Bio-Inspire

Deep Q Learning

Arnaud Bressot 11505990

Valentin Comas 11500223

Bio-Inspiré

Deep Q Learning

Table des matières

INTRODUCTION

Le but de ce TP était d'utilisé l'algorithme de Q-Learning, couplé avec des réseaux neuronaux afin de résoudre des environnements mis à disposition par gym : tout d'abord un simple : Cartpole puis un plus complexe faisant notamment utiliser la convolution 2D : Breakout.

Nous avons réalisé la partie Cartpole en entier. De son côté, la partie Breakout compile mais n'est pas fonctionnelle bien que codée dans son ensemble (voir partie dédiée).

.

I. CARTPOLE

a) Implémentation

L'implémentation des réseaux de neurones s'est faite avec pytorch (fichier et classe DeepQNetwork). Après tests, nous avons choisis de mettre 3 couches de neurones prenant une entrée à 4 (taille de l'état), puis 16, puis 8 pour se retrouver avec une sortie de 2 (nombre d'actions possibles) avec activation sigmoid entre les couches (apparemment plus efficace que ReLU).

Pour la stratégie d'exploration, nous avons implémenté la stratégie greedy, pour la simplicité de sa mise en place.

La classe Agent va contenir les éléments nécessaires au Q-learning : les réseaux de neurones eval et target, l'optimizer et la loss function. L'apprentissage se fait dans optimize_model().

Tout d'abord, nous avons essayé de calculer les Q-Values à partir d'opérations directement sur les tensor. Cependant, pour des raisons qui nous sont encore inconnues, ce code compilait mais n'obtenait aucun résultat viable. Nous avons donc ensuite calculé les Q-Values une par une, pour chaque ligne du batch. En utilisant cette méthode, le code s'est considérablement ralentit mais garantit des bons résultats.

b) Résultats

Les résultats sont détaillés ici.

Globalement, un paramètre semblant le plus impacter l'apprentissage est epsilon (stratégie greedy). Plus on s'approche de 0.5, plus le modèle est efficace. Cela peut s'expliquer par le fait qu'une fois légèrement entraîné, il pourra se débrouiller pour ne pas mourir et l'aléatoire permet d'explorer de nouveaux états, améliorant l'apprentissage.

On peut également constater que le learning_rate influe énormément sur la performance.

Le batch_size ne doit pas être trop bas sous peine de tomber au final sur de l'aléatoire. Cependant, le placer trop haut ne sert qu'à ralentir l'apprentissage, sans être spécialement plus efficace.

Gamma ne semble pas influé plus que ça sur l'apprentissage (pourvu qu'il soir élevé).

II. BREAKOUT

Pour cette partie, nous avons mis en place un réseau de convolution.

Cependant, nous avons rencontré un problème non résolu au niveau du pré-processing qui semble toujours renvoyé le même état. Au vu du réseau que nous avons, nous ne pouvons que supposé qu'il marche étant donner que l'algorithme de Q-Values marche pour le réseaux précédents et que les états peuvent rentrés et traitées par ce nouveau réseau.