Le contrôle d'un système dynamique vu sous le prisme du reinforcement learning

Ghazi Bel Mufti

UR Analysis and Control of PDEs, Faculté des Sciences de Monastir
Ecole Supérieure de la Statistique et de l'Analyse de l'Information

CIP - Monastir, 9-11 mai 2022

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.

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.

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

Machine learning

- **Supervised Learning (apprentissage supervisé)**

OBJECTIF	Apprendre une fonction de prédiction à partir de données annotées
DONNÉES	Données étiquetées

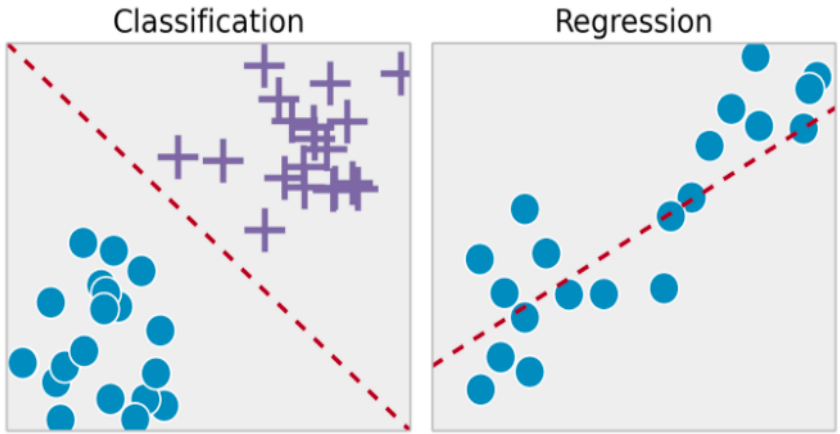
Machine learning

- **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 ».

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 (RL)

Quelques définitions

- **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.

Quelques définitions (suite)

- **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

Reinforcement learning (RL)

Quelques définitions (suite)

- **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.

Reinforcement learning (RL)

Exemples d'applications...

- **Un robot** : 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.

Reinforcement learning (RL)

Exemples d'applications...

- **Un robot** : 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.
- **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.

Reinforcement learning (RL)

Exemples d'applications...

- **Un robot** : 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.
- **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.
- **En 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.

Reinforcement learning (RL)

Exemples d'applications...

- **Un robot** : 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.
- **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.
- **En 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.
- **Le trafic routier**...

Reinforcement learning (RL)

Exemples d'applications...

- **Un robot** : 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.
- **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.
- **En 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.
- **Le trafic routier**...

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 $S \to A$ qui associe à chaque état s, une action $\pi(s)$ à effectuer.

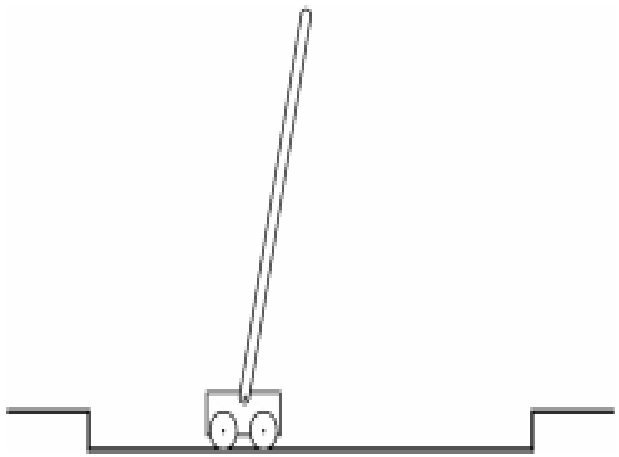
Exemple 1 Un robot aspirateur dont la récompense est le volume de poussière ramassée en 30 mn. Sa politique consiste à avancer, chaque seconde, avec une probabilité p ou de tourner aléatoirement vers la gauche ou la droite avec une probabilité $1 - p$.

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 $S \to A$ 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.

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 :

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

  - $\mathcal{R} : S \times A \to \mathbb{R}$, la récompense espérée quand on exécute une action a en un état s :

$$\mathcal{R}(s, a) = \mathbb{E}(r_{t+1} | s_t = s, a_t = a)$$

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 :

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

  - $\mathcal{R} : S \times A \to \mathbb{R}$, la récompense espérée quand on exécute une action a en un état s :

$$\mathcal{R}(s, a) = \mathbb{E}(r_{t+1} | s_t = s, a_t = a)$$

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 valeur d'un état-action 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 :

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

- Si l'agent agit de façon optimale, alors la valeur optimale de l'état courant est égale à la récompense espérée 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 valeur état-action 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 :`

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

Choix des actions : Exploitation/Exploration

- Un agent apprenant est sujet au compromis entre :
  - *l'exploitation* : refaire des actions, dont il sait qu'elles vont lui donner de bonnes récompenses,
  - *l'exploration* : essayer de nouvelles actions, pour apprendre de nouvelles choses.

- Sélection d'action $\epsilon - greedy$ où $\epsilon \in [0, 1]$:

  (a) tirer uniformément un nombre p au hasard dans l'intervalle [0,1] :
  (b) si $p < \epsilon$, tirer une action a_t dans A (*exploration*).
  (c) si $p \geq \epsilon$, agir de façon gourmande (*greedy*), en choisissant l'action a_t dans A ayant la valeur Q la plus élevée (*exploitation*).

Q-learning I

- Les problèmes d'apprentissage par renforcement avec des actions discrètes peuvent souvent être modélisés à l'aide des PDM, mais l'agent n'a initialement aucune idée des probabilités de transitions (i.e. les $T(s, a, a')$) ni sur les récompenses (i.e. les $R(s, a, s')$).

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

- L'algorithme du Q-learning suppose justement que l'agent connaît initialement uniquement les états et les actions possibles et rien de plus.
- Il doit tester au moins une fois chaque état et chaque transition pour connaître les récompenses.
- Il doit le faire à plusieurs reprises s'il veut avoir une estimation raisonnable des probabilités de transition.

Q-learning II

- On est à l'état s_k, on effectue l'action a_k, on passe à l'état s_{k+1} et on reçoit la récompense r_k.
- L'algorithme du Q-learning, comme l'algorithme d'apprentissage par différence temporelle (Time Difference) présente des similitudes avec la descente de gradient stochastique lors de la mise à jour de la Q-table :

$$Q(s_k, a_k)_{new} \leftarrow Q(s_k, a_k)_{old} + \alpha[Q_{target} - Q(s_k, a_k)_{old}]$$

- Dans le cas du Q-learning, Q_{target} est donnée par l'équation de Bellman :

$$Q_{target} = r_k + \gamma \max_{a' \in A(s_{k+1})} Q_{old}(s_{k+1}, a')$$

Remarque : A l'instar de la descente de gradient, l'algorithme ne peut converger que si l'en réduit progressivement le taux d'apprentissage α.

Q-learning III

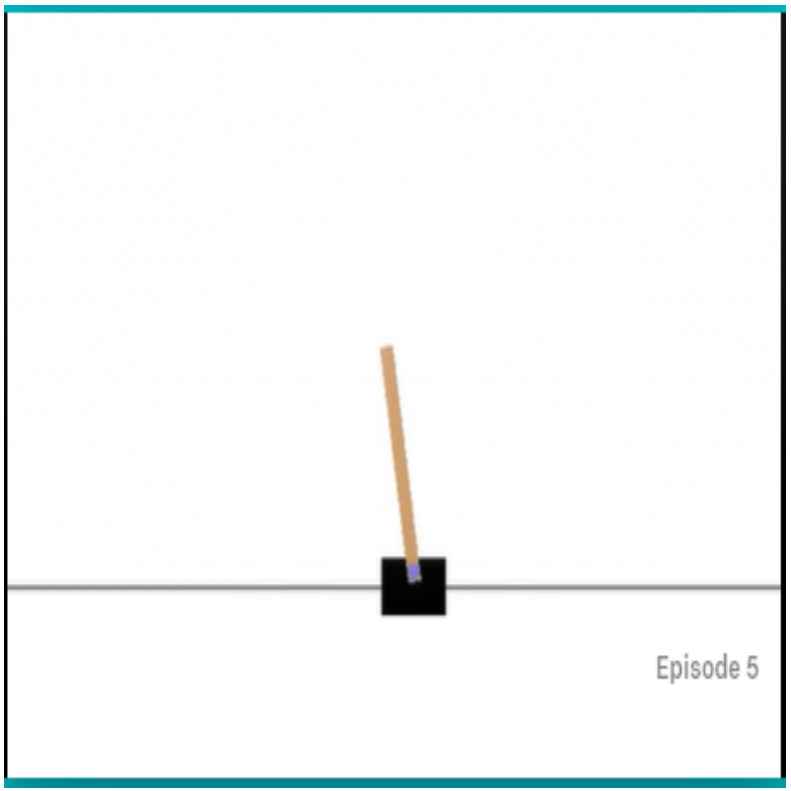
Algorithm 1 Q-learning

Input: espace (S, A), ϵ_{init}, α_{init} , γ
Output: Tableau Q de fonction valeur état-action

initialiser le tableau Q arbitrairement
boucle
initialiser l'état initial s de l'épisode
répéter
$a \leftarrow \pi_Q^{\epsilon}(s)$ (ϵ-greedy)
émettre a ; observer r et s'
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a' \in A(s')} Q(s',a') - Q(s,a)]$
$s \leftarrow s'$
jusqu'à s terminal
fin boucle
mettre à jour α et ϵ

Environnement CartPole-v1 de Openai Gym sous Python I

- Une tige est attachée à un chariot, qui se déplace le long d'une piste.
- Le système est contrôlé en appliquant une force positive ou négative au chariot.
- Le pendule démarre debout, et le but est de l'empêcher de tomber.
- Une récompense de $+1$ est fournie pour chaque pas de temps pendant lequel le poteau reste debout.
- L'épisode se termine lorsque le pôle est à plus de 15 degrés de la verticale ou que le chariot se déplace à plus de 2,4 unités du centre ou encore, que le chariot atteint les 500 pas.



https://gym.openai.com/envs/CartPole-v1/

Environnement CartPole-v1 de Openai Gym sous Python II

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Velocity At Tip	-Inf	Inf

Actions

Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

Starting State

All observations are assigned a uniform random value between ±0.05.

Q-table de l'environnement Cartpole-v1 discrétisé I

$$Q(s_t, a_t)_{new} \leftarrow Q(s_t, a_t)_{old} + \alpha[r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)_{old}]$$

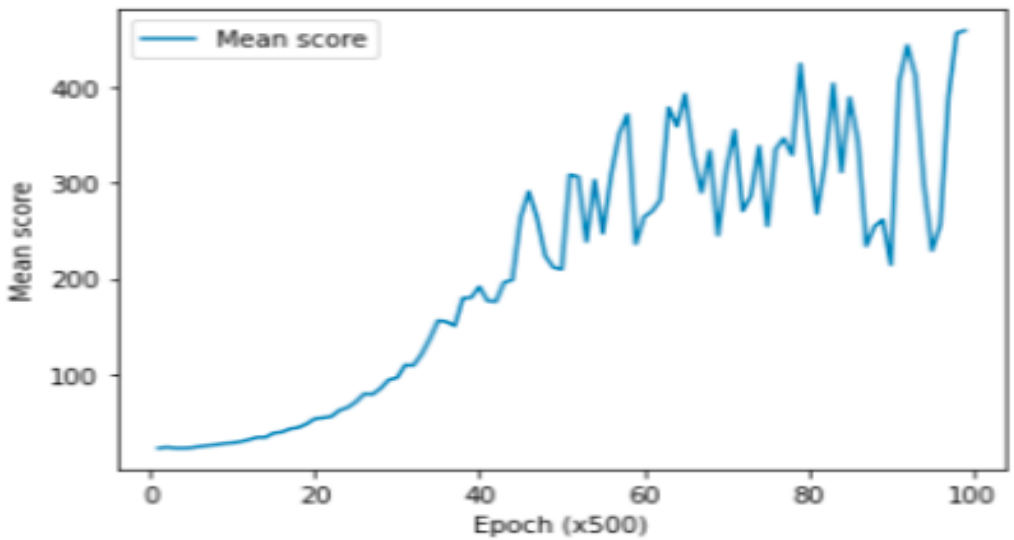
1. **Discrétisation des états**
2. **Paramètres**
   - Nombre d'épisodes (epochs) $= 50000$
   - Learning rate : α-start $= 0.05$ et $\alpha = \alpha * 0.99$
   - Paramètre de rabais $\gamma = 0.98$
   - Epsilon : ϵ-start $= 1$ et ϵ-min $= 0.05$
3. **Initialisation de la Q-table**
4. **Mise à jour de la Q-table**
   1. Déterminer a_t selon la stratégie $\epsilon - greedy$
   2. Déduire s_{t+1} et r à la suite de l'action a_t
   3. Déteminer a_{t+1} comme l'action réalisant le max de la fonction Q pour l'état s_{t+1}
   4. Calcul de $Q(s_t, a_t)_{new}$: mettre à jour la Q-table

Output : Reinforcemnt learning sur Cartpole I

1. **Nombre de réussites par paquet de 500 époques et scores moyens**

Temps d'éxécution : environ 8 mn

```
Epoch 046000/050000 reussite:0368/0500 epsilon=0.0500 Mean score=443.7360 alpha=0.0198
Sauvegarde ...
Epoch 046500/050000 reussite:0294/0500 epsilon=0.0500 Mean score=411.0760 alpha=0.0196
Epoch 047000/050000 reussite:0121/0500 epsilon=0.0500 Mean score=299.6840 alpha=0.0194
Epoch 047500/050000 reussite:0022/0500 epsilon=0.0500 Mean score=229.1260 alpha=0.0192
Epoch 048000/050000 reussite:0049/0500 epsilon=0.0500 Mean score=256.0360 alpha=0.0191
Epoch 048500/050000 reussite:0282/0500 epsilon=0.0500 Mean score=390.3820 alpha=0.0189
Epoch 049000/050000 reussite:0395/0500 epsilon=0.0500 Mean score=456.5140 alpha=0.0187
Sauvegarde ...
Epoch 049500/050000 reussite:0397/0500 epsilon=0.0500 Mean score=458.9140 alpha=0.0185
Sauvegarde ...
```



Output : Reinforcemnt learning sur Cartpole II

2. Q-**table** [shape $= (42, 42, 42, 42, 2)$]

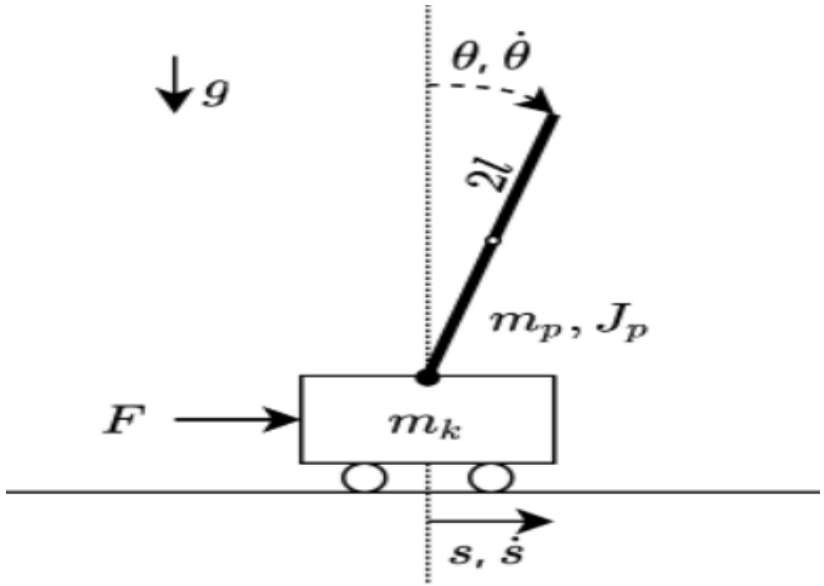
```
[[[[[ 7.69305806e-01  3.21338032e-01]
    [ 2.52563605e-01 -3.93269995e-01]
    [-3.72798709e-01  3.07857473e-01]
    ...
    [ 2.00437351e-01  8.07820774e-01]
    [ 4.08140863e-01  9.14546693e-01]
    [-2.89722083e-01 -4.18238407e-01]]
```

```
q_table[0][0][0][0]
```

```
array([0.76930581, 0.32133803])
```

Le Cart-Pole dans le contexte de contrôle optimal I

- Etat du Cart-Pole : la distance s du chariot, la vitesse $\dot{s}$ du chariot, l'angle du pôle θ et la vitesse angulaire du pôle $\dot{\theta}$.
- Les paramètres : m_p comme masse du poteau, m_k comme masse du chariot, la longueur l_p ainsi que le moment d'inertie J_p.
- Le système est **non-linéaire** et l'équilibre du système avec la tige en position verticale est **instable**.



Le Cart-Pole dans le contexte de contrôle optimal II

- En effectuant une linéarisation au voisinage du point d'équilibre supérieur de notre système, l'état $x = (s, \dot{s}, \theta, \dot{\theta})^T = (0, 0, 0, 0)^T$, nous obtenons un système linéaire qui s'écrit, dans le **cas continue** :

$$\dot{x} = Ax + Bu$$

$$A = \begin{pmatrix} 0, 1, 0, 0 \\ 0, 0, a, 0 \\ 0, 0, 0, 1 \\ 0, 0, a, 0 \end{pmatrix}$$

et

$$B = (0, \frac{1}{m_p + m_k}, 0, b)^T$$

où $a = \dfrac{g}{l_p * (\frac{4}{3} - \frac{m_p}{m_p + m_k})}$ et $b = \dfrac{-1}{l_p * (\frac{4}{3} - \frac{m_p}{m_p + m_k})}$.

Le Cart-Pole dans le contexte de contrôle optimal III

- La fonction coût :

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

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

- Le feedback control qui minimise la fonction coût est donné par :

$$u = -Kx$$

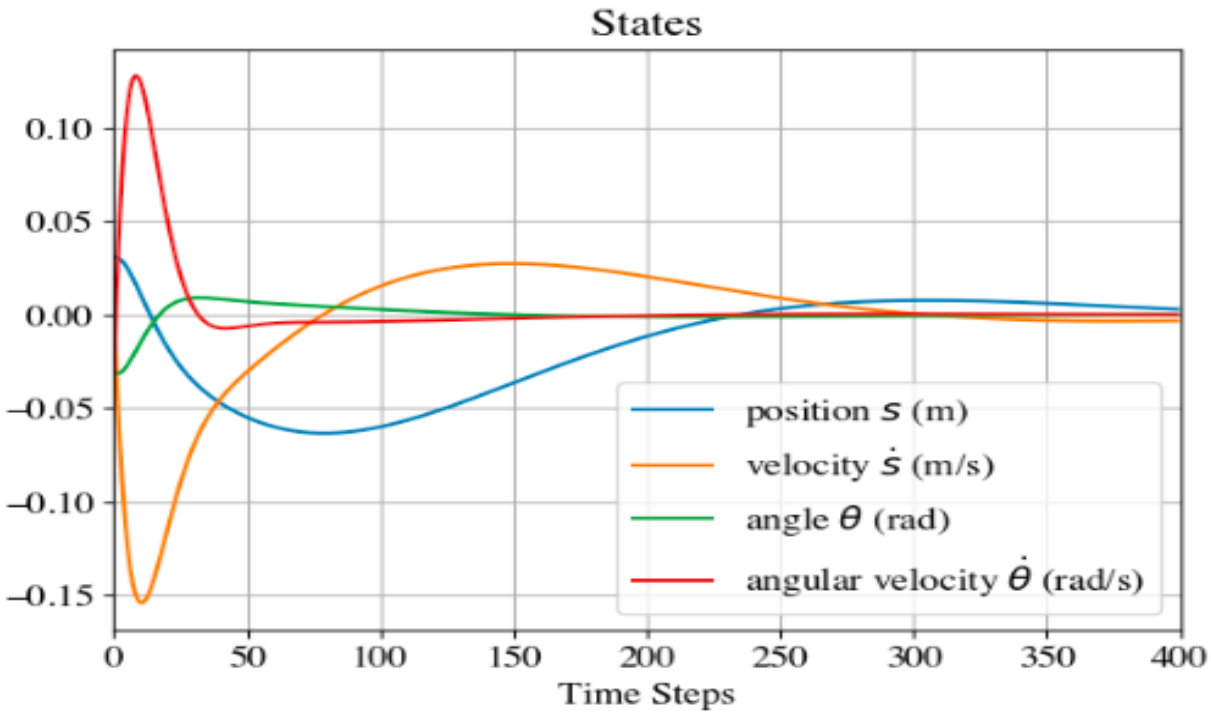
où K est donnée par $K = R^{-1}(B^T P)$, P étant la solution de l'équation de Ricatti.

Remarque : dans le **cas discret**, on a

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

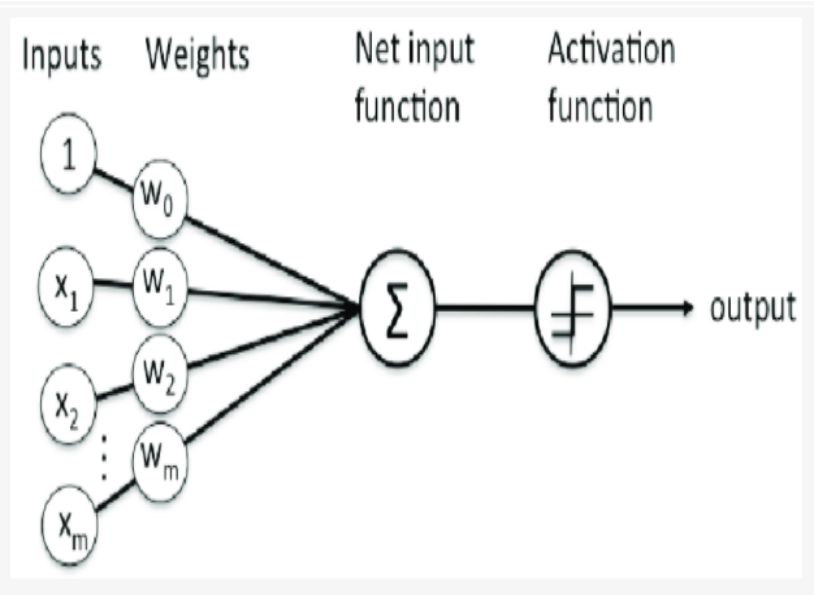
Output : Contrôle optimal sur Cartpole

- Dans l'environnement Cartpole, par défaut, la force est fixée.
- Dans notre cas : la magnitude $|Kx|$ est ajustée pour chaque pas de temps (via la fonction `env.env.force_mag` de l'environnement Cartpole-V1) et l'action (variable (O/1)) définit la direction.
- En partant d'un état initial tiré au voisinage de l'équilibre instable, on remarque qu'au bout de 150 pas la tige se stabilise au voisinage de la position $s = 0$:



Notions de deep learning : perceptron

- **Input : x**
- Calcul de la somme pondérée : $z = \mathbf{w}^T \mathbf{x}$
- Application de la fonction d'activation à z : $step(z)$
- **Output :** $step(z)$
- Fonction d'activation répondue : *fonction de heaviside*



Règle d'apprentissage du perceptron

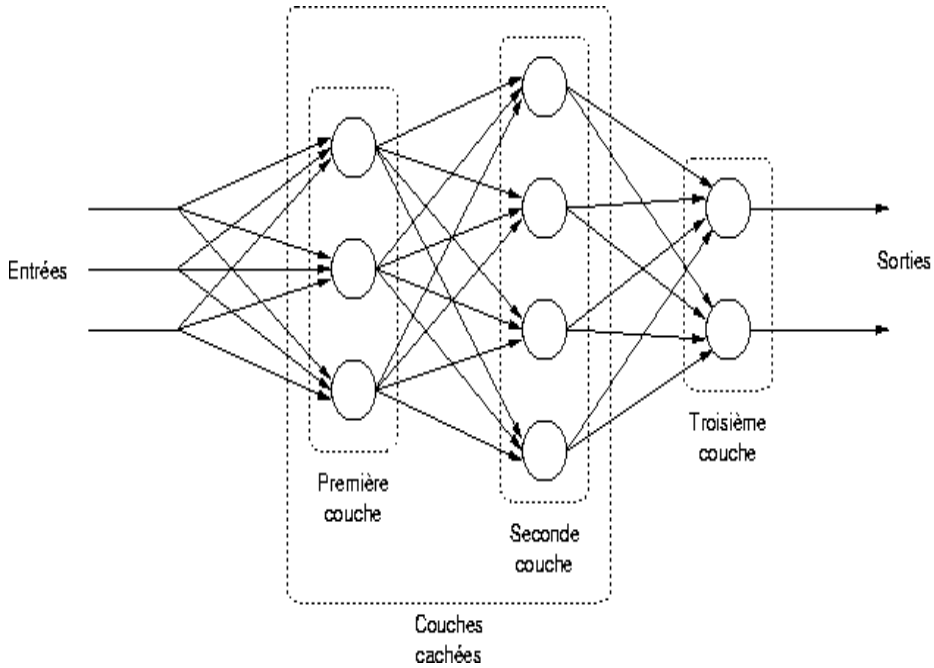
$$\boxed{w_{ij} \to w_{ij} + \eta(y_i - \hat{y}_i)}$$

- w_{ij} : poids de la connexion entre le $i^{ème}$ neuronne d'entrée et le $j^{ème}$ neuronne de sorie ;
- x_i : $i^{ème}$ valeur d'entrée de l'instance d'entraînement courante ;
- $\hat{y}_i$: sortie du $j^{ème}$ neuronne de sorie pour l'instance d'entraînement courante ;
- y_i : sortie **souhaitée** pour le $j^{ème}$ neuronne de sorie pour l'instance d'entraînement courante ;
- η : taux d'apprentissage.

Perceptron multicouche

- Le perceptron est un classifieur limité au cas de deux classes linéairement séparables.
- Cette limite est levée en empilant plusieurs perceptron $\rightarrow$ le perceptron multicouche (PMC).

- Un PMC est constitué
  - Couche d'entrée
  - Couches cachées
  - Couche de sorties
- **Sorties :**
  - Classification : classes binaires (0/1) ; probabilité d'appartenance à chaque classe par le biais de la fonction *softmax*
  - Régression



PMC : rétropropagation

Pseudo algorithme du PMC : rétropropagation

1. Fournir une instance d'entraînement au réseau
2. Calculer la sortie de chaque neuronne dans chaque couche consécutive
   $\rightarrow$ **Fonction d'activation :** ReLU, Sigmoïde, ...
3. Effectuer une prédiction (passe vers l'avant)
4. Mesurer l'erreur de sortie du réseau
   $\rightarrow$ **Fonction de perte :** Erreur quadratique moyenne (MSE), L1, ...
5. Déterminer dans quelle mesure chaque neuronne de la couche cachée précédente a contribué à l'erreur de chaque neuronne de la couche de sortie (passe vers l'arrière)
6. Ajustement des poids des connexions de manière à réduire l'erreur
   $\rightarrow$ **Optimiseurs :** Descente de gradient, Inertie, Adam,...

Deep Q-Learning

- Le Q-learning s'adapte mal aux PDM de grande, voire moyenne, taille, avec un nombre élevé d'états.
- Par exemple, lorsque l'environnement d'apprentissage par renforcement est un simulateur de conduite autonome où un état est définit par les images fournie par la caméra frontale du véhicule : deux images différentes représentent deux états différents.

Deep Q-Learning

- Le Q-learning s'adapte mal aux PDM de grande, voire moyenne, taille, avec un nombre élevé d'états.
- Par exemple, lorsque l'environnement d'apprentissage par renforcement est un simulateur de conduite autonome où un état est définit par les images fournie par la caméra frontale du véhicule : deux images différentes représentent deux états différents.

Mnih et al. [2013] propose l'utilisation des réseaux de neurones afin d'avoir les meilleurs estimations de la Q-valeur donnée par l'équation d'optimalité de Bellman.

- Un réseau de neurones utilisé pour estimer la Q *valeur* est appelé *Deep* Q−*network* (*DQN*).
- L'utilisation d'un *DQN* pour l'approximation de la *valeur* Q s'appelle *Deep Q-Learning*.

Deep Q-Learning

Pseudo algorithme

1. Transmettre l'état au *DQN*
2. Récupérer les valeurs de Q correspondant à chaque action $Q^{\pi}(s_{t+1}, a), a \in A$
3. Choisir l'action $(\epsilon - greedy)$
4. Calculer la valeur Q cible :

$$Q_{target} = r_t + \gamma \max_a Q^{\pi}(s_{t+1}, a)$$

5. Exécuter une itération d'entraînement du modèle à l'aide de tout algorithme de descente de gradient en minimisant l'erreur entre la valeur Q de l'action choisie à l'étape (3.) et la Q_{cible}

Références

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Daan Antonoglou, Ioannis Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013.

[2] Aurélien Géron and Hervé Soulard. Deep learning avec TensorFlow : mise en oeuvre et cas concrets. O'Reilly Media, 2017.

[3] Changsheng Hua. Reinforcement Learning Aided Performance Optimization of Feedback Control Systems. Springer, 2020.

[4] Richard Sutton and Andrew Barto. Reinforcement learning : an introduction. MIT Press, 2018.

[5] Paul Brunzema. Optimal Control with OpenAI Gym, 2021.
https ://towardsdatascience.com/comparing-optimal-control-and-reinforcement-learning-using-the-cart-pole-swing-up-openai-gym-772636bc48f4

[6] Prajwal Koirala. Feedback control of cart-pole, 2020.
https ://medium.com/@prajwalkoirala/feedback-control-of-cart-pole-937e3d3fdc51

[7] Steve Brunton. Q-Learning : Model Free Reinforcement Learning and Temporal Difference Learning.
https ://www.youtube.com/watch ?v=0iqz4tcKN58t=33sab_channel=SteveBrunton