ML : Q-learning

TP5 Apprentissage automatique

Q-learning

Jérôme Buisine
jerome.buisine@univ-littoral.fr

12 décembre 2022

Durée : 3h

L'objectif de ce TP est la mise en place d'un algorithme de Q-learning dans un contexte de jeu (environnement) où un agent doit atteindre un objectif.

1 Ressources

Voici un ensemble de ressources qui peuvent vous être utiles durant ce TP :

— 1. Documentation officielle de l'outil `Git` ;
— 2. Gitlab : interface web pour la gestion de projets versionnés sous `Git` ;
— 3. pyenv : permet la gestion d'environnement Python.
— 4. matplotlib : librairie Python qui permet un affichage rapide de données.
— 5. jupyter : un outil de travail permettant une interaction rapide et visuelle avec une console Python.
— 6. Gym environnement : la documentation des environments de jeux proposés par l'API Gym.

2 Configuration de l'environnement

Créez un dossier à la racine de votre projet nommé `tp5-qlearning`. Dans ce dossier et depuis votre terminal, lancez les commandes suivantes :

```
# spécifie l'environnement virtuel Python à Jupyter
ipython kernel install --user --name=ml-venv
# lance l'application Jupyter
jupyter-lab
```

3 Fonctionnement de l'API Gym

Il vous faudra tout d'abord installer les dépendances qui nous permettront d'utiliser l'API `Gym` :

```
pip install gym pygame
```

FIGURE 1 – Aperçu du jeu à l'initialisation

Puis créez un nouveau notebook nommé « gym_introduction.ipynb ». L'objectif est de tout d'abord se se familiariser avec l'API gym.

Pour cette première partie de TP, nous allons utiliser l'environnement de jeu « `FrozenLake-v1` » où un agent doit se déplacer vers un objectif sur une carte simplifiée mais piégée (voir figure 1). Voici quelques informations sur l'environnement :

- Les variables `action_space` et `observation_space` donnent des informations globales sur l'environnement.
- L'agent peut se déplacer en haut, en bas, à gauche et à droite (quand cela est possible).
- L'agent connaîtra à chaque étape de jeu, après réalisation d'une action, l'endroit où il se situe sur la carte.
- Le jeu se termine lorsque l'agent atteint l'objectif ou qu'il tombe une cellule de glace.
- La méthode `reset` permet de réinitialiser l'environnement.
- La méthode `render` permet l'affichage du jeu soit en mode `ansi`, soit en mode `rgb_array`.
- La méthode `step` qui prend en entrée une action à réaliser et fournie des informations : notamment sur l'état suivant et la récompense obtenue.

Note : vous pouvez vous référer à la documentation spécifique à l'environnement FrozenLake-1.

En important le package Python `gym`, il est possible d'initialiser l'environnement de jeu de la manière suivante :

```
import gym
env = gym.make('FrozenLake-v1', desc=None, map_name="4x4", is_slippery=False,
        render_mode="rgb_array")
```

Tâche 1 : Créez votre environnement et proposez un affichage de l'état initial.

Indications :

→ Vous pouvez pour cela utiliser la méthode `render` de votre environnement.

→ Il est possible d'afficher une image avec `matplotlib` avec la méthode `imshow` du module `matplotlib.pyplot`.

Tâche 2 : Faire avancer manuellement votre agent dans l'environnement nouvellement créé de tel sorte à ce qu'il atteigne son objectif (voir un exemple en figure 2).

Indications :

→ Vous pouvez pour cela utiliser la méthode `step` de votre environnement qui retourne des informations importantes sur votre nouvel état.

→ Utiliser l'`action_space` (voir documentation Gym).

FIGURE 2 – Nouvel état du jeu après avoir choisi l'action `DOWN`

4 Q-learning pour le jeu « Cliff walking »

Nous allons maintenant développer l'algorithme du Q-learning sur le jeu « Cliff walking ». Un aperçu de la position de l'agent initial est disponible en figure 3.

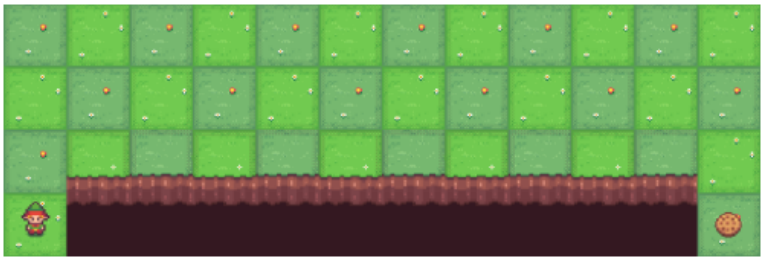

FIGURE 3 – Aperçu de l'état initial pour le jeu « Cliff walking »

Voici quelques particularités de l'environnement :

- L'objectif de l'agent est d'aller chercher le cookie à l'opposé de la carte sans aucune information au préalable.
- L'état de l'agent est désigné par la cellule sur laquelle il se trouve.
- Pour chaque pas effectué, l'agent reçoit une récompense négative de -1.
- Si l'agent tombe dans le ravin, alors il recommence au point de départ et récupère une récompense négative de -100.
- La partie se termine lorsque l'agent trouve le cookie.

Il est possible d'initialiser l'environnement de jeu de la manière suivante :

```
import gym
env = gym.make('CliffWalking-v0', render_mode="rgb_array")
```

Tâche 3 : Simuler une partie complète (épisode) en proposant des mouvements aléatoires jusqu'à ce que le personnage atteigne son objectif (fin de partie).

Tâche 4 : Initialiser une matrice Q-table de la taille de l'environnement (observation) et du nombre d'actions possibles. Toutes les valeurs doivent être initialisées à 0.

Indications :

→ Vous pouvez utiliser la librairie `numpy` et notamment `numpy.zeros`.

→ La Q-table va permettre de stocker l'ensemble des récompenses obtenues pour chaque tuple (état, action).

Tâche 5 : Définir une fonction `epsilon_greedy` qui prend en paramètres la Q-table, un état s et une valeur d'epsilon fixée à 0.1 par défaut. Cette fonction retourne l'indice d'une action. Pour un nombre r aléatoire ($r \in [0, 1]$), la fonction doit :

— Si toutes les $Q(s, a_i)$ sont de la même valeur, on retourne une action aléatoire.

— Si $r < \epsilon$, alors on retourne une action aléatoire pour l'état.

— Sinon, on retourne l'action ayant le plus de récompenses.

Tâche 6 : Développer l'algorithme du Q-learning avec comme politique de choix ϵ-greedy :

— Le taux d'apprentissage α sera fixé par défaut à 0.5.

— Le facteur d'actualisation γ sera fixé à 0.9.

— À chaque fin de partie, il faudra fixer une récompense positive de réussite pour l'agent.

— Proposez de simuler 200 épisodes et visualiser votre Q-table.

Tâche 7 : Visualiser les performances de convergence de l'algorithme du Q-learning :

— Pour chaque épisode simulé lors de l'apprentissage, stocker le nombre de pas nécessaire pour que l'agent atteigne son objectif.

— Visualiser à l'aide de `matplotlib` les nombres de pas effectués pour chacun des 200 épisodes.

Indications :

→ On nomme « épisode » une partie terminée du jeu.

→ Pour le jeu étudié en question, celui-ci se termine une fois l'objectif atteint.

Questions

- Quels sont les avantages d'un tel algorithme ?
- Quelles sont les limites actuelles de l'algorithme dans le cadre de ce jeu ?

5 Bonus

Tâche 8 : Étudier les paramètres α et γ de l'algorithme Q-learning :

- Que pouvez-vous en déduire, par exemple pour $\alpha \in [0.1, 0.5, 0.9]$ et $\gamma = 0.9$?
- Que pouvez-vous en déduire, par exemple pour $\gamma \in [0.1, 0.5, 0.9]$ et $\alpha = 0.5$?

Tâche 9 : Développer l'algorithme SARSA, qui est lui orienté politique de choix :

- Comparer les performances des deux algorithmes développés.
- Que pouvez-vous en conclure ?

6 Remise des travaux

N'oubliez pas de réaliser un commit de vos travaux. Taguez également votre projet avec le tag « `tp5` » et soumettez-le sur le serveur Gitlab. Il fera office de rendu.