Reinforcement Learning

from scratch

Rémi Besson
rb.remi.besson@gmail.com

Frédéric Logé
frederic.logemunerel@gmail.com

ENSAI
March 2nd & 3rd, 2023

Overview of this course

Two essential concepts:

- dynamic programming
- exploration-exploitation dilemma

Hour slot	March 2nd	March 3rd	March 20th
9h45 - 11h15		DYN PRO IN RL	EXPLORATION
11h30 - 12h45			
12h45 - 14h00	BREAK		
14h00 - 15h15	INTRO + DYN PRO		PROJECTS
15h30 - 17h00			

We will also code using `R`, building RL algorithms from scratch.

Evaluation of this course

Evaluation of the course will be done on:

- Group homework (start: Monday 6th, end: 14th, max: 5 people)
- Group project (start: Monday 20th, end: 28th, max: 5 people)

Important

1. To submit, put all your code, report, etc into a folder and zip it.
2. Send the zip file to frederic.logemunerel@gmail.com
3. As title please put "ENSAI RL - Name1 Name2 Name3 ..." with the names of all the participants. Please put the full list of names also somewhere inside the archive, so that we know who you worked with.

Program for today

Machine Learning taxonomy

Shortest Path Problem #1

Shortest Path Problem #2

Shortest Path Problem #3

Homework

Some project ideas

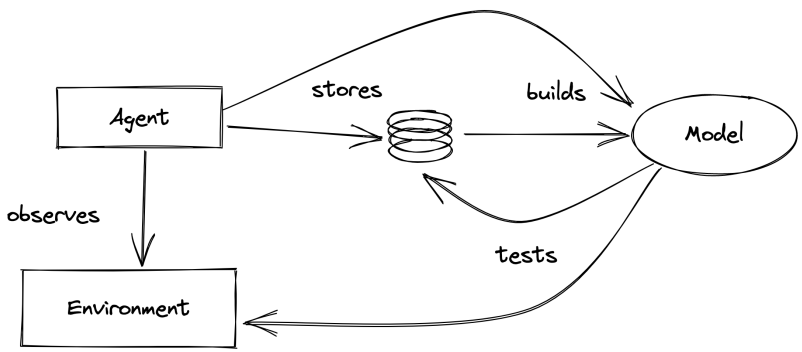
Machine Learning taxonomy

What is Machine Learning ?

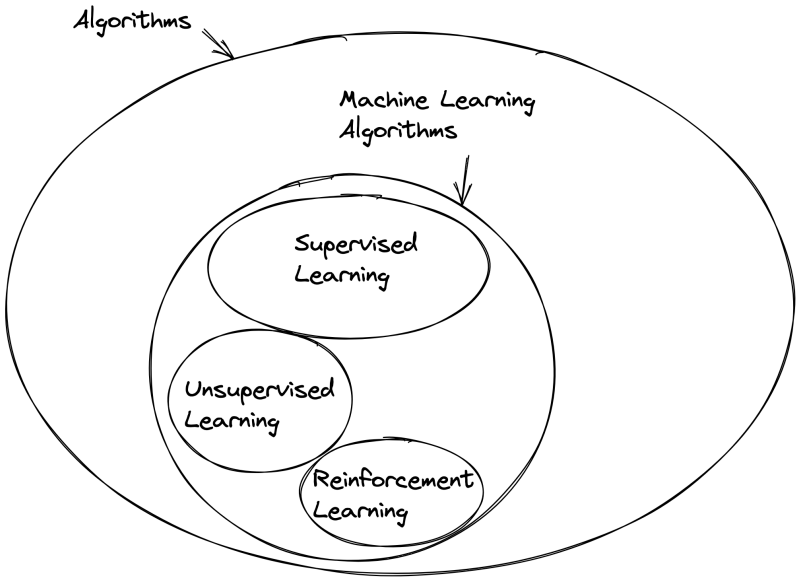
Definition of ML by Tom Mitchell

A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E.

Process



Categories



What is Unsupervised Learning ?

Definition

Based on collected i.i.d. samples $(X_i)_{i=1}^n$, (X: features) build an algorithm to summarize the dataset, either by grouping samples or by grouping features together preserving data structure.

Examples

- Reduce dimension of dataset with large number of attributes
- Find customers with similar shopping patterns
- Identify similar economic phases across countries

Typically, part of the Exploratory Data Analysis part in a Data Science project.

What is Supervised Learning ?

Definition

Based on collected i.i.d. samples $(X_i, Y_i)_{i=1}^n$, build a prediction algorithm $\hat{f}$ such that for any new sample X_{n+1}, $\hat{f}(X_{n+1}) \approx Y_{n+1}$ i.e. minimizes prediction error.

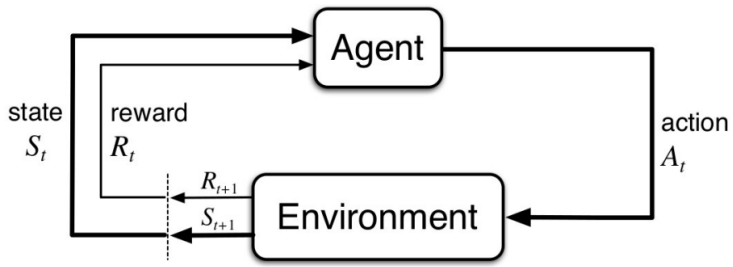
Examples

- Predict stock levels on next day
- Predict object position in next time
- Explain macro-economic impacts from country to country

What is Reinforcement Learning ?

RL algorithms are designed to learn which decisions to take for a good long-term return. At any time $t \in \mathbb{N}$, the agent (you):

1. is in state $s_t \in \mathcal{S}$
2. takes action $a_t \in \mathcal{A}$
3. receives reward $r_{t+1} \in \mathrm{R}$
4. reaches state $s_{t+1} \in \mathcal{S}$



Objective: find $\hat{f}$ which maps $\mathcal{S}$ to $\mathcal{A}$ which solves:

$$\hat{f} = \arg\max_{f:\mathcal{S}\to\mathcal{A}} \mathbb{E}\left[\sum_{t\geq t_0} R(S_t, f(S_{t-1}))\right].$$

E.g. you are in your favorite city, in location A, you go towards location B and each step you take is an action. Each step costs 1 (i.e. reward is -1) and when you arrive the sequence ends.

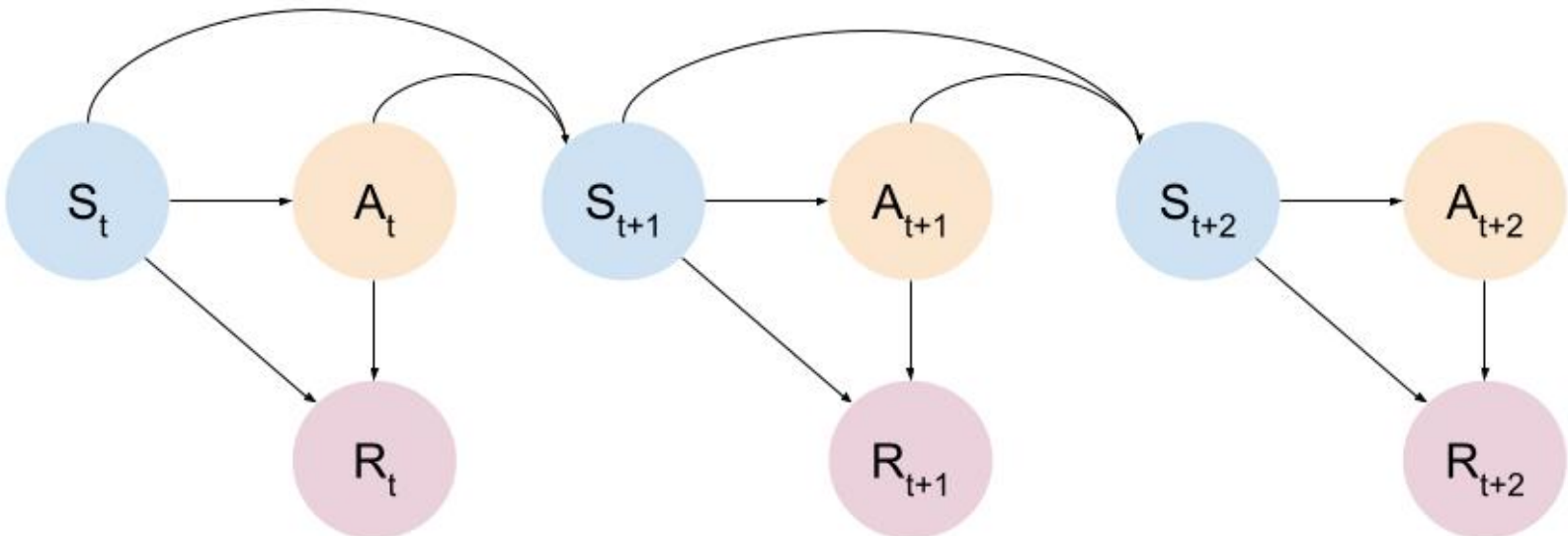
What is RL, compared to SL ?

Take the previous example: you are in your favorite city, in location A and you wish to go to location B.

Supervised Learning can help you predict where you will end up if you take a given step (e.g. N-W-S-E). It is a one-step view only.

Reinforcement learning is designed to deal with sequential decision-making i.e. multi-step view, and usually very long term.

Graphically, it looks like the following Markov Decision Process, the go-to model for RL problems:

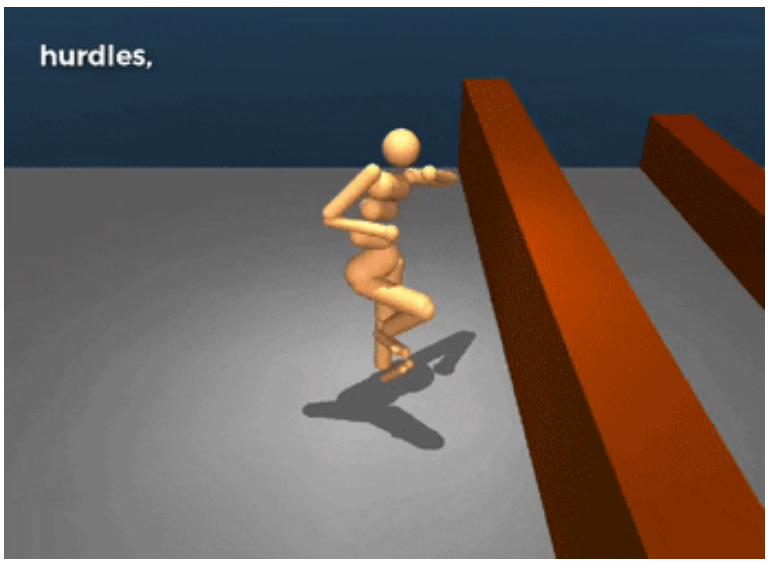


What is Reinforcement Learning ?

Definition

Based on samples $(S_t, A_t, R_t)_{t=1}^{T}$ (S: state, A: action, R: reward), build a decision-making algorithm $\hat{f}$ such that for any new state S_{T+1}, $\hat{f}(S_{T+1})$ maximizes future rewards.

Learning Parkour



$\rightarrow$

Log file

{

position, *environment*,

action, *reward*

}

$\downarrow$

Action recommendation system

What is Reinforcement Learning ?

Notion	Description
Agent	the element making decisions robot
System	(syn. environment) the setup with which the agent interacts obstacle course
State $\mathcal{S}$	current infos on agent and system (S_t) robot parameters & position in course
Action $\mathcal{A}$	interaction of the agent with the system (A_t) change in model parameters
Reward R	quantifies how good a decision was on short-term a necessary feedback for the machine (R_t) $r_0 \cdot 1\{\texttt{passed obstacle}\} + l_0 \cdot 1\{\texttt{fell}\}$
Policy π	action recommendation system how to change parameters given course ?
Strategy	reinforcement learning approach to learn best policy what part of the system to (not) explore ?

Quizz: identify the ML category

1. House prices: predict the market price from characteristics.
2. Cliff problem: on a map, find path from start to end.
3. Dinosaur run: the famous Chrome game.
4. Chess: learn to play the game.
5. Sales: predict which of your product is going to sell best.
6. Train delays: send emails if major delays are expected.
7. Influencers: find influencing accounts in Twitter network.
8. PacMan: the famous game.
9. Autonomous Driving: build the software for Tesla car.
10. Customer Churn: identify customers at risk.

For the problems identified as RL problems, let's identify the state, the action and the reward.

Big news on Reinforcement learning


Le Monde

PIXELS

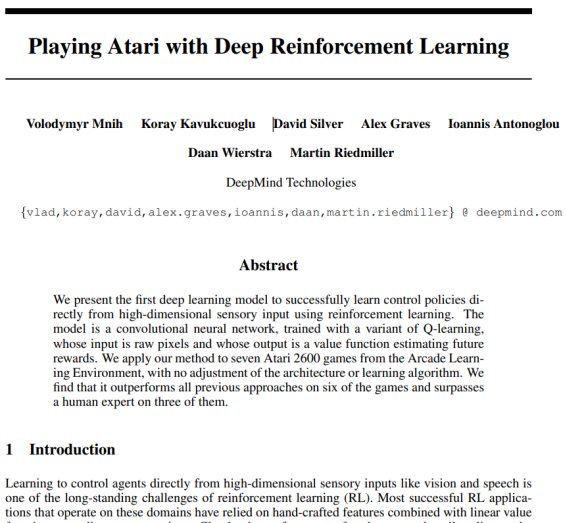
Intelligence artificielle : toujours plus puissant, AlphaGo apprend désormais sans données humaines

La nouvelle version du programme de Google DeepMind, qui avait battu les meilleurs professionnels du jeu de go, se montre beaucoup plus puissante que son prédécesseur.

Par Morgane Tual

Pour soutenir le travail de toute une rédaction, nous vous proposons de vous abonner.

S'abonner


Playing Atari with Deep Reinforcement Learning

Volodymyr Mnih Koray Kavukcuoglu David Silver Alex Graves Ioannis Antonoglou

Daan Wierstra Martin Riedmiller

DeepMind Technologies

{vlad,koray,david,alex.graves,ioannis,daan,martin.riedmiller} @ deepmind.com

Abstract

We present the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. The model is a convolutional neural network, trained with a variant of Q-learning, whose input is raw pixels and whose output is a value function estimating future rewards. We apply our method to seven Atari 2600 games from the Arcade Learning Environment, with no adjustment of the architecture or learning algorithm. We find that it outperforms all previous approaches on six of the games and surpasses a human expert on three of them.

1 Introduction

Learning to control agents directly from high-dimensional sensory inputs like vision and speech is one of the long-standing challenges of reinforcement learning (RL). Most successful RL applications that operate on these domains have relied on hand-crafted features combined with linear value

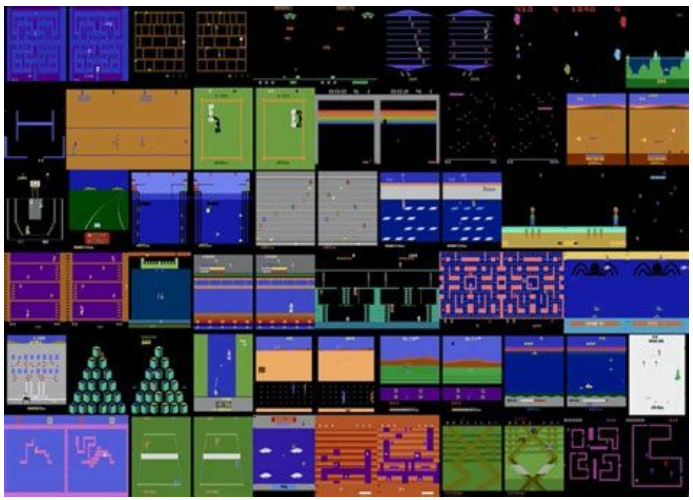

Le Point Tech & Net

AlphaGo Zero, l'IA autodidacte qui a terrassé AlphaGo

En 2016, la machine avait battu l'homme au jeu de go, qui suppose de l'intuition. Un an plus tard, une nouvelle version l'a battue à plate couture !

Source AFP

1€ POUR 2 MOIS



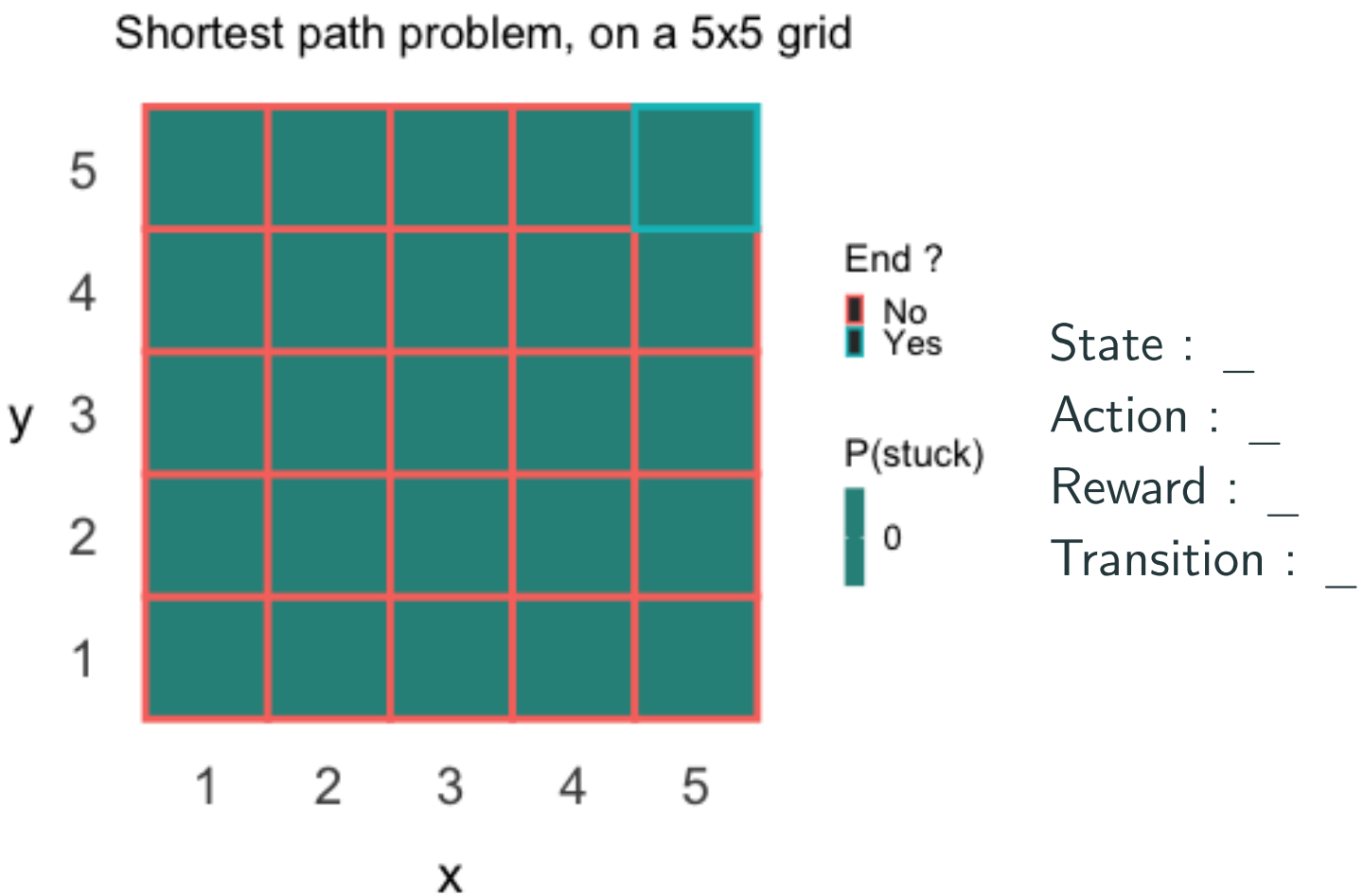
Now used also for Large Language Model training, a very interesting approach to tackling difficult problems.

Shortest Path Problem #1

Problem definition

Goal : guide an agent towards an exit position in the shortest amount of time possible.

Define state, action, reward in this problem.



State : _

Action : _

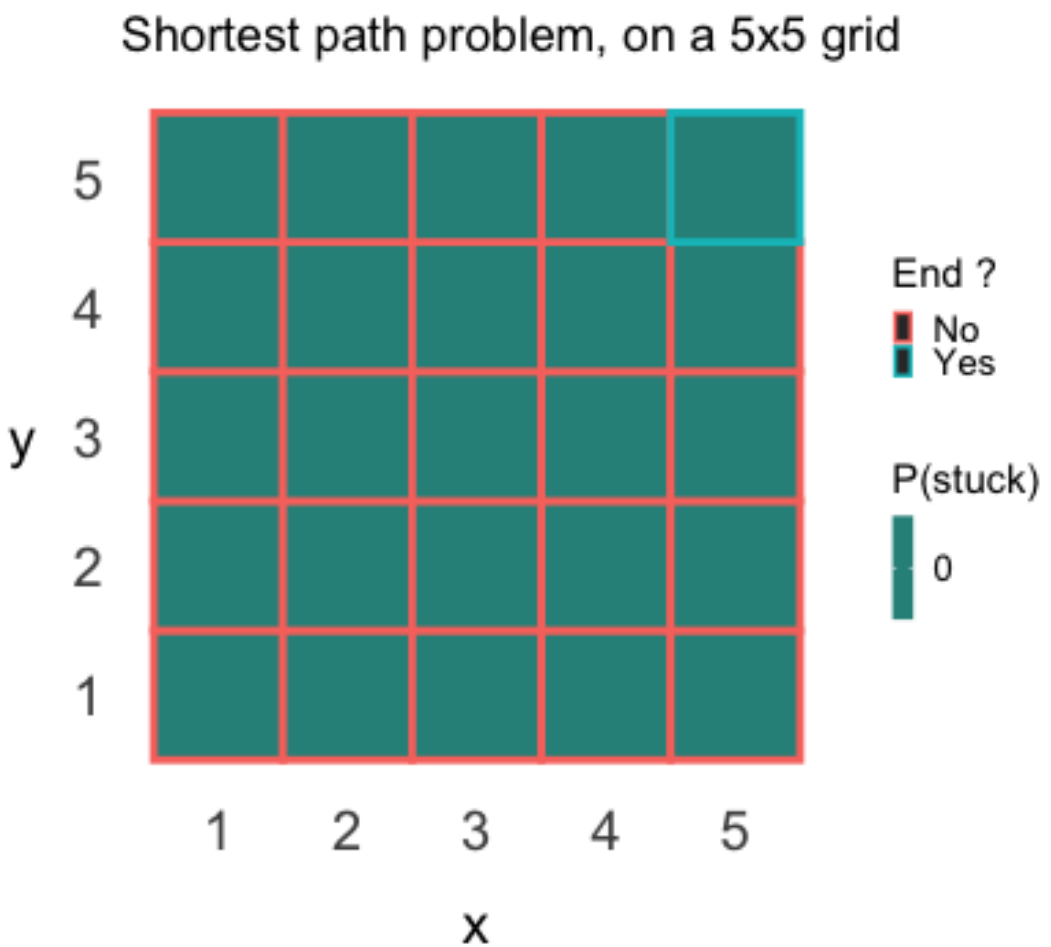
Reward : _

Transition : _

Problem definition

Goal : guide an agent towards an exit position in the shortest amount of time possible.

Define state, action, reward in this problem.



State : position on the grid
Action : N-W-S-E (unless border)
Reward : -1
Transition : deterministic

Resolution

How do we find the shortest path to the exit cell ?

1. Brute force approach
2. Explicit approach

Coding time $\rightarrow$ 💻

Resolution

How about we break down the problem ?

Defining $\texttt{min\_steps}(s_A, s_C)$ as the minimum number of steps to go from state s_A to state s_C. We have that:

$$\texttt{min\_steps}(s_A, s_C) = \min_{s_B \in \mathcal{S}} \left(\texttt{min\_steps}(s_A, s_B) + \texttt{min\_steps}(s_B, s_C)\right).$$

Now that we have a recursive function over the `min_steps` function, let's try to compute it. Calling $V(s) = -\texttt{min\_steps}(s, s_{END})$, we have:

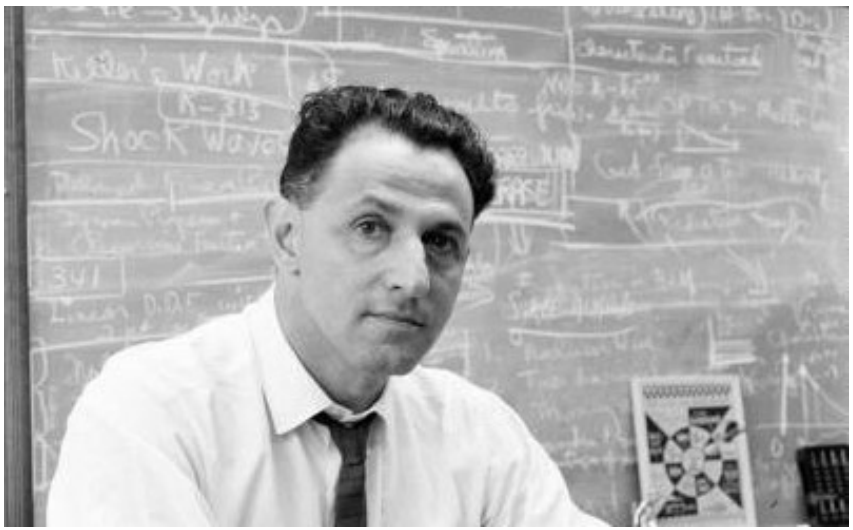
Pseudo Algorithm $\rightarrow$ 💻

Initialize $V(s) = -1000 \ \forall s \in \mathcal{S}$
Until convergence:
    For every state $s \in \mathcal{S}$:
        If s not s_{END}:
            $V(s) = -1 + \max_{s' \in neighbour(s)} V_{cache}(s')$

Dynamic programming

The pseudo-algorithm above is applying the Dynamic Programming principle.

Dynamic programming ($\sim$ 1953) "=" solving an optimisation problem by decomposing it in a recursive manner into interrelated sub problems.



Richard Bellman, father of Dynamic Programming

Dynamic programming

The pseudo-algorithm above is applying the Dynamic Programming principle.

Dynamic programming ($\sim$ 1953) "=" solving an optimisation problem by decomposing it in a recursive manner into interrelated sub problems.

Taking $\forall s, V_\pi(s) = \mathbb{E}_\pi[\sum_{t\geq 0} \mathrm{R}(S_t)|S_0 = s]$ we have indeed the following decomposition

$$V_\pi(s) = \mathbb{E}_\pi[\mathrm{R}(s, \pi(s))] + \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V_\pi(s') \tag{1}$$

which is a recursive equation.

Policy Iteration

As such, if the reward function R and the transition function P are known, then it gives us an approach to find the optimal policy π^*:

Policy Iteration, pseudo algorithm

Initialize randomly $\pi : \mathcal{S} \to \mathcal{A}$

Initialize also V_π to a pertinent value

Until convergence:

For every non-terminal state $s \in \mathcal{S}$:

$$V_\pi(s) = \mathbb{E}_\pi[\mathrm{R}(s, \pi(s))] + \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V_\pi(s') \qquad \text{Evaluation}$$

$$\pi(s) = \arg\max_{a \in \mathcal{A}} (\mathbb{E}[\mathrm{R}(s, a)] + \sum_{s' \in \mathcal{S}} P(s'|s, a) V_\pi(s')) \qquad \text{Improvement}$$

Return (V_π, π)

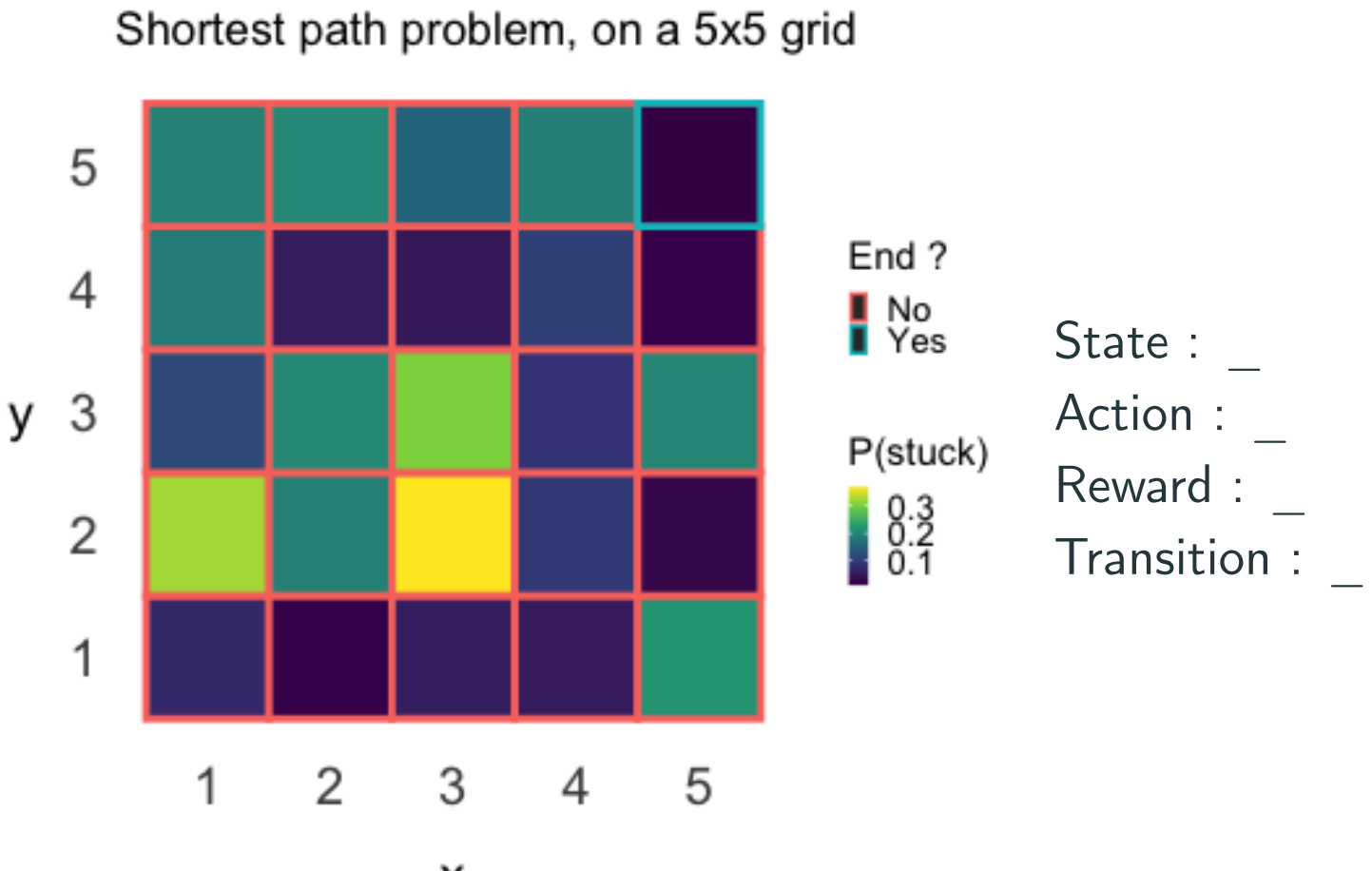
The algorithm alternates between policy evaluation and improvement. Under some conditions on R and P, this algorithm can be proven to converge. Value Iteration algorithm merges the two phases.

Shortest Path Problem #2

Shortest path problem on a grid, problem #2

Goal : guide an agent towards a required position in the shortest amount of time possible. Twist on problem #1: on some cells, there is a known probability of getting stuck temporarily.

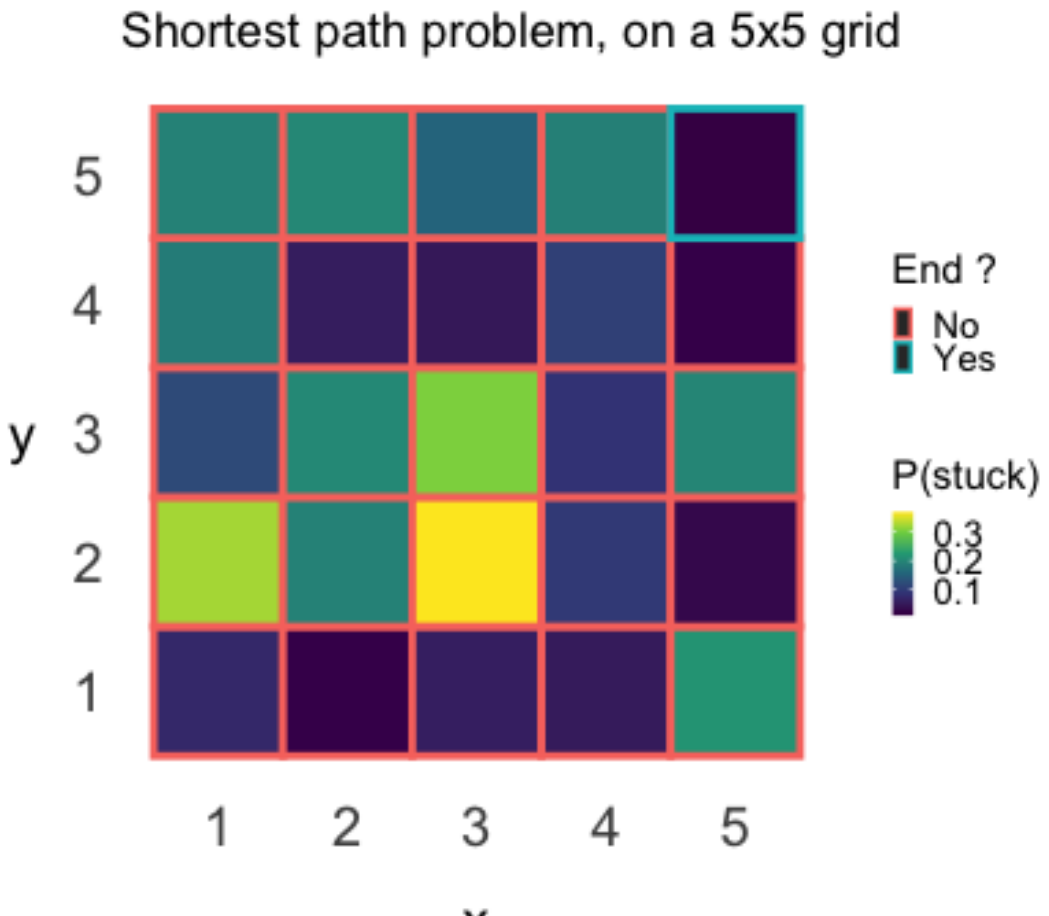
Define state, action, reward in this problem.



Shortest path problem on a grid, problem #2

Goal : guide an agent towards a required position in the shortest amount of time possible. Twist on problem #1: on some cells, there is a known probability of getting stuck temporarily.

Define state, action, reward in this problem.



State : position on the grid
Action : N-W-S-E (except border)
Reward : -1
Transition : probabilistic

Resolution

Pseudo Algorithm $\rightarrow$ 💻

Initialize $V(s) \forall s \in \mathcal{S}$

Until convergence:

  For every state $s \in \mathcal{S}$:

    If s not s_{END}:

      $V(s) = -1 + \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s, a) V(s')$

Open questions

1. Change the algorithm when reward is random
2. What happens if we don't know the probability of getting stuck ?
3. What if, when we get stuck, it is for a random amount of time ?

Shortest Path Problem #3

Shortest path problem on a grid, problem #3

Assume now that you have observed samples $(S_t, A_t, R_t)_{t>0}$ for a given Markov Decision Process.

How would you go about constructing an estimate for the best possible policy ?

Q-Learning algorithm

Let us consider $Q_\pi(s,a) = \mathbb{E}[\sum_{t\geq 0} R(s_t, \pi(s_t))|S_0 = s, A_0 = a]$ which is similar to the Bellman Value V_π of the slides before, only we condition on the first action taken.

Based on a sample (S_t, A_t, R_t, S_{t+1}), the idea is to update $Q_\pi(s,a)$, $\forall s \in \mathcal{S}, a \in \mathcal{A}$ using the following updating rule:

$$Q_\pi(S_t, A_t) \leftarrow (1-\eta) \cdot \underbrace{Q_\pi(S_t, A_t)}_{\text{old estimate}} + \eta \cdot \underbrace{(R_t + \gamma \max_{a\in\mathcal{A}} Q_\pi(S_{t+1}, a))}_{\text{new estimate}} \quad (2)$$

The pseudo-code using this update rule is called the Q-Learning algorithm, and is provided in the next slide.

Q-Learning pseudo-code

Most of the value-based algorithms share this structure:

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
    $S \leftarrow S'$
  until S is terminal

possible to learn a parametric version of $Q(S, A)$, Mnih et al. 2013.

Monte-Carlo

Q-Learning is a TD-Learning type approach, where we look only at a single sample (S_t, A_t, R_t, S_{t+1}) for an update.

Another approach is to wait longer to provide an update. Typically, the Monte-Carlo approach consists in taking as estimate the realized sum of rewards for a single sequence:

$$Q_\pi(S_t, A_t) \leftarrow (1-\eta) \cdot \underbrace{Q_\pi(S_t, A_t)}_{\text{old estimate}} + \eta \cdot \underbrace{\sum_{t' \geq t} R_{t'}}_{\text{new estimate}} \quad . \tag{3}$$

Homework

Homework

Here are the tasks for the homework:

1. Prove equation 1.
2. Implement efficiently the Fibonacci computation (*recursion*).
3. Solve the Arsène Lupin problem (*dynamic programming*).
4. Implement Puissance 4 game (*qlearning*).

All the tasks are described in **the next slides**.

Fibonacci computation

The Fibonacci sequence (F_n) is defined as $F_0 = 0$, $F_1 = 1$, $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$.

Code a function that returns $(F_i; i \leq n)$ for input $n \geq 2$ in linear time i.e. at most n iterations.

The Arsène Lupin problem

As Arsène Lupin, you plan to rob the different houses of a street. Each house has a specific amount of money in it, known in advanced. However, if you rob one house, you cannot rob the one next to it, you have to skip it, because of their sophisticated connected detection system.

Given a vector of money amounts in input, build an algorithm to figure out which houses to rob to collect the maximum total amount of money.

Example: if the amounts are c(house 1: 10, house 2: 10, house 3: 4, house 4: 2), then the maximum is 14, achieved by robbing house 1 and house 3.

Provide your own test examples and a nice visualization of the problem.

Bonus: in an app, allow the user to change the amounts to check how the optimal solution changes.

Puissance 4 implementation

The game of Puissance 4 is quite straightforward regarding rules.

Your task is to build a function with generates a Puissance 4 play, following random policies for both players. Build a visualization of the game on your development interface.

Bonus: implement an RL algorithm to find the optimal action to take (it can be Q-Learning or more basic ones).

Hint: you can focus exclusively on one player for simplicity.

Some project ideas

Generalities

Tackle a decision-making problem with Reinforcement Learning

Group Project (max 5 people), due date : March 28th 2023

Steps

- Modeling the problem as a Markov Decision Process (state, action, reward, transitions are defined)
- Data collection and/or simulator
- System visualization
- Learning strategies to define
- Results: hyperparameters that work, optimal policy found, quali/quanti comparison with a baseline policy

Expected output

GitHub repository with the code that works and a markdown file, such as README.md presenting the findings and graphics you got. A nice .gif file with the agent playing the optimal agent would be great.

Project suggestions

Games

Puissance4 / Quarto

Autonomous Driving

Follow-the-line Car / Car with traffic

Healthcare

Continuous / Episodic drug prescription

Marketing

Personalized recommendation / Retaining customers

Finance

Trading / Dynamic Pricing

... these are only our suggestions, original projects are encouraged !