Introduction au reinforcement learning

Ghazi Bel Mufti

ESSAI - 2023

Table of Contents

Machine learning

Le *Machine learning* (apprentissage automatique) est la science de programmer les ordinateurs de sorte qu'ils puissent apprendre à partir de données :

- **Unsupervised Learning (apprentissage non supervisé)**

OBJECTIF Découvrir les structures sous-jacentes à des données non étiquetées.

DONNÉES Données non étiquetées.

EXEMPLES Segmenter les clients en groupes semblables ; détection d'anomalies ; visualisation des données par réduction de la dimensionalité.

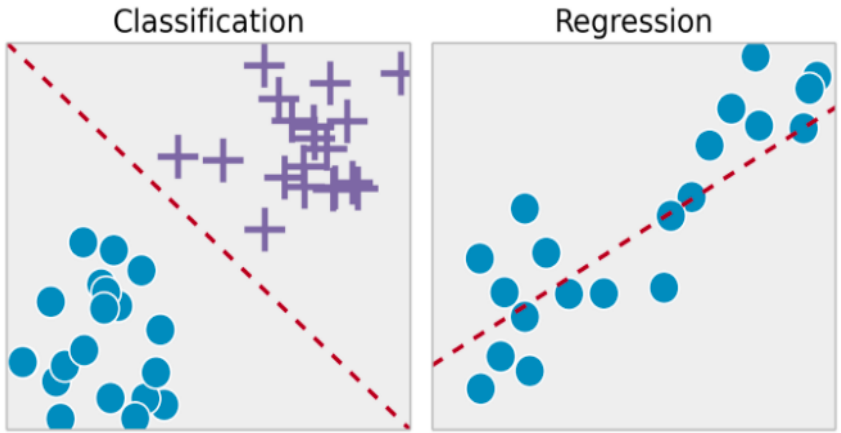
Machine learning

- **Supervised Learning (apprentissage supervisé)**

OBJECTIF Apprendre une fonction de prédiction à partir de données annotées

DONNÉES Données étiquetées

EXEMPLES Prédiction d'une classe dans des données naturelles (photos, manuscrits, paroles, etc.) ; prédiction d'une valeur dans des problèmes de régression.



Machine Learning

- **Reinforcement Learning (apprentissage par renforcement)**

OBJECTIF Laisser l'algorithme apprendre de ses propres erreurs. Face à un choix aléatoire au départ, s'il se trompe il est « puni », dans le cas contraire, une bonne décision est « récompensée ».

DONNÉES Etats et Actions

Alors que l'objetif de l'apprentissage supervisé et non supervisé est la minimisation de l'erreur, l'apprentissage par renforcement vise à maximiser la récompense !

Reinforcement learning : quelques définitions I

- **Agent :** une entité autonome qui agit, orientant son activité vers la réalisation d'objectifs, sur un environnement en utilisant l'observation à travers des capteurs et des actionneurs conséquents.
- **État :** il contient les valeurs prises par les variables permettant de localiser l'agent relativement à l'environnement et à ses composants. L'état d'un agent peut inclure sa position, sa vitesse, la position d'autres objets, ...

  $\rightarrow$ On appelle S l'ensemble des états.

- **Action :** les actions correspondent aux comportements possibles que l'agent peut adopter vis-à-vis de l'environnement.

  $\rightarrow$ On appelle A l'ensemble des actions.

Reinforcement learning : quelques définitions II

- **Récompense :** à chaque instant t, l'agent qui se trouve à l'état s_t et choisit d'effectuer une action a_t reçoit en contrepartie une récompense r_t qui peut être positive, négative ou nulle :
  - $r_t > 0$ si l'agent se comporte bien
  - $r_t < 0$ sinon

- En RL, un *agent* procède à des *observations* et réalise des *actions* au sein d'un *environnement*. Il recoit, en retour des *récompenses*.
- Son objectif est d'apprendre à agir de façon à maximiser les récompenses espérées sur le long terme.

Exemples d'applications...

- **Robotique :** l'*agent* est le programme qui contrôle un robot qui marche dans le monde réel (*environnement*) qu'il observe au travers d'un ensemble de capteurs. Les *actions* consistent à envoyer des signaux pour activer les moteurs. Il reçoit des *récompenses* positives lorsqu'il se rapproche de la destination visée et négatives s'il s'en éloigne.

Exemples d'applications...

- **Robotique :** l'*agent* est le programme qui contrôle un robot qui marche dans le monde réel (*environnement*) qu'il observe au travers d'un ensemble de capteurs. Les *actions* consistent à envoyer des signaux pour activer les moteurs. Il reçoit des *récompenses* positives lorsqu'il se rapproche de la destination visée et négatives s'il s'en éloigne.
- **Gamming :** l'agent peut être le programme qui contrôle Ms Pac-Man. Dans ce cas l'environnement est une simulation du jeu Atari, les actions sont les neufs positions possibles du joystick(en haut à gauche, en bas, au centre, ...), les observations sont les captures d'écran et les récompenses sont les points gagnés au jeu.

Exemples d'applications...

- **Robotique :** l'*agent* est le programme qui contrôle un robot qui marche dans le monde réel (*environnement*) qu'il observe au travers d'un ensemble de capteurs. Les *actions* consistent à envoyer des signaux pour activer les moteurs. Il reçoit des *récompenses* positives lorsqu'il se rapproche de la destination visée et négatives s'il s'en éloigne.
- **Gamming :** l'agent peut être le programme qui contrôle Ms Pac-Man. Dans ce cas l'environnement est une simulation du jeu Atari, les actions sont les neufs positions possibles du joystick(en haut à gauche, en bas, au centre, ...), les observations sont les captures d'écran et les récompenses sont les points gagnés au jeu.
- L'agent ne contrôle pas forcémment le déplacement d'un objet physique ou virtuel. Il peut s'agir d'un **thermostat intelligent** qui reçoit des récompenses positives dès qu'il permet d'atteindre la température visée pour économiser de l'énergie et négatives en cas d'intervention humaine.

Exemples d'applications...

- **Robotique :** l'*agent* est le programme qui contrôle un robot qui marche dans le monde réel (*environnement*) qu'il observe au travers d'un ensemble de capteurs. Les *actions* consistent à envoyer des signaux pour activer les moteurs. Il reçoit des *récompenses* positives lorsqu'il se rapproche de la destination visée et négatives s'il s'en éloigne.
- **Gamming :** l'agent peut être le programme qui contrôle Ms Pac-Man. Dans ce cas l'environnement est une simulation du jeu Atari, les actions sont les neufs positions possibles du joystick(en haut à gauche, en bas, au centre, ...), les observations sont les captures d'écran et les récompenses sont les points gagnés au jeu.
- L'agent ne contrôle pas forcémment le déplacement d'un objet physique ou virtuel. Il peut s'agir d'un **thermostat intelligent** qui reçoit des récompenses positives dès qu'il permet d'atteindre la température visée pour économiser de l'énergie et négatives en cas d'intervention humaine.
- **Finance :** l'agent observe les prix des actions et décide du nombre d'actions à acheter ou à vendre chaque seconde. Les récompenses dépendent des gains et des pertes.

Exemples d'applications...

- **Robotique :** l'*agent* est le programme qui contrôle un robot qui marche dans le monde réel (*environnement*) qu'il observe au travers d'un ensemble de capteurs. Les *actions* consistent à envoyer des signaux pour activer les moteurs. Il reçoit des *récompenses* positives lorsqu'il se rapproche de la destination visée et négatives s'il s'en éloigne.
- **Gamming :** l'agent peut être le programme qui contrôle Ms Pac-Man. Dans ce cas l'environnement est une simulation du jeu Atari, les actions sont les neufs positions possibles du joystick(en haut à gauche, en bas, au centre, ...), les observations sont les captures d'écran et les récompenses sont les points gagnés au jeu.
- L'agent ne contrôle pas forcémment le déplacement d'un objet physique ou virtuel. Il peut s'agir d'un **thermostat intelligent** qui reçoit des récompenses positives dès qu'il permet d'atteindre la température visée pour économiser de l'énergie et négatives en cas d'intervention humaine.
- **Finance :** l'agent observe les prix des actions et décide du nombre d'actions à acheter ou à vendre chaque seconde. Les récompenses dépendent des gains et des pertes.
- **Trafic routier :** l'agent va gérer les feux de circulation de manière à fluidifier la circulation routière.

Politique

- Une politique (*policy*) est une stratégie qu'un agent suit afin d'atteindre ses objectifs.
  - Elle dicte les actions que l'agent entreprend en fonction de l'état de l'agent et de l'environnement.
  - Une politique π est une fonction de $S \rightarrow A$ qui associe à chaque état s, une action $\pi(s)$ à effectuer.

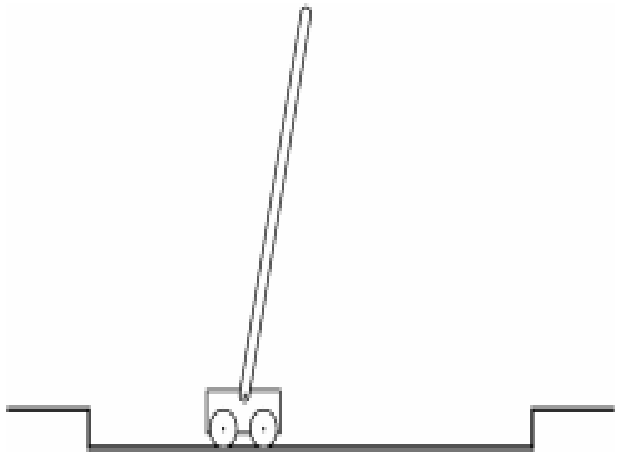
Exemple 1 Un robot aspirateur dont la récompense est le volume de poussière ramassée en 30 mn. Sa politique consiste à avancer, chaque seconde, avec une probabilité p ou de tourner aléatoirement vers la gauche ou la droite avec une probabilité $1 - p$.

Politique

- Une politique (*policy*) est une stratégie qu'un agent suit afin d'atteindre ses objectifs.
  - Elle dicte les actions que l'agent entreprend en fonction de l'état de l'agent et de l'environnement.
  - Une politique π est une fonction de $S \rightarrow A$ qui associe à chaque état s, une action $\pi(s)$ à effectuer.

Exemple 1 Un robot aspirateur dont la récompense est le volume de poussière ramassée en 30 mn. Sa politique consiste à avancer, chaque seconde, avec une probabilité p ou de tourner aléatoirement vers la gauche ou la droite avec une probabilité $1 - p$.

Exemple 2 Un environnement *CartPole* : une politique simple serait de déclencher une accélération vers la gauche lorsque le baton penche vers la gauche et *vice versa*.



Evaluation des actions

- Dans l'état $s_t \in S$, l'agent émet l'action a_t et reçoit en retour une récompense immédiate r_{t+1} et son nouvel état $s_{t+1} \in S$.
- Soit une séquence $r_{t+1}, r_{t+2}, \ldots$ de récompenses reçues en suivant une politique, alors la fonction de récompenses R_t à partir d'un instant t est donnée par :

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

  - $\gamma \in [0, 1]$ est **le paramètre de rabais (discount rate)** qui détermine la valeur présente des récompenses futures.
  - Une récompense reçue k unités de temps plus tard vaut seulement γ^{k-1} ce qu'elle vaudrait au temps courant :
    - $\gamma = 0$ agent *myope* qui maximise les récompenses immédiate.
    - $\gamma \to 1$ agent avec un horizon de plus en plus lointain.

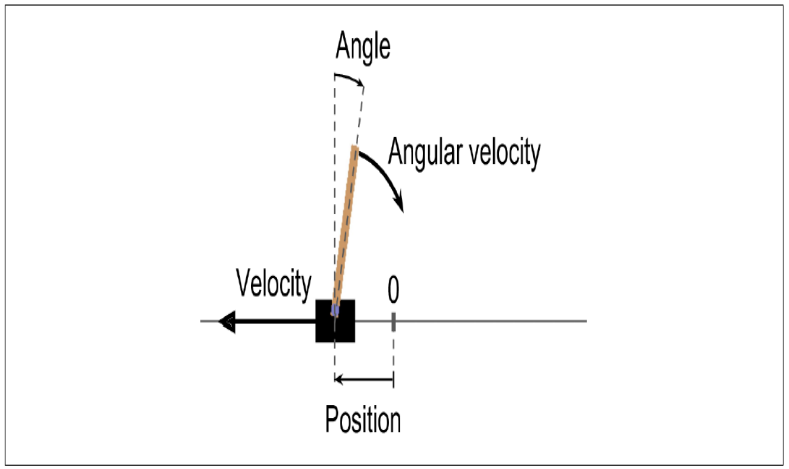
Introduction à OpenAI Gym I

- One of the challenges of Reinforcement Learning is that in order to train an agent, you first need to have a working environment.
- If you want to program an agent that will learn to play an Atari game, you will need an Atari game simulator.
- OpenAI Gym is a toolkit that provides a wide variety of simulated environments (Atari games, board games, 2D and 3D physical simulations, and so on), so you can train agents, compare them, or develop new RL algorithms.

Gym documentation : https ://www.gymlibrary.dev/index.html

Example : CartPole environment I

```
> import gym
> env = gym.make("CartPole-v1")
> obs = env.reset()
> obs
array([-0.01258566, -0.00156614,
0.04207708, -0.00180545])
```



- gym.make("environment name") : returns the environment that was passed as parameter.
- Here, using gym.make(), we've created a CartPole environment. This is a 2D simulation in which a cart can be accelerated left or right in order to balance a pole placed on top of it.

Example : CartPole environment II

- env.reset() : this command will reset the environment. It returns an initial observation.
- After the environment is created, you must initialize it using the reset() method. This returns the first observation.
- Observations depend on the type of environment.
- For the CartPole environment, each observation is a 1D NumPy array containing four floats : these floats represent the cart's horizontal position (0.0 = center), its velocity (positive means right), the angle of the pole (0.0 = vertical), and its angular velocity (positive means clockwise).

- env.render() : this command will display a popup window. Since it is written within a loop, an updated popup window will be rendered for every new action taken in each step.
- The environment is displayed by calling its render() method :

```
> env.render()
True
```

Example : CartPole environment III

- env.action_space : it is used to define characteristics of the action space of the environment. With this, one can state whether the action space is continuous or discrete, define minimum and maximum values of the actions, etc.
- Let's ask the environment what actions are possible :

```
> env.action_space
Discrete(2)
```

- Discrete(2) means that the possible actions are integers 0 and 1, which represent accelerating left (0) or right (1). Other environments may have additional discrete actions, or other kinds of actions (e.g., continuous).

- env.observation_space : which prints Box(4,) a four dimensinal vector of real numbers.
- You can also find out what is the range of each observation variable by env.observation_space.low and env.observation_space.high which will print the minimum and maximum values for each observation variable.

Example : CartPole environment IV

- env.step() : this command will take an action at each step. The action is specified as its parameter. Env.step function returns four parameters, namely observation, reward, done and info. These four are explained below :
- Since the pole is leaning toward the right (obs[2] $>$ 0), let's accelerate the cart toward the right : `> action = 1 # accelerate right`
  `> obs, reward, done, info = env.step(action)`
  `> obs # an environment-specific object representing your observation of the environment.`
  `array([-0.01261699, 0.19292789, 0.04204097, -0.28092127])`
  `> reward  # amount of reward achieved by the previous action. It is a floating data type value. The scale varies between environments.`
  `1.0`
  `> done # A boolean value stating whether it's time to reset the environment again.`
  `False`
  `> info # diagnostic information useful for debugging`
  `{}`

Exercice : Ecrire un scripte qui teste une politique aléatoire dans l'environnement CartPole.

Exercice : Ecrire un scripte qui teste une politique aléatoire dans l'environnement CartPole.

Solution :

```
!pip install gym
import gym
import random
```

Exercice : Ecrire un scripte qui teste une politique aléatoire dans l'environnement CartPole.

Solution :

```
!pip install gym
import gym
import random

env = gym.make('CartPole-v0')
states = env.observation_space
actions = env.action_space.n
```

Exercice : Ecrire un scripte qui teste une politique aléatoire dans l'environnement CartPole.

Solution :

```
!pip install gym
import gym
import random

env = gym.make('CartPole-v0')
states = env.observation_space
actions = env.action_space.n

episodes = 10
for episode in range(1, episodes+1):
    state = env.reset()
    done = False
    score = 0
    while not done:
        env.render()
        action = random.choice([0,1])
        n_state, reward, done, info = env.step(action)
        score+=reward
    print('Episode:{} Score:{}'.format(episode, score))
```

Processus de décision de Markov (PDM)

Hypothèse de Markov

En tout instant t, la probabilité de passer de s à s' en faisant une action a dépend seulement de s et s', et **pas des états précédents**.

$$P(s_{t+1} = s' | s_t = s, a_t = a, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- Un PDM ressemble à une chaîne de Markov : à chaque étape, le processus est dans un certain état s et l'agent choisit une action a. La probabilité que le processus arrive à l'état s' est déterminée par l'action choisie. Il est défini comme un ensemble $(S, A, \mathcal{T}, \mathcal{R})$:
  - $\mathcal{T} : S \times A \times S \to [0, 1]$, fonction de transition qui donne la probabilité d'arriver en s' quand on exécute une action a en un état s :

$$\mathcal{T}(s, a, s') = \mathcal{T}^a_{ss'} = Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

  - $\mathcal{R} : S \times A \to \mathbb{R}$, fonction de récompense où $\mathcal{R}(s, a) = \mathbb{E}(r_{t+1} | s_t = s, a_t = a)$

Processus de décision de Markov (PDM)

Hypothèse de Markov

En tout instant t, la probabilité de passer de s à s' en faisant une action a dépend seulement de s et s', et **pas des états précédents**.

$$P(s_{t+1} = s' | s_t = s, a_t = a, s_{t-1}, a_{t-1}, ..., s_0, a_0) = P(s_{t+1} = s' | s_t = s, a_t = a)$$

- Un PDM ressemble à une chaîne de Markov : à chaque étape, le processus est dans un certain état s et l'agent choisit une action a. La probabilité que le processus arrive à l'état s' est déterminée par l'action choisie. Il est défini comme un ensemble $(S, A, \mathcal{T}, \mathcal{R})$:
  - $\mathcal{T} : S \times A \times S \to [0, 1]$, fonction de transition qui donne la probabilité d'arriver en s' quand on exécute une action a en un état s :

$$\mathcal{T}(s, a, s') = \mathcal{T}^a_{ss'} = Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

  - $\mathcal{R} : S \times A \to \mathbb{R}$, fonction de récompense où $\mathcal{R}(s, a) = \mathbb{E}(r_{t+1} | s_t = s, a_t = a)$

Un problème de RL se définit comme un PDM où $\mathcal{T}$ et $\mathcal{R}$ sont inconnus.

Fonction valeur et Fonction action-valeur

- La fonction valeur d'un état s sous une politique π, dénotée $V^{\pi}(s)$, est le retour moyen obtenu quand on suit π au départ de s :

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}\{R_t | s_t = s\} \\ &= \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \end{aligned}$$

Fonction valeur et Fonction action-valeur

- La fonction valeur d'un état s sous une politique π, dénotée $V^{\pi}(s)$, est le retour moyen obtenu quand on suit π au départ de s :

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}\{R_t | s_t = s\} \\ &= \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} \end{aligned}$$

- La fonction état-valeur définit le retour attendu en partant de l'état s en émettant l'action a suivant la politique π :

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}\{R_t | s_t = s, a_t = a\} \\ &= \mathbb{E}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\right\} \end{aligned}$$

Equation d'optimalité de Bellman I

- La fonction valeur optimale (i.e. l'espérance des gains futurs), correspondant à la politique optimale π^*, en partant de l'état s :

$$V^*(s) = \max_a [\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}^a_{ss'} V^\pi(s')]$$

- Si l'agent agit de façon optimale, alors la valeur optimale de l'état courant est égale à la récompense qu'il obtiendra après avoir effectué une action optimale, plus la valeur optimale espérée pour tous les états suivants possibles vers lesquels conduit cette action.

`Algorithme qui permet de l'atteindre en procédant d'une manière itérative :`

$$V_{k+1}(s) = \max_a [\mathcal{R}(s,a) + \gamma \sum_{s' \in S} \mathcal{T}^a_{ss'} V_k(s')]$$

Equation d'optimalité de Bellman II

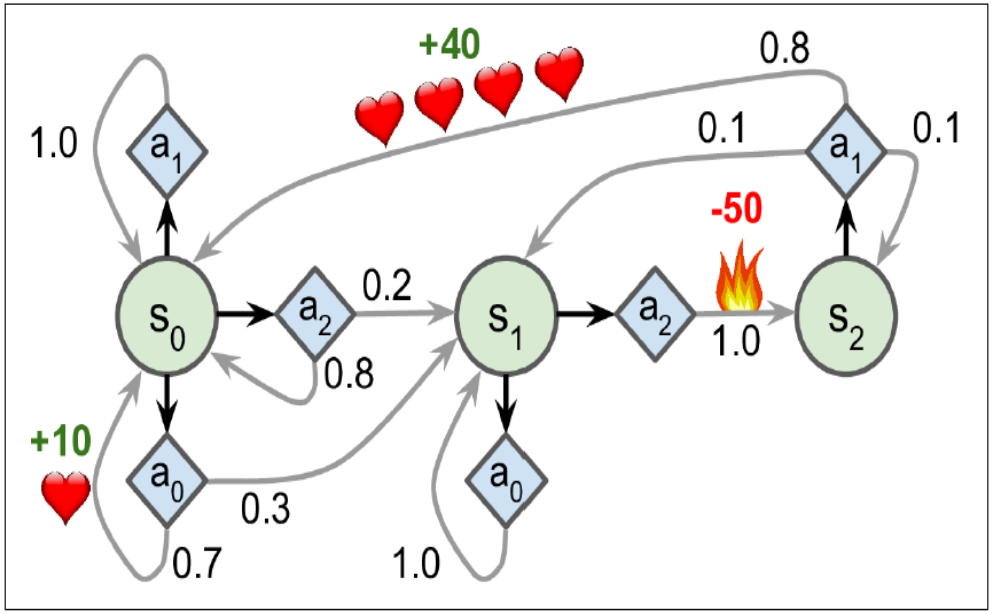
- Bellman propose un algorithme comparable à celui de la fonction de valeur pour estimer la fonction état-valeur optimale à l'état s et en choisissant l'action a :

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \max_{a'}[\sum_{s' \in S} \mathcal{T}^a_{ss'} Q^*(s',a')]$$

`Algorithme d'itération sur la Q-valeur :`

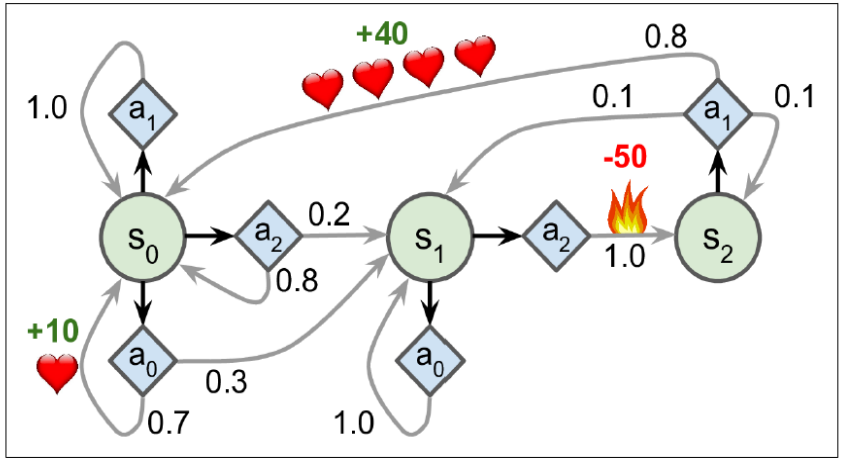
$$Q_{k+1}(s,a) = \mathcal{R}(s,a) + \gamma \max_{a'}[\sum_{s' \in S} \mathcal{T}^a_{ss'} Q_k(s',a')]$$

Example of a Markov decision process I



- For example, the MDP represented here has three states (represented by circles) and up to three possible discrete actions at each step (represented by diamonds).

Example of a Markov decision process II



- If it starts in state s_0 , the agent can choose between actions a_0, a_1, or a_2 . If it chooses action a_1 , it just remains in state s_0 with certainty, and without any reward. But if it chooses action a_0, it has a 70% probability of gaining a reward of $+10$ and remaining in state s_0. It can then try again and again to gain as much reward as possible, but at one point it is going to end up instead in state s_1.
- In state s_1 it has only two possible actions : a_0 or a_2. It can choose to stay put, or it can choose to move on to state s_2 and get a negative reward of –50. In state s_2 it has no other choice than to take action a_1, which will most likely lead it back to state s_0, gaining a reward of $+40$ on the way.

Q–value iteration algorithm I

- First, we need to define the MDP :

```
transition_probabilities = [ # shape=[s, a, s']
        [[0.7, 0.3, 0.0], [1.0, 0.0, 0.0], [0.8, 0.2, 0.0]],
        [[0.0, 1.0, 0.0], None, [0.0, 0.0, 1.0]],
        [None, [0.8, 0.1, 0.1], None]]
rewards = [ # shape=[s, a, s']
        [[+10, 0, 0], [0, 0, 0], [0, 0, 0]],
        [[0, 0, 0], [0, 0, 0], [0, 0, -50]],
        [[0, 0, 0], [+40, 0, 0], [0, 0, 0]]]
possible_actions = [[0, 1, 2], [0, 2], [1]]
```

- As :

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \cdot \max_{a'} Q_k(s',a')\right] \quad \text{for all } (s'a)$$

Q–value iteration algorithm II

- The Q-value Iteration algorithm : it applies the equation above repeatedly, to all Q-values, for every state and every possible action :

```
gamma = 0.90 # the discount factor

for iteration in range(50):
    Q_prev = Q_values.copy()
    for s in range(3):
        for a in possible_actions[s]:
            Q_values[s, a] = np.sum([
                    transition_probabilities[s][a][sp]
                    * (rewards[s][a][sp] + gamma * np.max(Q_prev[sp]))
                for sp in range(3)])
```

- The resulting Q-values look like this :

```
>>> Q_values
array([[18.91891892, 17.02702702, 13.62162162],
       [ 0.        ,        -inf, -4.87971488],
       [       -inf, 50.13365013,        -inf]])
```

Q–value iteration algorithm III

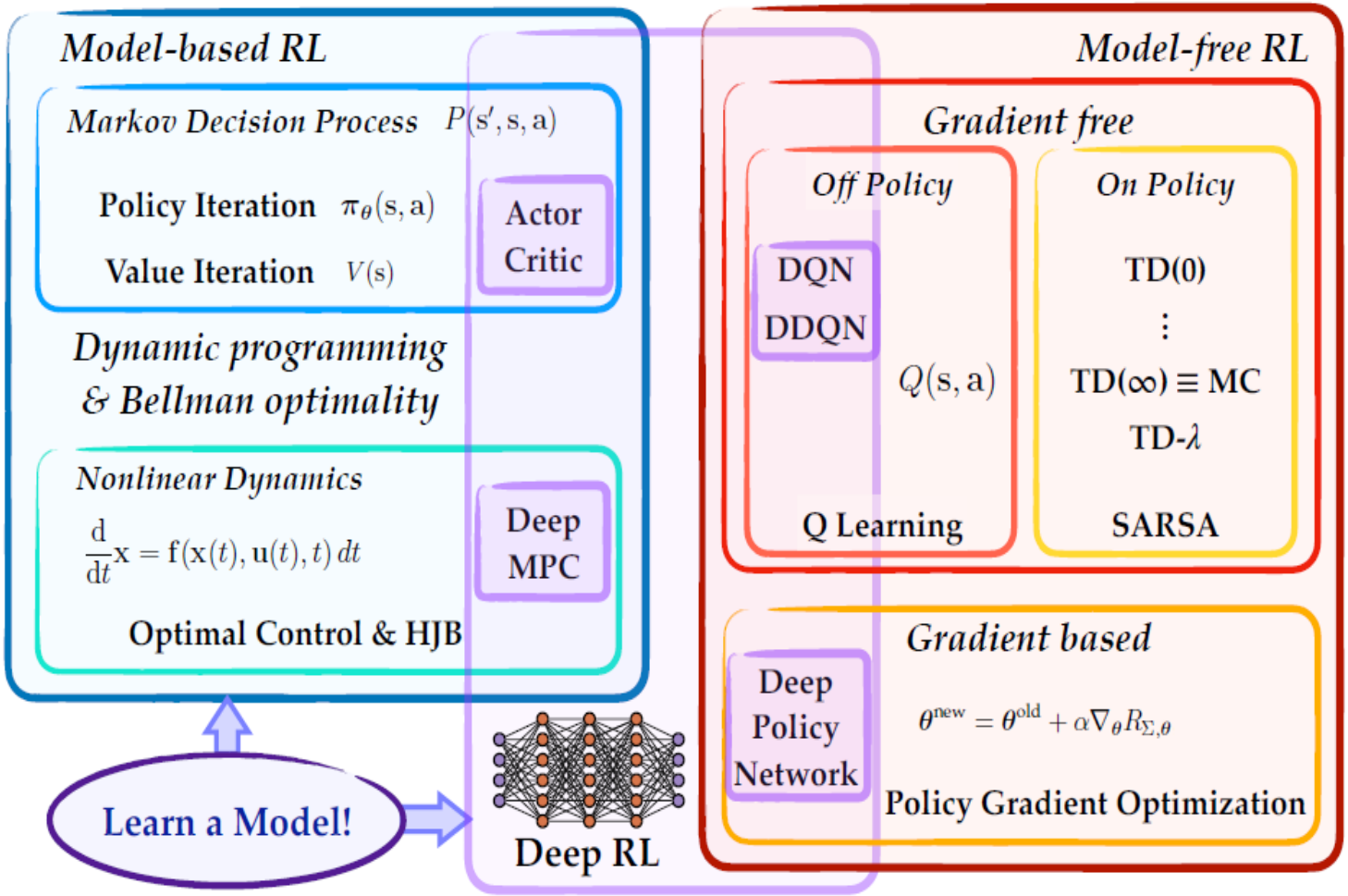
- The resulting Q-values look like this :

```
>>> Q_values
array([[18.91891892, 17.02702702, 13.62162162],
       [ 0.        ,        -inf, -4.87971488],
       [      -inf, 50.13365013,        -inf]])
```

- For example, when the agent is in state s_0 and it chooses action a_1 , the expected sum of discounted future rewards is approximately 17.0.
- For each state, let's look at the action that has the highest Q-Value :

```
>>> np.argmax(Q_values, axis=1) # optimal action for each state
array([0, 0, 1])
```

Catégorisation des techniques de RL



- There are many approaches to learn an optimal policy, which is the ultimate goal of RL. A major dichotomy in reinforcement learning is that of model-based RL versus model-free RL.

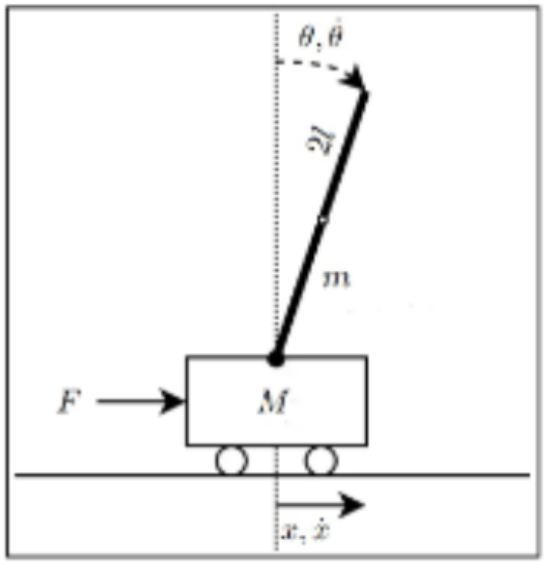
Le Cart-Pole dans le contexte de contrôle optimal I

- Le système non linéaire du CartPole est décrit par les équations suivantes :

$$\ddot{x} = \frac{F + ml(\dot{\theta}^2 sin\theta - \ddot{\theta} cos\theta)}{M + m} \quad (1)$$

$$\ddot{\theta} = \frac{(M + m)gsin\theta - cos\theta(F + ml\theta^2 sin\theta)}{\frac{4}{3}(M + m)l - mlcos^2\theta}, \quad (2)$$

où M est la masse du chariot, m est la masse du pondule, l est la longueur du pivot and F est la force appliquée au chariot.



Le Cart-Pole dans le contexte de contrôle optimal II

- En effectuant une linéarisation du système dynamique au voisinage du point d'équilibre supérieur, on obtient un système linéaire qui, dans le cas discret, s'écrit :

$$x_{k+1} = Ax_k + Bu_k$$

où

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & a & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & a & 0 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 0 \\ \frac{1}{m+M} \\ 0 \\ b \end{pmatrix}$$

avec $a = \dfrac{g}{l * (\frac{4}{3} - \frac{m}{m+M})}$ and $b = \dfrac{-1}{l * (\frac{4}{3} - \frac{m}{m+M})}$. et une foinction coût donnée par :

$$J = \int_0^{\infty} \left(x^T Q x + u^T R u\right) dt$$

où Q et R sont des matrices de poids à préciser.

Le Cart-Pole dans le contexte de contrôle optimal III

- Le modèle est ainsi défini par les matrices A et B.
- Le feedback control qui minimise la fonction coût est donné par : $u = -Kx$, où $K = R^{-1}(B^T P)$ et P étant la solution de l'équation de Ricatti.
- Notons que dans le cas du Cart-Pole, le feed back u corresponf à l'intensité de la force F avec laquelle doit être poussé le chariot.