Martin

CS-UPC

March 18, 2018

(*)Some parts of this slides are taken from David Silver’s UCL Course and Sutton’s supplementary material for his book
What is reinforcement learning?: RL Framework
Main characteristics of RL:

1. Goal is learning a behavior (*policy*), not a class
2. Grounded *agent-like* learning:
   - Agent is *active* in the environment
   - Learning is continuous
Reinforcement Learning concept

Main characteristics of RL:

1. Goal is learning a behavior (*policy*), not a class
2. Grounded *agent-like* learning:
   - Agent is *active* in the environment
   - Learning is continuous

**Informal definition**

Learning about, from, and while interacting with an environment to achieve a goal (learning a behavior).
RL Framework

State → Agent → World → Action
RL Framework

State → Agent → World → Action

Reward
RL Framework

Agent

State: $s_{t+1}$

Reward: $r_{t+1}$

Action: $a_t$

World
Why use reward instead of examples?:

1. Usually it’s easy to define a reward function.
2. You don’t need to know the goal behavior to train an agent.
3. Behavior is grounded an efficient (optimal in some cases) given the perceptions and actions of the agent.
Why use reward instead of examples?:

1. Usually it’s easy to define a reward function.
2. You don’t need to know the goal behavior to train an agent.
3. Behavior is grounded an efficient (optimal in some cases) given the perceptions and actions of the agent.

**Reward assumption**

All goals can be formalized as the outcome of maximizing a cumulative reward.
What makes reinforcement learning harder than other machine learning paradigms?

- Feedback is not the *right action*, but a *sparse* scalar value (*reward function*).
- Relevant feedback is delayed, not instantaneous.
- Time really matters (sequential, non i.i.d. data).
- Environment can be stochastic and uncertain.
Informal definition

Learning about, from, and while interacting with an environment to achieve a goal (learning a behavior).
**Informal definition**

Learning about, from, and while interacting with an environment to achieve a goal (learning a behavior).

... read as ... 

**Formal definition**

Learning a mapping from situations to actions to maximize long-term reward, without using a model of the world.
Agent and environment interact at discrete time steps: \( t = 0, 1, 2, \ldots \)

- Agent observes state at step \( t \): \( s_t \in S \)
- produces action at step \( t \): \( a_t \in A(s_t) \)
- gets resulting reward: \( r_{t+1} \in \mathbb{R} \)
- and resulting next state: \( s_{t+1} \)
RL Framework

Snapshot of a trial of the agent:

\[ \ldots \rightarrow s_t \xrightarrow{a_t} r_{t+1} \xrightarrow{s_{t+1}} a_{t+1} \xrightarrow{r_{t+2}} s_{t+2} \xrightarrow{a_{t+2}} r_{t+3} \rightarrow s_{t+3} \xrightarrow{a_{t+3}} \ldots \]
**RL Framework: MDP process**

- RL Problem can be formulated as a Markov Decision Process (MDP): a tuple $< S, A, P, R >$ where
  - $S$: Finite set of states
  - $A$: Finite set of actions
  - $P$: Transition Probabilities *(Markov property):*
    \[ P^a_{ss'} = \Pr \{ s_{t+1} = s' \mid s_t = s, a_t = a \} \forall s, s' \in S, a \in A(s). \]
  - $R$: Reward Probabilities:
    \[ R^a_{ss'} = \mathbb{E} \{ r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s' \} \forall s, s' \in S, a \in A(s). \]
RL Framework: MDP process

- RL Problem can be formulated as a Markov Decision Process (MDP): a tuple $< S, A, P, R >$ where
  - $S$: Finite set of states
  - $A$: Finite set of actions
  - $P$: Transition Probabilities (Markov property):
    $$P_{ss'}^a = \Pr \{ s_{t+1} = s' \mid s_t = s, a_t = a \} \forall s, s' \in S, a \in A(s).$$
  - $R$: Reward Probabilities:
    $$R_{ss'}^a = \mathbb{E} \{ r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s' \} \forall s, s' \in S, a \in A(s).$$

- Some constraints can be relaxed later:
  - Markov property (fully vs. partial observability)
Definition of markovian environment:

**Markov property**

An environment is Markovian if and only if for each state $S_t$

$$P(S_{t+1}|S_t) = P(S_{t+1}|S_1 \ldots , S_{t-1}, S_t)$$

Ways to say the same:
- The future is independent of the past given the present
- Once the state is known, the history may be thrown away
- The state is a sufficient statistic of the future
Not all problems are markovian.

- A robot with a camera isn’t told its absolute location
- A trading agent only observes current prices
- A poker playing agent only observes public cards

This could lead to *perceptual aliasing*: confusing different states with the same perception.

In the following example, states 1 and 3 (f.i.) are aliased if sensors of the agent only report information about the clear/not-clear of neighbor cells.
Several ways to solve this problem:

1. The agent builds a belief about the current state from past observations and actions (POMDP approach).

2. Use memory to disambiguate states: Use last $H$ perceptions to represent current state: $S_t = \langle P_1, \ldots P_t \rangle$

3. Learn to build a representation of the state using history of the agent: (f.i. Recurrent networks, LSTMs)
RL Framework: MDP process

- RL Problem can be formulated as a Markov Decision Process (MDP): a tuple $< S, A, P, R >$ where
  - $S$: Finite set of states
  - $A$: Finite set of actions
  - $P$: Transition Probabilities (Markov property):
    \[
    P^a_{ss'} = \Pr \{ s_{t+1} = s' \mid s_t = s, a_t = a \} \forall s, s' \in S, a \in A(s).
    \]
  - $R$: Reward Probabilities:
    \[
    R^a_{ss'} = \mathbb{E} \{ r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s' \} \forall s, s' \in S, a \in A(s).
    \]
- Some constraints can be relaxed later:
  - Markov property (fully vs. partial observability)
  - Infinite (or continuous) sets of actions and states
RL elements:
In RL are **key** the following elements:

1. **Policy**: What to do.
2. **Model**: What follows what. Dynamics of the environment.
3. **Reward**: What is good
4. **Value function**: What is good because it predicts reward.
A **policy** is the agent’s behavior.

It is a map from current state to action to execute:

\[ \pi : s \in \mathcal{S} \longrightarrow a \in \mathcal{A} \]

*Policy* could be deterministic:

\[ a = \pi(s) \]

... or stochastic:

\[ \pi(a|s) = P[A_t = s|S_t = s] \]
A **model** predicts next state and reward

- Allows modeling of stochastic environments with probability transition functions:
  - \( P \) predicts the next state
    - \( P(s_{t+1} = s' | S_t = s, A_t = a) \)

- **Usually not known by the agent**
Rewards

- **Immediate reward** $r_t$ is a *scalar* feedback value that depends on the current state and action taken in that state.

- **Reward function** $R$ determines (immediate) reward $r_t$ at each step of the agent’s life.

- It is very sparse and does not evaluate of the goodness of the last action but the goodness of the whole chain of actions (*trajectory*).

- It can be stochastic:

$$R_s^a = \mathbb{E}[r_{t+1}|S_t = s, A_t = a]$$

- Notice that reward depends on next state $s'$, so we can assume that is a function of only the state $r(s')$. 
Rewards: examples

- Fly stunt manoeuvres in a helicopter
  - +ve reward for following desired trajectory
  - -ve reward for crashing
- Defeat the world champion at Go
  - +ve/-ve reward for winning/losing a game
- Make a humanoid robot walk
  - +ve reward for forward motion
  - -ve reward for falling over
- Play Atari games better than humans
  - +ve reward for increasing/decreasing score
Agents will learn from experiences that in this case are sequences of actions.

Interaction of the agent with the environment can be organized in two different ways:

- **Trials (or episodic learning):** The agent has a final state after which he receives the reward. In some cases, it must be achieved within a limited maximum time $H$. After he arrives at the goal state (or surpasses the maximum time allowed), a new trial is started.

- **Non-ending tasks:** The agent has no limit in time or it has not a clear final state. Learning by trials can be simulated by adding random extra-transitions from goal states to initial states.
The agent’s job is to maximise cumulative reward over an episode

Long term reward must be defined in terms of the goal of the agent

Definition of long-term reward must be derived from local rewards
Long-term Return

First intuitive definition of long-term reward:

Infinite horizon undiscounted return

\[ R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots = \sum_{k=0}^{\infty} r_{t+k+1} \]

Problem: Long-term reward should have a limit.
First intuitive definition of long-term reward:

**Infinite horizon undiscounted return**

\[ R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots = \sum_{k=0}^{\infty} r_{t+k+1} \]

Problem: Long-term reward should have a limit.

**Finite horizon undiscounted return**

\[ R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_H = \sum_{k=0}^{H} r_{t+k+1} \]

Problem: Optimal policy depends on horizon \( H \) and becomes no-stationary.
Long-term Return

With $H = 3$, $\pi(S_1)$ is different if you look at the problem from $S_0$ or $S_1$. 

---

Mario Martin (CS-UPC)  
Reinforcement Learning  
March 18, 2018 25 / 59
Long-term Return

Infinite horizon discounted return

The return $R_t$ is the total discounted reward from time-step $t$.

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- The discount $\gamma \in [0, 1]$ is the present value of future rewards. Usually very close to 1.
- The value of receiving reward $r$ after $k + 1$ time-steps is $\gamma^k r$.
- This values immediate reward above delayed reward: $\gamma$ close to 0 leads to *myopic* evaluation $\gamma$ close to 1 leads to *far-sighted* evaluation.
Long-term Return

\[ \gamma = 0.7 \]

\[ S_1 \xrightarrow{a=A_1, r=5} S_2 \]
\[ S_1 \xrightarrow{a=A_2, r=0} S_3 \]
\[ S_3 \xrightarrow{a=A_1, A_2, r=0} S_4 \]
\[ S_4 \xrightarrow{a=A_1, A_2, r=10} S_5 \]
Long-term Return

- Infinite horizon *discounted* return is limited by:

\[
R_t \leq \sum_{k=0}^{\infty} \gamma^k r_{max} = \frac{r_{max}}{1 - \gamma}
\]

- So, also useful for learning non-ending tasks, because addition is unlimited.
- Greedy policies are stationary
- Elegant and convenient definition recursive definition (see Bellman eqs. later)
Long-term Return

- Infinite horizon discounted return is limited by:

\[ R_t \leq \sum_{k=0}^{\infty} \gamma^k r_{\text{max}} = \frac{r_{\text{max}}}{1 - \gamma} \]

- So, also useful for learning non-ending tasks, because addition is unlimited.
- Greedy policies are stationary
- Elegant and convenient definition recursive definition (see Bellman eqs. later)
- Maximization of Long-term Return should lead to optimal behavior.
Long-term Return examples

- Pole balancing example:

  - Episodic learning.
  - Three possible actions: \(-F, 0, F\)
  - State is defined by \((x, \dot{x}, \theta, \dot{\theta})\)
  - Markovian problem because \((x', \dot{x}', \theta', \dot{\theta}') = F(x, \dot{x}, \theta, \dot{\theta})\)
  - Goal: \(|\theta|\) below a threshold (similar to a Segway problem)
Reward definition:

**Case 1:** $\gamma = 1, r = 1$ for each step except $r = 0$ when pole falls.

$\implies R =$ number of time steps before failure

- Return is maximized by avoiding failure for as long as possible.
Long-term Return examples

Reward definition:

Case 2: $\gamma < 1$, $r = 0$ for each step, and $r = -1$ when pole falls.

$\Rightarrow R = -\gamma^k$ for $k$ time steps before failure

- Return is maximized by avoiding failure for as long as possible.
Long-term Return examples

Other examples:

- **Classic control**
  Control theory problems from the classic RL literature.

- **Acrobot-v1**
  Swing up a two-link robot.

- **CartPole-v1**
  Balance a pole on a cart.

- **MountainCarContinuous-v0**
  Drive up a big hill with continuous control.

- **MountainCar-v0**
  Drive up a big hill.

- **Pendulum-v0**
  Swing up a pendulum.
Recap

- Definition of RL
- Framework
- Concepts learned:
  - Model
  - Policy: deterministic and non-deterministic
  - Reward functions, immediate reward
  - Discounted and undiscounted Long-term reward
  - ... $\gamma$, $\pi$, markovian condition
Value functions
Value function

- Value function is a prediction of future reward
- Used to evaluate goodness/badness of states
- Depends on the agent’s policy...
- ... and is used to select between actions

**state-value function** $V^\pi(s)$

$V^\pi(s)$ is defined as the expected return starting from state $s$, and then following policy $\pi$

$$V^\pi(s) = \mathbb{E}_\pi[R_t|S_t = s] = \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s \right\}$$
**Q-Value function**

**action-value function** $Q^\pi(s, a)$

$Q^\pi(s, a)$ is the expected return starting from state $s$, taking action $a$, and then following policy $\pi$

$$Q^\pi(s, a) = \mathbb{E}_\pi[R_t | S_t = s, A_t = a] = \mathbb{E}_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a \right\}$$
Bellman expectation equation

The value function can be decomposed into two parts:

- immediate reward $r_{t+1}$
- discounted value of successor state $\gamma V^\pi(S_{t+1})$

$$V^\pi(s) = \mathbb{E}_\pi[R_t | S_t = s]$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | S_t = s]$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \ldots) | S_t = s]$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s]$$

$$= \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s]$$
So, the state-value function can again be decomposed recursively into immediate reward plus discounted value of successor state,

**Bellman equation for state-value function**

\[
V_\pi(s) = \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(S_{t+1}) | S_t = s]
\]

Equivalent expressions without the expectation operator:

\[
V_\pi(s) = \sum_{s'} P_{ss'}^{\pi(s)} \left[ R_{ss'}^{\pi(s)} + \gamma V_\pi(s') \right]
\]

\[
V_\pi(s) = \sum_{s'} P_{ss'}^{\pi(s)} R_{ss'}^{\pi(s)} + \sum_{s'} P_{ss'}^{\pi(s)} \gamma V_\pi(s')
\]

\[
V_\pi(s) = r(s, \pi(s)) + \sum_{s'} P_{ss'}^{\pi(s)} \gamma V_\pi(s')
\]
The action-value function can similarly be decomposed:

\[ Q^\pi(s, a) = \mathbb{E}_\pi[R_t | S_t = s, A_t = a] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots | S_t = s, A_t = a] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma (r_{t+2} + \gamma r_{t+3} + \ldots) | S_t = s, A_t = a] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma R_{t+1} | S_t = s, A_t = a] \]
\[ = \mathbb{E}_\pi[r_{t+1} + \gamma V^\pi(S_{t+1}) | S_t = s, A_t = a] \]
Bellman Expectation Equation for $Q^\pi$

Notice that:

$$V^\pi(S_t) = Q^\pi(S_t, \pi(S_t))$$
Bellman Expectation Equation for $Q^\pi$

Notice that:

$$V^\pi(S_t) = Q^\pi(S_t, \pi(S_t))$$

So,

**Bellman equation for state-action value function**

$$Q^\pi(s, a) = \mathbb{E}_{\pi}[r_{t+1} + \gamma Q^\pi(S_{t+1}, \pi(S_t)) | S_t = s, A_t = a]$$

Equivalent expressions without the expectation operator:

$$Q^\pi(s, a) = \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma V^\pi(s') \right]$$

$$Q^\pi(s, a) = \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma Q^\pi(s', \pi(s')) \right]$$

$$Q^\pi(s, a) = \sum_{s'} P^a_{ss'} R^a_{ss'} + \sum_{s'} P^a_{ss'} \gamma Q^\pi(s', \pi(s'))$$

$$Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} P^a_{ss'} Q^\pi(s', \pi(s'))$$
Maze example

- Rewards: -1 per time-step
- Actions: N, S, W, E
- States: Agent’s location
Arrows represent policy $\pi(s)$ for each state $s$. 

- Arrows represent policy $\pi(s)$ for each state $s$. 

Mario Martin  (CS-UPC)  Reinforcement Learning  March 18, 2018  42 / 59
Maze example: value function

- Numbers represent $V^\pi(s)$ for each state $s$, for $\gamma = 1$
- How much is $Q^\pi(\langle 2, 1 \rangle, \downarrow)$?
Given $\pi$, policy evaluation methods obtain $V^\pi$.

First method: Algebraic solution using Bellman equations in matrix form
Policy evaluation (1)

- Given $\pi$, **policy evaluation** methods obtain $V^\pi$.
- First method: Algebraic solution using Bellman equations in matrix form

\[
V^\pi(s) = \sum_{s'} P^{\pi(s)}_{ss'} \left[ R^{\pi(s)}_{ss'} + \gamma V^\pi(s') \right]
\]

\[
V^\pi = P^\pi [R + \gamma V^\pi]
\]

\[
V^\pi = P^\pi R + P^\pi \gamma V^\pi
\]

\[
V^\pi - P^\pi \gamma V^\pi = P^\pi R
\]

\[
(I - \gamma P^\pi) V^\pi = P^\pi R
\]

\[
V^\pi = (I - \gamma P^\pi)^{-1} P^\pi R
\]

**Algebraic solution**

\[
V^\pi = (I - \gamma P^\pi)^{-1} P^\pi R
\]
Second method: iterative policy evaluation

Given arbitrary \( V(s) \) as estimation of \( V^{\pi} \), we can tell the error using Bellman equations

\[
\text{error} = \max_{s \in S} \left| V(s) - \sum_{s'} P_{ss'}^{\pi} \left[ R_{ss'}^{\pi} + \gamma V(s') \right] \right|
\]

Consider to apply iteratively Bellman equations to update \( V \) for all states (Bellman operator)

\[
V(s) \leftarrow \sum_{s'} P_{ss'}^{\pi} \left[ R_{ss'}^{\pi} + \gamma V(s') \right]
\]

Solution is a fixed point of the application of this operator

Apply updates of Bellman operator until convergence.

Convergence can be proved: at each iteration error is reduced by a \( \gamma \) factor.
Iterative policy evaluation

Given $\pi$, the policy to be evaluated, initialize $V(s) = 0 \ \forall s \in S$

repeat

$\Delta \leftarrow 0$

for each $s \in S$ do

$v \leftarrow V(s)$

$V(s) \leftarrow \sum_{s'} P_{ss'}^\pi(s) \left[ R_{ss'}^\pi(s) + \gamma V(s') \right]$  

$\Delta \leftarrow \max(\Delta, |v - V(S)|)$

end for

until $\Delta < \theta$ (a small threshold)
Optimal Policies
We can define a partial ordering of policies “≤” in the following way:

\[ \pi' \leq \pi \iff V^{\pi'}(s) \leq V^{\pi}(s) \quad \forall s \]

Under this ordering, we can prove that:
- There exists at least an optimal policy (\(\pi^*\))
- Could be not unique
- In the set of optimal policies some are deterministic
- All share the same value function
We say a policy $\pi$ is **greedy** when:

$$\pi(S_t) = \arg \max_{a \in A} \mathbb{E}_\pi[R_{t+1}]$$

if value states are estimations of $R_t$, then in greedy policies:

$$\pi(s) = \arg \max_{a \in A} \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma V^\pi(s') \right]$$

$$V^\pi(s) = \max_{a \in A} \sum_{s'} P^a_{ss'} \left[ R^a_{ss'} + \gamma V^\pi(s') \right]$$

It is easy to see that **the optimal policy is greedy**.
Infinite number of actions (f.i. continuous space of actions)

Implementations of:

\[ \pi(S_t) = \arg \max_{a \in A} \mathbb{E}_\pi[R_{t+1}] \]

is easy when we have a finite number of actions. When we have an infinite number of actions like in case of continuous space of actions (parametrized actions), computation is harder!
Finding Policies: Model based methods
Finding policies

Knowing that optimal policy is greedy...

... and using recursive Bellman equations

we can apply *Dynamic Programming* (DP) techniques to find the optimal policies

Main methods to find optimal policies using DP

- Policy iteration (PI)
- Value iteration (VI)
Finding policies

- Knowing that optimal policy is greedy...
- ... and using recursive Bellman equations
- we can apply *Dynamic Programming* (DP) techniques to find the optimal policies
- Main methods to find optimal policies using DP
  - Policy iteration (PI)
  - Value iteration (VI)
- *Model based method*: In these methods, knowledge of the model is assumed.
A policy $\pi$ can be improved iif

$$\exists \, s \in S, a \in A \text{ such that } Q^\pi(s, a) > Q^\pi(s, \pi(s))$$

Obvious. In this case, $\pi$ is not optimal and can be improved setting $\pi(s) = a$

Simple idea for the algorithm:

1. Start from random policy $\pi$
2. Compute $V^\pi$
3. Check for each state if the policy can be improved (and improve it)
4. If policy cannot be improved, stop. In other case repeat from 2.
Finding policies: Policy iteration

Policy Iteration (PI)

Initialize $\pi, \forall s \in S$ to a random action $a \in A(s)$, arbitrarily
repeat
  $\pi' \leftarrow \pi$
  Compute $V^\pi$ for all states using a policy evaluation method
  for each state $s$ do
    $\pi(s) \leftarrow \arg\max_a \sum_{s'} P_{ss'}^a \left[ R_{ss'}^a + \gamma V^\pi(s') \right]$
  end for
until $\pi(s) = \pi'(s) \ \forall s$
Finding policies: Value iteration

- Problem with Policy iteration: policy evaluation inside the main loop
- Policy evaluation takes a lot of time. Has to be done before improving the policy
- We can stop policy evaluation before convergence
- In the extreme case, we can stop policy evaluation after a single sweep (one update of each state).
- This algorithm is called Value Iteration and can be proved to converge to the optimal policy
- It combines in one step improvement of the policy and computation of $V$
Finding policies: Value iteration

Value Iteration (VI)

Initialize $V(s) \forall s \in S$ arbitrarily (for instance to 0)
repeat
  $\Delta \leftarrow 0$
  for each $s \in S$ do
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |v - V(S)|)$
  end for
until $\Delta < \theta$ (a small threshold)

Return deterministic policy, $\pi$, such that:
$\pi(s) = \arg \max_a \sum_{s'} P_{ss'}^a [R_{ss'}^a + \gamma V^\pi(s')]$
Finding policies: Asynchronous versions

- All the DP methods described so far require exhaustive sweeps of the entire state set.
- Asynchronous DP does not use complete sweeps.
- Pick a state at random and apply the appropriate backup. Repeat until convergence criterion is met:
  - Still need lots of computation, but does not get locked into hopelessly long sweeps
- Can you select states to backup intelligently? YES: an agent’s experience can act as a guide.
Lab session

- Install software. For this lab you only need Python 3.x installed, numpy, matplotlib and Jupyter.
- Go to web page of the course and download notebooks for:
  1. Policy evaluation
  2. Policy iteration and Value iteration