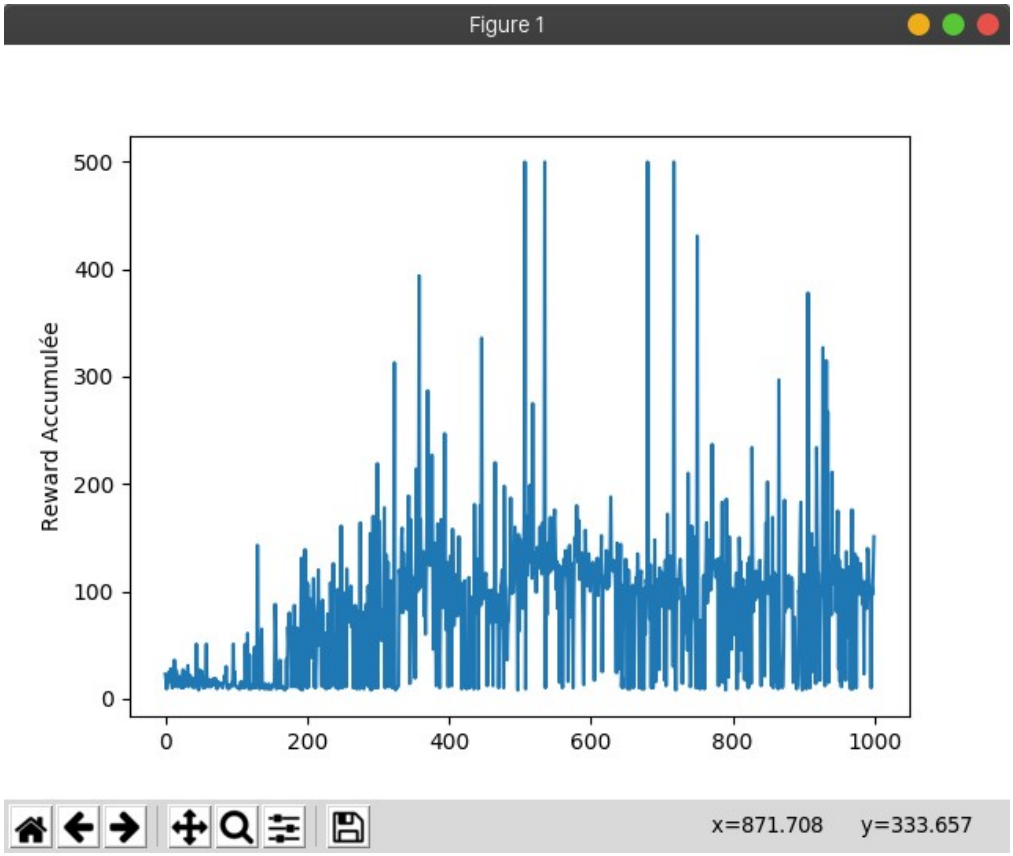
MAGNANA Lucas
FLANDIN Alban

I. Introduction

Dans le cadre de l'UE Intelligence Bio-Inspirée, nous avons développé 2 agents capable (en théorie) d'apprendre à maîtriser 2 environnements. Nous avons choisi d'utiliser PyTorch pour nous aider. Le programme est constitué de 2 fichiers main, de 2 agents et de 2 réseaux de neurones distincts. Pour lancer la première partie, il faut exécuter le fichier mainStick.py avec python 3. Pour lancer la deuxième partie, il faut exécuter le fichier mainAtari.py.

II. L'environnement Cartpole-V1

Pour cet environnement, nous avons codé un réseau de neurone qui approxime la fonction Q pour déterminer quelle action (gauche ou droite) notre agent doit choisir pour maintenir le bâton debout. Notre réseau est un shallow network dont la couche cachée est de taille 15. Nous avons testé différentes tailles (jusqu'à 25) et c'est une couche de taille 15 qui nous a donnée les meilleurs résultats. Le problème de l'utilisation d'un réseau de neurone pour approximer la fonction Q est que lorsque l'on calcule l'erreur dans le but de propager le gradient, il faut calculer la valeur de Q de l'état précédent. Modifier le réseau pour un état modifie donc aussi ce réseau pour l'état précédent. Pour régler ce problème, il nous a été demandé d'utiliser un réseau cible qui servira uniquement à calculer la Q valeur des états précédents et qui ne sera pas modifié de la même façon que le premier réseau. Nous avons choisi de modifier le réseau cible en copiant le premier réseau dans celui-ci toutes les 5000 étapes d'apprentissage (une étape d'apprentissage est une rétropropagation du gradient dans le réseau «principal»). Nous avons aussi choisi d'arrêter l'apprentissage du réseau cible au bout de 75000 étapes d'apprentissage pour ne pas surapprendre (car nous avons remarqué une baisse des récompenses au bout d'un certain nombres d'épisodes). Pour la stratégie d'exploration, nous avons choisi la e-greedy. Après plusieurs tests, nous avons décider de commencer avec un epsilon à 1 puis de le multiplier par 0.99 à chaque fois que l'agent apprend. Pour l'apprentissage, nous utilisons des samples de 100 sur un buffer de 100000. Pour le reste, le taux d'apprentissage est de 0.01, l'optimiseur utilisé est Adam et le gamma est à 0.9. Tous les tests ont été faits sur 1000 épisodes, et l'agent apprend à la fin de chacun de ces épisodes. Pour évaluer les performances des hyper-paramètres, un graphe est affiché à la fin des 1000 épisodes montrant les récompenses finales de chacun des épisodes. Un exemple est affiché ci-dessous.



III. L’environnement BreakoutNoFrameskip-v4

Cet agent marche en théorie mais dans la pratique, un épisode dure plus de 20 minutes et nous n’avons pas pu comprendre pourquoi.

Pour cet agent, nous avons commencé par reprendre l’agent précédent. La première étape a été le preprocessing des données. Le wrapper AtariPreprocessing nous a permis de faire quasiment tous les traitements sauf le stacking des 4 frames. Il a donc fallu que nous utilisions la wrapper FrameStack qui nous renvoie une LazyFrame que nous devons reconvertir en tenseur, et ce à chaque étape de chaque épisode (peut être une perte de performances ?). Une fois cela fait, il a fallu utiliser la fonction «unsqueeze» de PyTorch car la dimension du tenseur n’était pas la bonne (shape de (4, 84, 84) tandis qu’un reseau convolutionel attend un tenseur de 4 dimensions). Nous avons ensuite pu créer notre réseau convolutionel. Celui-ci est constitué de 2 objes Conv2D. A la fin de chaque passage, nous passons le résultat de la transformation convolutionel dans un MaxPool2d de taille 2x2. Cela nous permet de réduire la taille des données ainsi que de mieux détecter les formes présentes dans l’image. Une fois les transformations convolutionels faites, nous passons le résultat (après un reshape) dans un deep network de 3 couches pour finalement retrouver la Q valeur des 2 actions possibles. N’ayant pas pu tester cet agent à cause des problèmes de performances, nous n’avons pas pu tenter d’améliorer les hyper-paramètres du réseau. Quand à l’agent, il garde les mêmes paramètres que le précédent.