APPRENTISSAGE PAR RENFORCEMENT

L'intelligence artificielle dans le monde de l'entreprise

Résoudre les problèmes réels de l'entreprise avec des solutions d'IA

Thème :
Optimisation des processus de flux dans un entrepôt e-commerce

Par **Roland MONDJEHI et Aristote MUTOMBO**

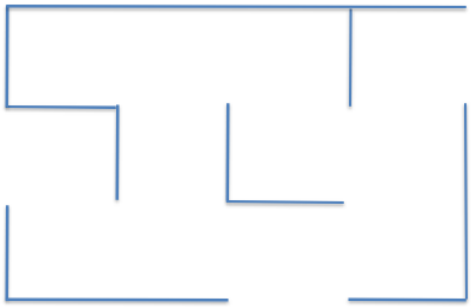
Sous la responsabilité de **Bertrand Koebel**

Introduction

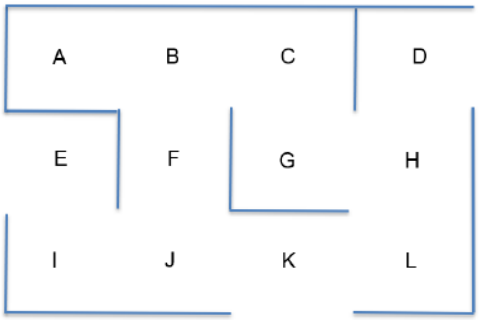
Cet exercice va de pair avec notre cours de renforcement learning. La problématique que nous avons choisie est d'optimiser des flux dans un entrepôt e-commerce. Et pour faire cela, nous commencerons par exposer le problème que nous allons résoudre et nous construirons à partir de zéro l'environnement de notre travail. Ensuite, nous définirions non seulement les concepts utilisés, mais aussi tous les détails mathématiques du modèle d'IA qui résoudra notre étude de cas. Après cela, nous implémenterons la solution d'IA en Python et nous entrerons en production.

Optimisation des flux dans un entrepôt e-commerce
Problème à résoudre

Le problème à résoudre est d'optimiser les flux dans l'entrepôt suivant :



Nous supposons que l'entrepôt appartient à une entreprise e-commerce qui vend des produits à une clientèle variée. Les produits sont stockés à l'intérieur c'est à dire dans 12 emplacements différents d'un magasin, identifiés par les lettres suivantes de **A** à **L** :



Alors que les commandes sont effectuées par les clients en ligne, un robot autonome se déplace dans l'entrepôt afin de collecter les produits pour les livraisons ultérieures. Voici à quoi le robot peut ressembler :

Les 12 emplacements sont tous reliés à un système informatique qui indexe en temps réel les priorités de collecte des produits pour ces 12 emplacements. Par exemple, à un temps donné ***t***, il donnera le classement suivant :

Niveau de priorité	Emplacement
1	G
2	K
3	L
4	J
5	A
6	I
7	H
8	C
9	B
10	D
11	F
12	E

On suppose que l'emplacement **G** a la priorité 1 (priorité absolue), car il contient un produit qui doit être collecté et livré immédiatement. Le choix de l'emplacement est aléatoire, ce qui veut dire que nous pouvons choisir n'importe quel point de l'entrepôt.
Notre robot d'entrepôt autonome doit se déplacer vers l'emplacement **G** par l'itinéraire le plus court selon l'endroit où il se trouve.

Notre objectif est de développer une IA indiquant l'itinéraire le plus court où que se trouve le robot. Toutefois, les emplacements **K** et **L** figurent parmi les trois premières priorités.
Par conséquent, nous voulons intégrer une option pour passer par des emplacements intermédiaires avant d'atteindre l'emplacement final.
Ainsi, nous dirons simplement par exemple que l'emplacement **G** est la priorité absolue parce que l'un des clients premium de l'entreprise a passé une commande urgente d'un produit stocké à l'emplacement **G** qui doit donc être livré le plus rapidement possible.
Enfin, notre mission est de créer une IA qui prend toujours l'itinéraire le plus court vers l'emplacement le plus prioritaire quel que soit le point de départ, et qui peut passer par un emplacement intermédiaire qui est dans les 3 priorités principales.

Environnement à définir

Lors de la création d'une IA, la première chose que nous devons toujours faire est de définir l'environnement comme nous l'avons vu dans le cours, surtout pour faire de l'apprentissage par renforcement.

Définir les états

Commençons par les états. L'état d'entrée est simplement l'emplacement où se trouve notre robot d'entrepôt autonome à un temps défini ***t***. Cependant, puisque nous implémenterons notre IA avec des équations, nous encoderons les noms d'emplacement **(A, B, C,...)** avec des numéros d'index selon le schéma suivant :

Emplacement	état
A	0
B	1
C	2
D	3
E	4
F	5
G	6
H	7
I	8
J	9
K	10
L	11

Il y a une raison spécifique pour laquelle nous encodons les états avec des index de 0 à 11 au lieu d'autres nombres entiers. La raison est que nous travaillons avec des matrices. Une matrice de récompenses et une matrice de Q-Values. Chaque ligne/colonne de ces matrices correspond à un emplacement spécifique. Par exemple la *1ère ligne* de chaque matrice qui a *l'index 0* correspond à l'emplacement **A**, la *2ème ligne/colonne* qui a *l'index 1* correspond à l'emplacement **B** et ainsi de suite. Mais nous reviendrons sur l'importance des matrices en détails.

Définir les actions

Les actions sont simplement les mouvements potentiels du robot pour aller d'un emplacement à un autre. Par exemple, disons que le robot est à l'emplacement **J**, les actions possibles à exécuter sont **I**, **F** ou **K**. Et comme nous allons travailler avec des équations mathématiques, nous allons encoder ces actions avec les mêmes index que pour les états. Ainsi, en suivant notre exemple où le robot se trouve à l'emplacement **J** à un temps précis, les actions possibles du robot sont, d'après notre précédent schéma : 5, 8 et 10. En effet, *l'index 5* correspond à **F**, l'index 8 correspond à **I** et *l'index 10* correspond à **K**. Par conséquent, la liste totale des actions que l'IA peut exécuter globalement est la suivante :

Actions = {0,1,2,3,4,5,6,7,8,9,10,11}

Évidemment, quand le robot est dans un emplacement précis, il y a certaines actions que l'IA ne peut pas exécuter. Suivant l'exemple précédent, si le robot se trouve à l'emplacement **J**, il peut exécuter les actions 5, 8 et 10, mais pas les autres. Nous veillerons à attribuer une récompense de 0 aux actions qu'il ne peut pas exécuter et une récompense de 1 aux actions qu'il peut exécuter. Cela nous amène aux récompenses.

Définir les récompenses.

La dernière chose qui reste pour créer notre environnement est de définir un système de récompenses. Plus précisément, une fonction de récompense **R** qui prend comme entrées un état ***s*** et une action ***a***, et génère une récompense numérique que l'IA obtiendra en exécutant l'action a dans l'état ***s*** :

$$\mathbf{R : (état, action)} \mapsto r \in \mathbb{R}$$

Alors, comment concevoir une telle fonction pour notre étude de cas ? Ici, c'est simple. Puisqu'il y a un nombre discret et fini d'états *(index 0 à 11)* ainsi qu'un nombre discret et fini d'actions (mêmes *index 0 à 11*), la meilleure façon de concevoir notre fonction de récompense **R** est de faire une matrice. Notre fonction de récompense sera exactement une matrice de 12 lignes et 12 colonnes dont les lignes correspondent aux états et les colonnes correspondent aux actions. Ainsi, dans notre fonction $"\ R : (s, a) \mapsto r \in \mathbb{R}"$, ***s*** sera l'index de ligne de la matrice, ***a*** sera l'index de colonne de la matrice et ***r*** sera la cellule d'index ***(s, a)*** dans la matrice.

Par conséquent, pour définir notre fonction de récompense, nous devons simplement remplir cette matrice avec les récompenses numériques. Comme indiqué ci-dessus, nous devons d'abord attribuer pour chacun des 12 emplacements une récompense : 0 aux actions que le robot ne peut pas exécuter et 1 aux actions que le robot peut exécuter. En faisant cela pour

chacun des 12 emplacements, nous obtiendrons une matrice de récompenses. Nous allons la créer étape par étape en commençant par l'emplacement **A**.

Lorsque le robot se trouve à l'emplacement **A**, il ne peut se rendre qu'à l'emplacement **B**. Puisque l'emplacement **A** a pour *index 0* (*1ère ligne* de la matrice) et l'emplacement **B** a pour *index 1* (*2ème colonne* de la matrice), la *1ère ligne* de la matrice de récompenses aura un 1 dans la *2ème colonne* et un 0 dans toutes les autres comme indique le schéma ci-dessous :

Définition des récompenses	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	0	0	0	0	0	0	0	0	0	0
B												
C												
D												
E												
F												
G												
H												
I												
J												
K												
L												

Passons maintenant à l'emplacement **B**. Lorsque le robot se trouve à l'emplacement B, il ne peut se rendre qu'à trois emplacements différents : **A**, **C** et **F**. Puisque **B** a pour *index 1 (2ème ligne)* et que **A**, **C**, **F** ont pour index respectifs 0, 2, 5 (*1ère, 3ème et 6ème colonne)*, alors la *2ème ligne* de la matrice de récompenses aura 1 sur les colonnes 1, 3, 6 et 0 sur toutes les autres colonnes. Ainsi, nous obtenons :

Définition des récompenses	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	0	0	0	0	0	0	0	0	0	0
B	1	0	1	0	0	1	0	0	0	0	0	0
C												
D												
E												
F												
G												
H												
I												
J												
K												
L												

En faisant ainsi pour tous les autres emplacements, nous obtenons notre matrice de récompenses finale :

Définition des récompenses	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	0	0	0	0	0	0	0	0	0	0
B	1	0	1	0	0	1	0	0	0	0	0	0
C	0	1	0	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	0	0	1	0	0	0	0
E	0	0	0	0	0	0	0	0	1	0	0	0
F	0	1	0	0	0	0	0	0	0	1	0	0
G	0	0	1	0	0	0	0	1	0	0	0	0
H	0	0	0	1	0	0	1	0	0	0	0	1
I	0	0	0	0	1	0	0	0	0	1	0	0
J	0	0	0	0	0	1	0	0	1	0	1	0
K	0	0	0	0	0	0	0	0	0	1	0	1
L	0	0	0	0	0	0	0	1	0	0	1	0

Il nous reste l'attribution des récompenses élevées aux emplacements prioritaires par le biais d'un système informatique qui restitue les priorités de collecte des produits pour chacun des 12 emplacements. Par conséquent, puisque l'emplacement **G** est la priorité absolue, le système informatique mettra à jour la matrice de récompenses en attribuant une récompense élevée dans la cellule **(G, G)** :

Définition des récompenses

	A	B	C	D	E	F	G	H	I	J	K	L
A	0	1	0	0	0	0	0	0	0	0	0	0
B	1	0	1	0	0	1	0	0	0	0	0	0
C	0	1	0	0	0	0	1	0	0	0	0	0
D	0	0	0	0	0	0	0	1	0	0	0	0
E	0	0	0	0	0	0	0	0	1	0	0	0
F	0	1	0	0	0	0	0	0	0	1	0	0
G	0	0	1	0	0	0	1000	1	0	0	0	0
H	0	0	0	1	0	0	1	0	0	0	0	1
I	0	0	0	0	1	0	0	0	0	1	0	0
J	0	0	0	0	0	1	0	0	1	0	1	0
K	0	0	0	0	0	0	0	0	0	1	0	1
L	0	0	0	0	0	0	0	1	0	0	1	0

C'est ainsi que le système de récompenses fonctionne avec le Q-Learning. Nous attribuons la récompense la plus élevée (ici 1000) à l'emplacement de priorité supérieure **G**. Ensuite, nous verrons dans la suite comment attribuer une récompense élevée inférieure au deuxième emplacement de priorité supérieure **K** pour faire passer notre robot par cet emplacement intermédiaire, optimisant ainsi les flux d'entrepôt, ce qui est normal.

Solutions d'IA

La solution d'IA qui résoudra le problème décrit ci-dessus est un modèle Q-Learning. Puisque ce dernier est basé sur les processus décisionnels de **Markov (MDPs).** Nous commencerons par une définition, puis nous passerons à l'intuition et aux détails mathématiques qui se cache derrière le modèle Q-Learning.

Processus décisionnels de Markov

Un processus décisionnel de Markov est un tuple **(S, A, T, R)** où :

- S est l'ensemble des différents états. Par conséquent, dans notre étude de cas :

$$\mathbf{S = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}}$$

- **A** est l'ensemble des actions qui peuvent être exécutées à chaque temps t. Ainsi :

$$\mathbf{A = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}}$$

- **T** s'appelle la règle de transition :

$$T : (s_t \in S, s_{t+1} \in S, a_t \in A) \mapsto \mathbb{P}(s_{t+1} | s_t, a_t)$$

où $\mathbb{P}(s_{t+1} | s_t, a_t)$ est la probabilité d'atteindre l'état futur s_{t+1} en exécutant l'action a_t dans l'état s_t. Donc **T** est la distribution de probabilité des états futurs au temps $t + 1$ étant donné l'état actuel et l'action exécutée au temps t.
Par conséquent, nous pouvons prévoir l'état futur s_{t+1} en faisant un tirage aléatoire dans cette distribution

$$T : \ s_{t+1} \sim T(s_t, \ldots, a_t)$$

Dans notre étude de cas, nous verrons à travers une application que cette distribution **T** de notre IA sera simplement la distribution uniforme, ce qui est un choix classique de distribution qui fonctionne très bien en faisant du Q-Learning.

- **R** est la fonction de récompense :

$$R : (s_t \in S, a_t \in A) \mapsto r_t \in \mathbb{R}$$

où r_t est la récompense obtenue après avoir exécuté l'action a_t dans l'état s_t.
Dans notre étude de cas, cette fonction de récompense est exactement la matrice que nous avons créée précédemment.
Après avoir défini les processus décisionnels de Markov, il est maintenant important de rappeler qu'il repose sur l'hypothèse suivante :
La probabilité de l'état futur s_{t+1} dépend uniquement de l'état actuel s_t et de l'action a_t, et ne dépend d'aucun des états et actions précédents. C'est à dire :

$$\mathbb{P}(s_{t+1} | s_0, a_0, s_1, a_1, \ldots, s_t, a_t) = \mathbb{P}(s_{t+1} | s_t, a_t)$$

En d'autres termes, un processus décisionnel de Markov n'a pas de mémoire.
Récapitulons maintenant ce qui se passe en ce qui concerne les processus décisionnels de Markov. À chaque temps t :

- ☞ L'IA observe l'état actuel s_t
- ☞ L'IA exécute l'action a_t
- ☞ L'IA reçoit la récompense $r_t = R(s_t, a_t)$
- ☞ L'IA passe à l'état suivant s_{t+1}

Donc maintenant la question est :
Comment l'IA sait-elle quelle action exécuter à chaque temps t ?

Pour répondre à cette question, nous devons introduire la fonction de stratégie.
La fonction de stratégie π est exactement la fonction qui, étant donné l'état s_t, indique l'action a_t :

$$\pi : s_t \in S \mapsto \ a_t \in A$$

Désignons par Π l'ensemble de toutes les fonctions de stratégie possibles. Le choix des meilleures actions à exécuter devient alors une question d'optimisation. En effet, il s'agit de trouver la stratégie optimale π^* qui maximise la récompense accumulée :

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \sum_{t \geq 0} R(s_t, \pi(s_t))$$

Par conséquent, la question est la suivante : Comment trouver cette stratégie optimale π^* ?
C'est là que le Q-Learning intervient.

Le Q-Learning

Avant d'entamer les détails du Q-Learning, nous devons expliquer le concept de la Q-value.
À chaque couple d'état et action ***(s, a)***, nous allons associer une valeur numérique ***Q(s, a)*** :

$$Q : (s_t \in S, a_t \in A) \mapsto Q(s, a) \in \mathbb{R}$$

Nous dirons que ***Q(s, a)*** est "la Q-value de l'action ***a*** exécutée dans l'état ***s***".
Pour comprendre l'objectif de cette "Q-Value", nous devons introduire la Différence temporelle.

Au début $t = 0$, toutes les Q-values sont initialisées à 0 : $\forall\ s_t \in S, a_t \in A, Q(s, a) = 0$
Supposons maintenant que nous soyons au temps ***t***, dans un certain état s_t. Nous exécutons une action aléatoire a_t, ce qui nous amène à l'état s_{t+1} et nous obtenons la récompense $R(s_t, a_t)$
Nous pouvons maintenant introduire la différence temporelle qui est au cœur du Q-Learning.
La Différence temporelle au temps t, désignée par $TD_t(s_t, a_t)$, est la différence entre :

- $R(s_t, a_t) + \gamma max(Q(s_{t+1}, a))$, soit la récompense $R(s_t; a_t)$ obtenue en exécutant l'action a_t dans l'état s_t, plus la Q-Value de la meilleure action exécutée dans l'état futur s_{t+1}, actualisée par un facteur $\gamma \in [0, 1]$, appelé facteur de réduction.
- et $Q(s_t, a_t)$, soit la Q-Value de l'action a_t exécutée dans l'état s_t, ce qui mène à :

$$TD_t(s_t, a_t) = R(s_t, a_t) + \gamma max(Q(s_{t+1}, a)) - Q(s_t, a_t))$$

Mais quel est le but exact de cette différence temporelle $TD_t(s_t, a_t)$?

$TD_t(s_t, a_t)$ est comme une récompense intrinsèque. L'IA apprendra les Q-values de manière à ce que :

- Si $TD_t(s_t, a_t)$ est élevée, l'IA obtient une "bonne surprise".
- Si $TD_t(s_t, a_t)$ est petite, l'IA obtient une "frustration".

Dans cette mesure, l'IA va itérer certaines mises à jour de Q-Values (par une équation appelée équation de Bellman) vers des différences temporelles plus élevées.
Par conséquent, dans la dernière étape de l'algorithme du Q-Learning, nous utilisons la différence temporelle pour renforcer les couple (état, action) du temps ***t-1*** au temps ***t***, selon l'équation suivante :

$$Q(s_t, a_t) = Q_{t-1}(s_t, a_t) + \alpha TD_t(s_t, a_t)$$

où $\alpha \in \mathbb{R}$ est le taux d'apprentissage qui détermine la vitesse à laquelle l'apprentissage des Q-Values se fait, d'où l'importance des mises à jour des Q-Values. Sa valeur est généralement un nombre réel choisi entre 0 et 1, comme 0.01, 0.05, 0.1 ou 0.5. Plus sa valeur est faible, plus les mises à jour des Q-Values seront petites et plus le Q-Learning sera long. Plus sa valeur est élevée, plus les mises à jour des Q-Values seront importantes et plus le Q-Learning sera rapide. Ainsi, les Q-Values mesurent l'accumulation de surprise ou de frustration associées au couple action et état (s_t, a_t). Dans le cas de surprise, l'IA est renforcée et dans le cas de la frustration, l'IA est affaiblie. C'est pourquoi nous voulons apprendre les Q-Values qui donneront à l'IA le maximum de "bonne surprise".

Par conséquent, la décision de l'action à exécuter dépend principalement de la Q-value $Q(s_t, a_t)$. Si l'action a_t exécutée dans l'état s_t est associée à une Q-Value élevée $Q(s_t, a_t)$, l'IA aura plus tendance à choisir a_t.

Par contre si l'action a_t exécutée dans l'état s_t est associée à une petite Q-value $Q(s_t, a_t)$, l'IA aura moins tendance à choisir a_t. Il y a plusieurs façons de choisir la meilleure action à exécuter. Lorsque nous sommes dans un état s_t, nous pourrions simplement prendre l'action a_t qui maximise la Q-Value $Q(s_t, a_t)$ par exemple :

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a))$$

L'algorithme Q-Learning complet

Résumons les différentes étapes de l'ensemble du processus de Q-Learning :
Initialisation :
Pour tous les couples d'états s et d'actions a, les Q-Values sont initialisées à 0 :

$$\forall\ s_t \in S,\ a_t \in A,\ Q_0(s, a) = 0$$

Nous commençons dans l'état initial s_0. Nous exécutons une action aléatoire pour atteindre le 1^{er} *état* s_1.
Ensuite, pour chaque $t \geq 1$, nous répéterons un nombre de fois (1000 fois dans notre code) ce qui suit :

1) Nous sélectionnons un état aléatoire s_t parmi nos 12 états possibles :

$$s_t = random(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)$$

2) Nous exécutons une action aléatoire a_t qui peut mener à un futur état possible, c.-à-d.

$$R(s_t, a_t) > 0 : a_t = random(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)\ s.t.\ R(s_t, a_t) > 0$$

3) Nous atteignons l'état suivant s_{t+1} et nous obtenons la récompense $R(s_t, a_t)$
4) Nous évaluons la différence temporelle $TD_t(s_t, a_t)$:

$$TD_t(s_t, a_t) = R(s_t, a_t) + \gamma max(Q(s_{t+1}, a)) - Q(s_t, a_t))$$

5) Nous mettons à jour la Q-value en appliquant l'équation de Bellman :

$$Q(s_t, a_t) = Q_{t-1}(s_t, a_t) + \alpha TD_t(s_t, a_t)$$

Implémentation

Dans cette partie nous allons coder tout le processus à l'aide de Python.
Le document notebook python contient les codes avec toutes les explications.
Ceci met fin à cette première partie