Projet - Rock, Paper, Scissors

S1 2020-2021

1 Introduction

L'objectif du projet est le challenge kaggle pierre-papier-ciseaux. Les règles du jeu sont très simples : deux joueurs s'affrontent et doivent choisir à chaque tour un objet parmi le triplet *Pierre, Papier, Ciseaux* ; chaque joueur gagne ou perd un point selon la configuration : le papier gagne sur la pierre, la pierre gagne sur les ciseaux et les ciseaux gagne sur le papier ; le jeu se termine au bout d'un nombre de tours fixé à l'avance (1000 dans le cadre du challenge) et le gagnant est celui qui a le plus de points (avec une marge de 20 points dans le challenge, sinon égalité).

Nous vous demandons dans ce projet de réaliser des agents **en utilisant du Reinforcement Learning** - ce qui n'est pas une évidence vu les récompenses immédiates que les joueurs ont durant le jeu. Ce qui sera jugé en priorité dans votre projet ce sont les idées, leur mise-en-œuvre et votre protocole expérimental (et non pas le résultat de l'agent).

Vous pouvez bien sûr utiliser des agents naïfs et d'autres plus perfectionnés lors de l'entraînement de vos agents. Beaucoup d'agents sont disponibles (par exemple https ://www.kaggle.com/chankhavu/rps-dojo) que vous pouvez utiliser.

Vous êtes encouragé à discuter entre vous et à échanger vos agents. Le projet est à rendre en binôme sous la forme d'un rapport et de votre code.

2 Détails techniques

2.1 Module `kaggle_environments`

Vous devez installer le module `kaggle_environments`, qui fonctionne de manière similaire à `gym` mais plus orienté évaluation des agents.

Pour créer l'environnement : `make("rps", debug=[False|True], configuration = configuration`

La variable `configuration` contient le dictionnaire de configuration de la partie :

- `actTimeOut` : le temps que ne doit pas dépasser l'agent lors de son initialisation (1 seconde)
- `agentTimeOut` : le temps total dont dispose l'agent au début du jeu (60 secondes)
- `runTimeOut` : le temps maximum d'un épisode (non applicable dans ce jeu)
- `tieRewardThreshold` : la différence de points minimum à la fin du jeu pour qu'un vainqueur soit déclaré
- `signs` : nombre d'objets (ici 3, jusqu'à 5 possible)
- `episodeSteps` : nombre de tour de jeu (ici 1000)

Un agent est une fonction à deux arguments `my_agent(observation,configuration)` qui doit retourner l'action (ici un entier entre 0 et 2). La variable `configuration` contient la configuration de l'environnement et la variable `observation` contient le dictionnaire :

- `step` : le numéro de tour du jeu ;
- `remainingOverageTime` : le temps qu'il reste à l'agent pour réfléchir jusqu'à la fin du jeu (61 secondes au début) ;
- `lastOpponentAction` : le dernier coup joué par l'adversaire (n'existe pas au premier tour) ;

Vous disposez des commandes suivantes dans l'environnement :

- `env.reset()` pour réinitialiser l'environnement
- `env.run([agent1,agent2])` : pour faire jouer un agent contre un autre agent

— `evaluate(env,[agent1, agent2], configuration,num_episodes=10)` : pour évaluer deux agents sur `num_episodes` épisodes.
— `env.agents` : renvoie la liste des agents par défaut de l'environnement. Vous pouvez passer la chaîne de caractère directement à `run` ou `evaluate` pour jouer contre eux.

Pour entraîner un agent, vous devez créer un `trainer` : `env.train([None, adversaire])`. Cette structure contient une méthode `step(action)` qui prend l'action de votre agent et renvoie le n-uplet `obs, reward, done, info` comme `gym` et une méthode `reset()` pour réinitialiser le `trainer`. Le `None` spécifie la place de l'agent à entraîner, l'agent adverse doit obligatoirement être fourni.

Vous pouvez également charger un agent en indiquant uniquement dans la variable le nom du fichier python de votre soumission selon le format décrit ci-dessous.

2.2 Fichier de soumission

La soumission sur le site de `kaggle` est très particulière : vous pouvez soumettre un fichier python qui contient **comme dernière fonction déclarée** votre agent. Vous n'avez le droit dans ce fichier d'importer que les modules de la librairie standard, `gym`, `numpy`, `scipy` et `pytorch` en **CPU** seulement (pas de GPU). Cependant ce type de soumission rend difficile la soumission de réseaux de neurones (à moins de coder en binaire les poids de votre réseau dans le fichier). Vous pouvez également soumettre une archive `tar.gz` (et qui doit finir obligatoirement par cette extension) qui doit obligatoirement avoir un fichier `main.py`, votre agent étant toujours la dernière fonction déclarée. Votre répertoire de travail sur la plateforme `kaggle` est `/kaggle_simulations/agent` ce qui va vous permettre d'importer des modules locaux ou des fichiers sauvegardés avec `pickle`.

Un exemple vous est fourni dans le code d'accompagnement du sujet. Nous vous conseillons d'avoir un code différent pour l'entraînement de votre agent et pour l'agent que vous soumettez (un code épuré). Le fichier `tuto.py` reprend les principales commandes de l'environnement, le fichier `agents/randomagent.py` donne un exemple d'un agent avec un réseau de neurones, le fichier `training.py` montre comment sauvegarder en `pickle` votre agent et le fichier `main.py` est un exemple de fichier de soumission (ne pas oublier de charger le module qui contient la classe de votre agent). La soumission doit être obligatoirement faite ensuite avec la commande `tar -cvzf soumission.tar.gz agents/ main.py`.