



alpha*ai*

L'ÉDUCATION À L'INTELLIGENCE ARTIFICIELLE

Programmation de l'algorithme Q-learning

1 Q-LEARNING

1.1 INTRODUCTION

L'algorithme du **Q-learning** est un algorithme d'**apprentissage par renforcement**. Cela signifie que l'IA va **explorer** en autonomie différentes stratégies, et recevoir des **récompenses** ou pénalités selon des modalités déterminées à l'avance. L'objectif de l'algorithme d'apprentissage est alors d'exploiter les données ainsi générées afin d'améliorer les prises de décision de l'IA, dans le but de maximiser les récompenses reçues à plus ou moins long terme.

Voici un résumé du vocabulaire et des notations utilisées en apprentissage par renforcement :

1. Le robot est un **agent apprenant**.
2. L'**état** dans lequel se trouve l'agent est noté s . Ce terme peut représenter diverses informations, comme sa position, son orientation, le niveau de ses batteries, etc. Dans le cas de AlphaAI, on appelle s l'ensemble des informations qui sont fournies par les capteurs actifs du robot (certains capteurs peuvent être désactivés).
3. L'**action** choisie par l'agent est notée a . Dans le cas de AlphaAI, il s'agit de l'un des mouvements disponibles pour le robot.
4. Après avoir effectué une action, l'agent reçoit une récompense r , qui est simplement une valeur numérique positive ou négative.
5. Le nouvel état dans lequel se trouve l'agent après avoir agi est noté s' .

L'algorithme d'apprentissage par renforcement appelé *Q-learning* s'applique uniquement à des situations où le nombre d'états possibles n et le nombre d'actions possibles m sont finis et de taille raisonnable. L'algorithme utilise un tableau de valeurs afin de prendre des décisions. Ce tableau est appelé en anglais *Q-table*, ce qui donne son nom à l'algorithme. Chaque ligne du tableau correspond à un état s et chaque colonne à une action a . La valeur de la cellule correspondant à l'état s et à l'action a est donc notée $Q(s, a)$. Voir figure 1.

États	Actions		
	a_1	$\dots$	a_m
s_1	$Q(s_1, a_1)$	$\dots$	$Q(s_1, a_m)$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_n	$Q(s_n, a_1)$	$\dots$	$Q(s_n, a_m)$

FIGURE 1 – Représentation de la table de *Q-learning*.

L'objectif de l'apprentissage va être de faire en sorte que la valeur $Q(s, a)$ corresponde le mieux possible à la valeur moyenne reçue en récompense lorsqu'on

effectue l'action a à partir de l'état s .

Pour cela, l'algorithme va mettre à jour une valeur de la table après chaque action à l'aide des formules suivantes :

$$target = (1 - \gamma)r + \gamma \max_a Q(s', a) \quad (1)$$

$$Q'(s, a) = (1 - \alpha)Q(s, a) + \alpha target \quad (2)$$

où :

- $Q'(s, a)$ représente la nouvelle valeur de la cellule.
- α est un paramètre compris entre 0 et 1, appelé **vitesse d'apprentissage**.
- γ est un second paramètre compris entre 0 et 1, appelé **facteur d'actualisation**.
- $\max_a Q(s', a)$ est la récompense maximale que l'on peut obtenir à partir de l'état s' , selon les évaluations actuelles de la Q-table.

L'objectif de ce TP est de programmer en python cet algorithme *Q-learning* et de l'appliquer à l'apprentissage de l'**évitement d'obstacle** des robots AlphAI.

1.2 CONFIGURATION DU LOGICIEL

1. Chargez la configuration *Apprentissage par renforcement - Bloqué vs. Mouvement*.
2. Dans l'onglet *IA*, vérifiez que le type d'apprentissage est **apprentissage par renforcement**, puis sélectionnez la valeur **code python** pour le paramètre algorithme. Créez un nouveau fichier et choisissez un nom et un emplacement appropriés pour l'enregistrer.
3. Avec ces paramètres, quel est le nombre n d'états possibles du robot ? Quel est le nombre m d'actions possibles ?
4. Combien la table de *Q-learning* devra-t-elle posséder de lignes et de colonnes ?

1.3 FONCTION INIT

1. Tout en haut de votre fichier, écrivez une ligne permettant d'importer les fonctions `uniform` et `randint` de la bibliothèque `random`.
2. En dessous de cette ligne, définir la variable `q_table` comme un tableau vide.
3. Exécutez plusieurs fois l'instruction `print(uniform(-1, 1))` et en déduire ce que renvoie la fonction `uniform`. Idem pour la fonction `randint`. Vous pouvez aussi consulter la documentation du [module random](#).
4. À l'intérieur de la fonction `init` et à l'aide de la fonction `uniform`, initialisez la variable `q_table` afin qu'elle contienne un tableau de `n_state` lignes et `n_action` colonnes contenant des nombres aléatoires entre -1 et 1. Attention

à bien utiliser le mot-clé `global` afin de modifier la variable globale `q_table` depuis l'intérieur de la fonction `init`.

5. Définir une fonction auxiliaire `print_table` qui permet d'afficher les valeurs contenues dans la variable `q_table` de la manière la plus lisible possible. Par exemple : une ligne pour chaque état `s`, et se limiter à 2 chiffres après la virgule. Appeler cette fonction à la fin de la fonction `init`.
6. Dans le logiciel AlphAI, cliquez plusieurs fois sur le bouton *Réinitialiser l'IA*. Cela a pour effet d'appeler la fonction `init` que vous venez de définir. Vérifiez que la table du *Q-learning* possède bien les dimensions voulues et contient des valeurs aléatoires entre -1 et 1 .

1.4 FONCTION TAKE_DECISION

1. Définir une fonction auxiliaire `argmax` qui prend en paramètre une liste et renvoie l'indice du maximum de cette liste.
2. Pour prendre une décision, l'algorithme du *Q-learning* consiste à choisir l'action qui permet d'obtenir la meilleure récompense moyenne à partir d'un état `s` donné. Les valeurs des récompenses moyennes potentielles sont estimées à partir de la variable `q_table`. À l'aide de la fonction `argmax`, définir la fonction `take_decision`.
3. Dans AlphAI, activez le mode autonome et vérifiez que l'action choisie correspond bien à ce qui est attendu, en comparant les valeurs contenues dans `q_table`.
4. Nous allons maintenant ajouter une certaine quantité d'**exploration** dans cette fonction de décision. Définissez une variable `exploration` contenant un nombre entre 0 et 1. À l'aide des fonctions `uniform` et `randint`, faire en sorte que l'action choisie par le robot soit aléatoire avec une probabilité égale à la valeur de la variable `exploration`.
5. Ajoutez un message `exploration !` lorsque le robot réalise une exploration aléatoire.
6. Dans AlphAI, activez de nouveau le mode autonome et vérifiez que la proportion d'actions d'exploration est conforme à la valeur choisie.

1.5 FONCTION LEARN

1. Dans la fonction `learn`, définir deux variables `alpha` (la vitesse d'apprentissage) et `gamma` (le facteur d'actualisation) et choisir des valeurs qui vous semblent adaptées pour ces deux paramètres.
2. Ajoutez une ligne permettant de mettre à jour la q-valeur du tableau grâce à la formule 2 (page 2).
3. Ajoutez une ligne afin d'afficher les valeurs contenues dans la table.
4. Dans AlphAI, lancez l'apprentissage en mode autonome, et vérifiez que les valeurs présentes dans la table convergent bien vers les valeurs attendues et que le robot est capable d'apprendre.

2 DEEP Q-LEARNING

Comme vu dans la partie 1, la principale limitation de l'algorithme du *Q-learning* est qu'il ne peut s'appliquer qu'à des situations où le nombre d'états possibles et le nombre d'actions possibles sont finis et de tailles raisonnables. Dans le cas contraire, la table de l'algorithme serait de taille trop importante, ce qui peut causer des problèmes en terme de quantité de mémoire, et de temps nécessaire pour apprendre.

L'algorithme de *deep Q-learning* a pour objectif de dépasser ces limitations en remplaçant la table du *Q-learning* par un réseau de neurones avec un unique neurone de sortie et des neurones d'entrée permettant de représenter à la fois l'état s et l'action a .

Voici comment fonctionne l'algorithme :

- Pour calculer la valeur $Q(s, a)$, on utilise les valeurs s et a comme valeurs d'entrée pour le réseau de neurones. On considère alors que sa valeur de sortie est une approximation de $Q(s, a)$. Ce qui ne sera en réalité correct que lorsque le réseau aura été entraîné, tout comme pour la table du *Q-learning*.
- Pour prendre une décision à partir de l'état s , on calcule ainsi les valeurs $Q(s, a)$ pour toutes les valeurs possibles de a . Puis on choisit l'action qui donne la plus grande valeur.
- Pour entraîner le réseau de neurones, on génère des données d'apprentissage de la manière suivante. Après avoir choisi l'action a , on se trouve dans l'état s' . On calcule alors les valeurs de $Q(s', a)$ pour toutes les valeurs possibles de a , et on en déduit $\max_a Q(s', a)$. Cela permet d'appliquer la formule 1 afin d'obtenir la valeur *target*. On génère alors un point d'entraînement avec comme entrée les valeurs (s, a) et comme sortie la valeur *target*.
- Le réseau de neurones peut alors être entraîné à partir des données ainsi générées, et il va pouvoir approximer les Q-valeurs de manière de plus en plus précise.

Les plus courageux d'entre vous pourront tenter d'implémenter l'algorithme du *deep Q-learning* grâce au modèle `MLPRegressor` de la bibliothèque `sklearn`, dont la [documentation](#) est accessible en cliquant sur le lien. Cela devrait vous permettre de répliquer l'apprentissage de l'évitement d'obstacles avec la caméra.