Apprentissage par renforcement

Sergio Peignier

Un BIM s'amuse à entrainer un rat *in silico* pour qu'il trouve le chemin vers sa nourriture dans un petit labyrinthe. Pour cela il va programmer un algorithme d'apprentissage par renforcement. Dans cet exercice le rat *in silico* correspond à un agent classique d'apprentissage par renforcement, le labyrinthe correspond à l'environnement de l'agent et un état de l'agent correspond à sa position dans le labyrinthe.

1 Apprentissage par renforcement passif (estimation directe de l'utilité)

Dans un état s l'agent réalise l'action $\pi(s)$, appelée politique de l'agent à l'état s. Le but de l'agent est d'optimiser la fonction d'utilité $U^{\pi}(s)$, c'est à dire la qualité de la politique qu'il réalise.

Dans ce premier exemple l'agent ne fait pas intervenir un modèle de transition $P(s|s', a)$, i.e., il ne connait pas la probabilité conditionnelle de transiter de l'état s vers l'état s' en réalisant l'action a. L'agent ne connait pas explicitement $R(s)$ la fonction de récompense associée à chaque état.

L'agent fait une série d'essais dans l'envirronement et mémorise la séquence des états qu'il a visité ainsi que les recompenses qu'il reçoit à chaque état. La fonction d'utilité associé à un état s correspond à la recompense totale espérée à partir de cet état:

$$U^{\pi}(s) = E(\sum_{t=0}^{\infty}(\gamma^t R(S_t)))$$

γ étant un paramètre du modèle, et S_t une variable aléatoire telle que $S_0 = s$, S_1 correspond au premier état rencontré après s etc ...

Chaque esssai constitue un échantillon qui permetra d'estimer la valeur de la fonction d'utilité. Pour cela on fait en sorte que l'agent puisse s'entrainer plusieurs fois sur l'environnement (effectue plusieurs séquences d'essais), puis à chaque entrainement on met à jour la fontion d'utilité.

Même si un ∞ intervient dans la formule, en pratique nous allons considérer l'existence d'état finaux qui mettent fin à la séquence d'essais en cours lorsque l'un d'eux est atteint par l'agent. Dans ce cas, l'état de l'agent correspond à sa position sur la grille.

Les classes et les structures suivantes sont données uniquement à titre indicatif et vous pouvez concevoir votre propre programme.

1.1 classe environnement

Utilisez ce labyrinthe comme base:

vide	vide	vide	nourriture :)
vide	obstacle	vide	piège :'(
vide (start)	vide	vide	vide

Cette classe peut avoir:

- Un agent
- Un labyrinthe en 2D (matrice). Le labyrinthe contient des cases vides, des obstacles, des cases contenant de la nouriture ou des pièges.
- Un dictionnaire qui associe une récompense à chaque type de case qui est accésible à l'agent ($\{vide : r_v = -0.04$, $nourriture : r_n = 1$, $piege : r_p = -1\}$)
- Une liste des types de clases qui correspondent à un état final ($[nourriture, piege]$)
- La position de l'agent dans le labyrinthe.
- Une fonction qui renvoie la recompense associée à la position donnée à la fonction en argument (par défaut celle sur laquelle se trouve l'agent)
- Une fonction qui renvoie True si la position donnée en argument (par défaut celle sur laquelle se trouve l'agent) est un état final
- Une fonction qui renvoie la position de l'agent sous forme de chaine de character (e.g. "2-1" si l'agent est dans la case $x = 2$ et $y = 1$), cette chaine de caractéres correspond à l'état de l'agent.
- Une fonction qui évalue l'action de l'agent: Un agent peut réaliser 4 actions de déplacement (N,S,E et O), cette fonction met à jour la position de l'agent si la nouvelle position correspond à une position viable (si ce n'est pas un obstacle et si on ne sort pas du labyrinthe) et renvoie un tuple contenant la description sous forme de chaine de characteres de la position de l'agent après déplacement (ou la même position si l'agent n'a pas pu se déplacer) ainsi que la recompense associé à la position en question et un booléan qui indique si celle-ci est associé ou non à un état final.

1.2 classe Agent

Cette classe peut avoir:

- Un état (par exemple une chaine de caractères du type "PositionX-PositionY")
- Un dictionnaire contenant le modèle du monde de l'agent (vers quel état peut on passer, sachant qu'on réalise telle action et sachant qu'on est dans tel état à 'instant donné).
- Un paramètre γ.
- Un paramètre t qui compte le nombre d'essais faits.

- Une liste qui garde en mémoire les états rencontrés lors d'une séquence d'essais.
- Une liste qui garde en mémoire les recompenses rencontrés lors d'une séquence d'essais.
- Un dictionnaire de listes contenant la liste des états qui ont pu être visités à partir de chaque état qui figure comme clef de dictionnaire. On appélera ce dictionnaire le modèle du monde de l'individu.
- Un dictionnaire contenant pour chaque état s (clef du dictionnaire) l'utilitée totale associé à l'état en question (calculée au cours de différentes séquences d'entrainement). Ce dictionnaire sera appellé dictionnaire d'utilité.
- Un dictionnaire contenant pour chaque état s (clef du dictionnaire) le nombre de fois pour lesquels cet état a déjà été rencontré. Ce dictionnaire sera appellé dictionnaire de fréquences.
- Une liste contenant les actions que l'agent peut réaliser (4 déplacements posssibles: N, S, E, O).
- Une fonction qui garde en mémoire l'état et la recompense envoyés en paramètres.
- Une fonction qui met à jour le modèle du monde de l'individu (si un nouvel état est rencontré), mais aussi les clefs des dictionnaires d'utilité et de fréquences.
- Une fonction d'action qui choisi aléatoirement une action parmis les actions possibles.
- Une fonction qui met à jour l'utilité globale de chaque état.
- Une fonction qui fait le "reset" de la liste des états rencontrés et des recompenses pour préparer une nouvelle séquence d'entrainement.
- Une fonction run qui prend en entrée l'environnement et qui initialise l'état de l'agent avec la position de l'agent dans le labyrinthe (ainsi que la premiére récompense), qui met à jour les dictionnaires et qui ensuite répéte en boucle les événement suivant: l'agent agit, l'environnment lui renvoie son nouvelle état, l'agent mémorise son nouvel état ainsi que sa recompense et met à jour les dictionnaires si nécéssaire. La boucle s'arréte si l'agent rencontre un état terminal. Une fois la boucle arrétée, on met à jour l'utilité globale associé à chaque état.
- Une fonction qui permet à l'agent de construire une politique post-entrainement à partir de son état initial: À chaque état s l'agent effectue l'action qui l'emmène vers l'état accessible s' avec la plus grande utilité. L'agent s'arrête lorsqu'il a atteint un état optimal.
- Une fonction qui garde l'estimation des utilitées pour chaque point à chaque nouvel essai. Cette fonction doit être appellée à chaque fois qu'un essai a été réalisé.
- Une fonction qui écrit le paysage d'utilité à un moment donné (utilité pour chaque case accessible du labyrinthe).

1.3 Questions

- Visualiser l'évolution de l'estimation des utilitées aux cours du temps (100 essais). Que remarquez vous?
- Visualiser l'utilité finale associée à chaque état.
- Quel est l'impact de γ? Quel est son rôle?
- Tester l'algorithme dans un monde plus grand (avec et sans obstacles) que constatez vous?
- Lancez le programme avec 3 graines différentes pour le générateur pseudo alétoire (i.e. random.seed(graine))) et sauvegarder ces résultats dans un dossier bien précis qui sera crée si nécéssaire (depuis python! regardez os.path , os.listdir, os.path.isfile, os.path.join , os.path.isdir, os.makedirs ...).

2 Apprentissage par renforcement Q-learning

On se propose maintenant de tester un autre algorithme d'apprentissage par renforcement: le Q-learning. Cette algorithme n'apprend pas les fonctions d'utilité mais considére plutôt la représentation de la valeur d'une action dans un certain état c'est à dire la fonction $Q(s, a)$ (intuitivement $Q(s, a)$ correspond à l'adéquation de l'application de l'action a dans l'état s). Le Q-learning est qualifié comme méthode sans modèle. En effet, cette méthode ne fait pas appel à un modèle de transition $P(s|s', a)$, c'est à dire aux probabilités conditionnelles de transiter d'un état à un autre par la réalisation de l'action a. L'équation de mise à jour de l'algorithme Q-learning lorsque l'agent passe d'un état s à un état s' en réalisant une action a est:

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R(s) + \gamma max_{a'}(Q(s', a')) - Q(s, a))$$

Ou γ et α sont deux paramètres de l'algorithme. Modifiez la classe Agent précédente pour faire une classe AgentQ. Il faut en particulier remplacer le dictionnaire d'utilités par une strucutre de données contenant les valeurs $Q(s, a)$ pour chaque couple (s,a) (dictionnaire de dictionnaires, dictionnaire de listes, liste de listes ...) ainsi que les fonctions associées ainsi que la fonction qui permet la construction s'une politique post-entrainement (celle-ci fait maintenant intervenir la fonction Q).

2.1 Questions

- Jouez avec l'algorithme et comparez les deux approches.
- Quel est le rôle de α et γ?

2.2 Questions élargissement (bonus)

- Pensez vous que ces algorithmes ont une application pratique?

- Pensez vous que ces algorithmes sont des modèles appropriés des systémes biologiques apparentés?