Introduction to Reinforcement Learning
1/10

Jean Martinet

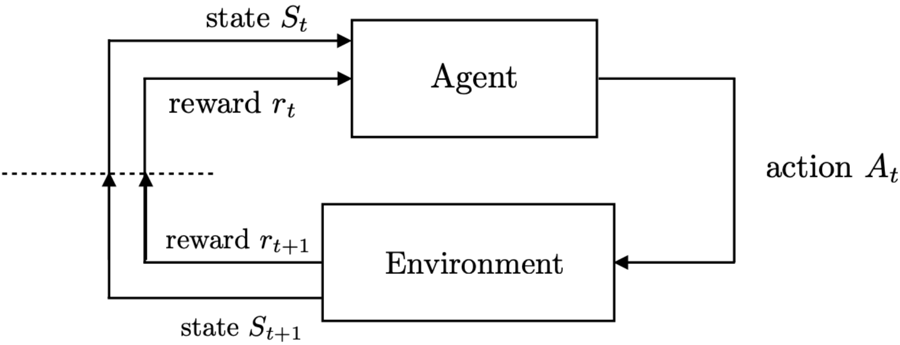
MSc DSAI

2024 – 2025

Three main learning paradigms

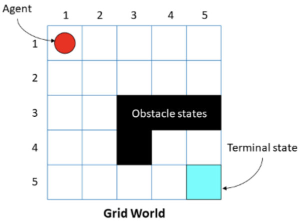
- **Supervised learning**
  - Learn a mapping between inputs and outputs
  - An oracle provides labelled examples of this mapping
- **Unsupervised learning**
  - Learn a structure in a data set (capture the distribution)
  - No oracle
- **Reinforcement Learning**
  - Learn to behave
  - Online learning
  - Sequential decision making under uncertainty, control

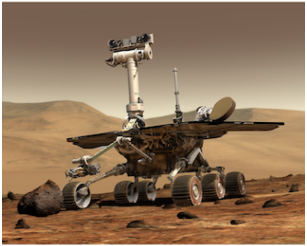
General problem



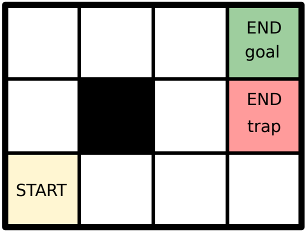
Examples

- **Artificial problems**
  - Mazes, grid worlds
  - Mountain car
  - Inverted Pendulum
  - Games : Backgammon, Chess, Atari, Go
- **Real world problems**
  - Man-Machine Interfaces
  - Data center cooling
  - Autonomous robotics





A classic toy example



- State : position
- Transition rules : 4 directions UP, DOWN, LEFT, RIGHT

- Example of stochastic environment
  - probability of move success : 0.8
  - probability of failure, end up in lateral position : 0.1 for each
  - example : chosen action = UP
    - probability 0.8 to go UP
    - probability 0.1 to go LEFT
    - probability 0.1 to go RIGHT
  - external bouncing walls
- Reward
  - +1 if end goal
  - -1 if end trap
  - -0.01 move cost in any case

About this class : contents

- Objectives
  - Understand the key concepts of RL, distinguish from other AI / ML
  - Know if a problem can be formulated as a RL problem and how
  - Implement standard RL algorithms
- Ten sessions (lectures + labs)
- Link to other courses : AI, ML, DL, etc.
- Prerequisites : Python proficiency, basics in probability and statistics

About this class : programme

- Introduction
  - Course 1 : Introduction to Reinforcement Learning (RL)
    - Motivations, key concepts, difference with other paradigms
    - Lab : install environment and libs, play with TicTacToe
- Part I on tabular methods
  - Course 2 : Markov Decision Processes
  - Course 3 : Dynamic programming in RL
  - Course 4 : Temporal difference 1/2 (Q-learning)
  - Course 5 : Temporal difference 2/2 (SARSA)
- Part II on approximate methods
  - Course 6 : Value function approximation
  - Course 7 : Eligibility traces
  - Course 8 : Policy gradient 1/2 (REINFORCE)
  - Course 9 : Policy gradient 2/2 (actor-critic methods)
  - Course 10 : Projects presentation session

About this class : evaluation

- Two assignments (first is individual, second is in group)
  - One graded lab, one open project (see next slide)
  - Respectively 20% – 30%
- IMPORTANT
  - Late submission policy = <24h=-10%, 24h-48h=-20%, >48h=0
- One final exam 50%

Open project

- **Choose from :**
  - Articles/advanced topics/applications
    - Conference paper or book chapter
    - Advanced theme (e.g. actor-critic, eligibility trace, etc.)
    - Application domain (e.g. temp. control, revenue management, etc.)
  - Deepening or exploration project
    - Subject to be chosen/defined and validated
- **Choice to be validated before session 4**

Four key concepts in RL

- **Policy**
  - The way of behaving at a given time
  - Mapping between perceived state of environment and action to take
- **Reward signal**
  - Defines the goal of a reinforcement learning problem
  - Number sent by the environment to the agent at each time step
  - Defines what are the good and bad events
  - Analogous to the experiences of pleasure or pain in biology
- **Value function**
  - Specifies what is good in the long run – different from the reward
  - Important and hard to estimate
- **Model (optional)**
  - Mimics the behaviour of the environment
  - Given a state and action, predicts next state and reward
  - Model-based (planning) vs. model-free methods (trial-and-error)

Sequential decision making

- At each time step t, agent in state $s_t \in S$ executes action $a_t \in A$
- As a consequence, the agent reaches a new state s_{t+1} and receives from the environment a reward r_{t+1}
  - feedback that measures the success or failure of an agent's action
- The total reward (return, also called utility) at time step t is

$$G_t \doteq r_{t+1} + r_{t+2} + \ldots + r_T$$

- Time horizon can be finite / infinite / indefinite
- Rather use a *discounted return*

$$G_t \doteq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

- Discount factor γ says how much you care about immediate/future
  - if $\gamma = 1$, get a reward on step 1000 is as good as on step 3
  - if $0 < \gamma < 1$, more important to get rewards sooner
  - if $\gamma = 0$, only care for immediate reward, ignore future (myopic)

Sequential decision making

- Credit assignment problem
  - Rewards can be extremely delayed ; how to select actions that lead to a certain outcome ?
- A policy $\pi(a|s)$ is a mapping from states to probabilities of selecting each possible action – optimal policy π^*
- The **state** value function of s_t under π is the expected return

$$\boxed{V^\pi(s_t) \doteq \mathbb{E}_\pi[G_t|s_t] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t]}$$

- The **state-action** value function of taking a_t in s_t under π is

$$Q^\pi(s_t, a_t) \doteq \mathbb{E}_\pi[G_t|s_t, a_t] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1}|s_t, a_t]$$

- Goal : select actions to maximise total expected future rewards
  - Requires to balance immediate and long term rewards
  - Requires a strategy – balance exploration vs exploitation

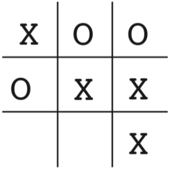
How to learn a policy ?

- Brute force ?
  - evaluate all policies and return the best one : π^*
  - check your understanding : how many different policies ?
- Dynamic programming – when you know P and R (course 3)
  - Policy iteration (evaluation+improvement), value iteration
- Temporal Difference (courses 4 and 5)
- Monte Carlo methods (discussed in courses 4)

"If one had to identify one idea as central and novel to reinforcement learning, it would undoubtedly be temporal-difference (TD) learning."
Sutton and Barto, 2018.

Example : TicTacToe

- Consider the TicTacToe game
- Let's assume an imperfect player
- How can we train a player that finds the imperfections opponent's play and learns to maximise its chances of winning ?
  - Minmax ? No – assumption of perfect player
  - Dynamic programming ? Meh – need a perfect model = complete specification of opponent
- This information can be estimated from experience, by playing many games (e.g. evolutionary method : hill-climb the policy space)

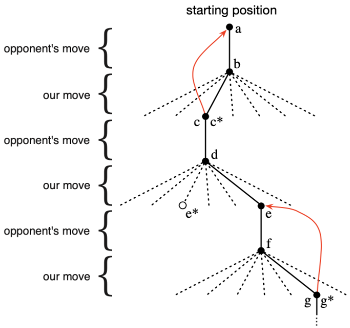


Example : TicTacToe

- With the value function : table of numbers for each state
  - Record the latest estimate of the probability of winning from the state
  - Initialised with 1 for win states, 0 for loose states, 0.5 for others (50% chance of winning)
- Play games
  - Select move by looking up current values in the table
    - Greedy strategy, sometimes random choice (exploratory)
  - Update the table to make the values more accurate estimates of p(win)
    - The current value of earlier state updated to be closer to the value of the later state
    - Example of temporal difference : with s_t the state before the greedy move, s_{t+1} the state after the move, and α a small positive step size :

$$V(s_t) \leftarrow V(s_t) + \alpha[V(s_{t+1}) - V(s_t)]$$

Example : TicTacToe



- Solid black : moves taken
- Dashed : our possible moves
  - Our second move was exploratory (e^* was ranked higher)
  - (Note : no learning for exploratory moves)
- Red : Updates of estimated values

Example : TicTacToe

- For any fixed opponent, this method converges to the true probabilities of winning from each state
    - Given optimal play by our agent
    - Provided that the step size parameter is reduced properly over time
- The greedy moves taken are optimal against this (imperfect) opponent
    - It is an optimal policy

Example : TicTacToe

- Difference with evolutionary methods
  - Hold a policy fixed and play many games to estimate p(win)
  - Change the policy only after many games
  - What happens *during* the games is ignored
  - When the player wins, *all* the behaviour is given credit (including moves that never happened)
- Value function methods evaluate individual states

Example : TicTacToe

- Key RL features
  - Learn while interacting with the environment
  - Achieves the effect of planning and lookahead without a model of the environment and without an explicit search over possible sequences of future states and actions
- RL applies to problems
  - with no opponent, not limited to episodes, when the environment keeps changing, when the magnitude of rewards can change, and to continuous-time problems

Today's lab

- Install Jupyter notebook if not done yet
- Install Gymnasium (https ://gymnasium.farama.org)
  - Check https ://github.com/Farama-Foundation/Gymnasium
- If there is time, start implementing TicTacToe with 1/2 player(s)