Apprentissage par renforcement

Lounès Meddahi (dépôt git : Github)

3 juin 2021

Résumé

Table des matières

Table des figures

1 Introduction

Ce document est le document reprenant les idées principales de l'implémentation des algorithmes d'apprentissage par renfocement. Les fichiers qui implémentes les algorithmes sont spécifiés dans chaque partie.

Pour mes premiers algorithmes, j'ai utilisé des dictionnaires afin de représenter mes fonctions de transitions et de retour pour pouvoir visualiser plus facilement ces notions dans mon code pour le début. Ces algorithmes peuvent être réadaptés afin d'utiliser des listes plutôt que des dictionnaire (la modification est rapide et simple). Utiliser des listes peut être plus rentable si on utilise beaucoup d'actions et d'états (car on n'aura pas besoin de stocker les clés).

2 Algorithmes

2.1 Algorithme n°0 : Calcul de la valeur

2.1.1 Idées de l'algorithme :

Ce premier algorithme a pour but de calculer la valeur d'une politique grâce à l'équation de Bellman :

$$V^{\pi}(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \tag{1}$$

Un algorithme simple pour faire cette tâche peut être :
-initialiser V(s) $\leftarrow 0, \forall s \in S, k \leftarrow 0$
-itérer : $V_{k+1}(s) = \sum_{a \in A} \pi$(s,a) $\sum_{s' \in S}$P(s,a,s')[R(s,a,s') + γ $V^{\pi}(s')$] ; k+1
-jusqu'à ce que l'écart entre V_k et V_{k-1} soit faible

2.1.2 Programme :

On peut retrouver cet algorithme dans le fichier Algo0.py.
Dans ce fichier on y trouve les fonctions :

- politique_taxi() qui produit la politique stochastique du taxi (sous forme d'un dictionnaire (Etat,{action :probabilité}))
- transition_taxi() qui produit la fonction de transition du taxi (sous forme d'un dictionnaire (Etat,(action,{Etat :probabilité}))
- retour_taxi() qui produit la fonction de retour du taxi (sous forme d'un dictionnaire (Etat,(action,{Etat :retour}))
- valeur(politique,actions,etats,transitions,retour,gamma) qui permet d'avoir la valeur associée à la politique passée en paramètre en utilisant l'environnement d'écrit en paramètre (actions , etats , transitions , retour). gamma est le facteur déprécié

2.1.3 Affichage :

A la fin du programme, un code a été ajouté afin de trouver les valeurs données dans le document source :
'A' : 87.43423712514684, 'B' : 98.56278129353218, 'C' : 87.39517337027189

On a donc bien trouvé que son revenu à long terme sera en moyenne de 87,43 s'il démarre dans l'état A, 98,6 s'il démarre dans l'état B et 87,40 s'il démarre dans l'état C.

2.2 Algorithme n°1 : itération sur les politiques

2.2.1 Idées de l'algorithme :

Cet algorithme a pour but de calculer la valeur ϵ-optimale d'un environnement grâce à des politiques aléatoires et la formule :

$$\sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s') \tag{2}$$

Pour ce fait, on générera des politiques aléatoires et garder la politique qui nous donne la meilleur valeur.

On trouvera une politique π telle que $\| V^{\pi}\text{-}V^{*} \|_{\infty} \leq \epsilon$.

2.2.2 Algorithme :

Algorithm 1 L'algorithme d'itération sur les politiques.

Require: un PDM : $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
Require: un seuil de précision ϵ
initialiser π_0 (aléatoirement ou autrement)
$k \leftarrow 0$
repeat
  initialiser V_0^{π} (aléatoirement ou autrement)
  $i \leftarrow 0$
  repeat
    for tout état $s \in \mathcal{S}$ **do**
      $V_{i+1}^{\pi_k}(s) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}(s, \pi_k(s), s')[\mathcal{R}(s, \pi(s), s') + \gamma V_i^{\pi_k}(s')]$
    end for
    $i \leftarrow i + 1$
  until $||V_i^{\pi_k} - V_{i-1}^{\pi_k}||_{\infty} \leq \epsilon \frac{(1-\gamma)}{2\gamma}$
  for tout état $s \in \mathcal{S}$ **do**
    $\pi_{k+1}(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s, a, s')[\mathcal{R}(s, a, s') + \gamma V^{\pi_k}(s')]$
  end for
  $k \leftarrow k + 1$
until $\pi_k = \pi_{k-1}$

FIGURE 1 – Algorithme 1

-

2.2.3 Programme :

Cet algorithme est disponible dans le fichier Algo1.py. Ce fichier réutilise les fonctions transition_taxi() et retour_taxi() du fichier Algo0.py. Dans ce fichier on y retrouve également les fonctions :

— random_pol(actions) qui renvoie une action aléatoires parmi les actions possibles
— IterationPolitiques(etats , actions , transitions , retour , gamma , epsilon) qui renvoie la politique ϵ-optimale par la méthode par itération sur les politiques

2.2.4 Affichage :

A la fin du programme, un code a été ajouté afin de trouver une politique 0.01-optimale.
On trouve comme politique (déterministe) :
'A' : 'a2', 'B' : 'a3', 'C' : 'a2'
Et les valeurs qu'on trouve pour cet politique sont :
'A' : 121.64862519185584, 'B' : 135.30142959222255, 'C' : 122.83205714451859
(Qui sont meilleurs que celles trouvés juste avant).
Cette politique détérministe peut-être vue comme une politique stochastique :
— politique['A'] = 'a1' :0 , 'a2' :1 , 'a3' :0
— politique['B'] = 'a1' :0 , 'a3' :1
— politique['C'] = 'a1' :0 , 'a2' :1 , 'a3' :0

2.3 Algorithme n°2 : itération sur la valeur

2.3.1 Idées de l'algorithme :

Cet algorithme a pour but de calculer la valeur ϵ-optimale d'un environnement grâce à l'équation d'optimalité de Bellman :

$$\max_{a \in A(s)} \sum_{s' \in S} P(s,a,s')[R(s,a,s') + \gamma V^{\pi}(s')] \tag{3}$$

2.3.2 Algorithme :

Algorithm 2 L'algorithme d'itération sur la valeur.

Require: un PDM : $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$
Require: un seuil de précision ϵ
initialiser $V_0 \leftarrow 0$
$k \leftarrow 0$
repeat
  for tout état $s \in \mathcal{S}$ **do**
    $V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V_k(s')]$
  end for
  $k \leftarrow k + 1$
until $||V_k - V_{k-1}||_{\infty} \leq \frac{\epsilon(1-\gamma)}{2\gamma}$
for tout état $s \in \mathcal{S}$ **do**
  $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s,a,s')[\mathcal{R}(s,a,s') + \gamma V(s')]$
end for

FIGURE 2 – Algorithme 2

-

2.3.3 Programme :

Cet algorithme est disponible dans le fichier Algo2.py. Ce fichier réutilise les fonctions transition_taxi() et retour_taxi() du fichier Algo0.py.

— IterationValeur(etats , actions , transitions , retour , gamma , epsilon) qui renvoie la politique ϵ-optimale par la méthode par itération sur les politiques

2.3.4 Affichage :

A la fin du programme, un code a été ajouté afin de trouver une politique 0.01-optimale.
On trouve comme politique (déterministe) :
'A' : 'a2', 'B' : 'a3', 'C' : 'a2'
Et les valeurs qu'on trouve pour cet politique sont :
'A' : 121.64862519185584, 'B' : 135.30142959222255, 'C' : 122.83205714451859
(Comme à la partie 2.2.4).