TP Apprentissage profond par renforcement

Antoine DULHOSTE et Imran KRIMI

1 Deep Q-network sur CartPole

1.1 Le buffer

Nous avons donc fait un buffer pour nous souvenir de chaque step pour pouvoir entrainer notre modèle. Chaque step contient un état, une action, l'état suivant, la récompense associée et si l'épisode est fini. La fonction sample du buffer permet de récupérer un sample aléatoire de taille égale au batch et sépare tous ses composants. La taille du buffer est définie avec le paramètre "B_SIZE".

1.2 Le réseau neuronal

Pour le réseau neuronal nous avons fait un réseau composé de 4 neurones en entrées correspondantes au 4 données d'un état puis trois couches à 32 neurones et enfin une couche a deux neurones correspondants à la Q-valeur des deux actions possibles. Nous avons choisi comme fonction d'activation Relu. Le réseau prend en paramètres ici la taille de l'état et le nombre d'actions ici respectivement 4 et 2.

1.3 La Stratégies d'exploration

Nous avons choisi la stratégie d'exploration epsilon greedy, ici epsilon à valeur de départ "EPS_START" et cette valeur décroit au fur et à mesure des itérations a une valeur "EPS_END". La formule utilisée ici est:

EPSILONE = EPS_END + (EPS_START - EPS_END) * e(-1. * nb_itération / EPS_DECAY)

Nous avons fait le choix d'enlever cette stratégie lors de l'évaluation du modèle, donc lors de l'évaluation nous choisissons toujours les actions avec la plus forte Q-valeur.

1.4 L'entrainement du modèle

Pour ce qui est de l'entrainement du modèle nous lançons un nombre d'épisodes décrits dans la variable "NB_EP", puis pour chaque épisode, tant qu'il n'est pas terminé, nous demandons à l'agent l'action qu'il veut faire en lui passant l'état actuel de l'environnement, puis il nous retourne une action en fonction de ce que le réseau de neurones lui a retourné et de la stratégie d'exploration. Puis nous enregistrons le step dans le buffer puis s'il y a assez de données dans le buffer par rapport à la taille du batch "BATCH_SIZE" alors on lance une phase d'entrainement de l'agent à chaque itération.

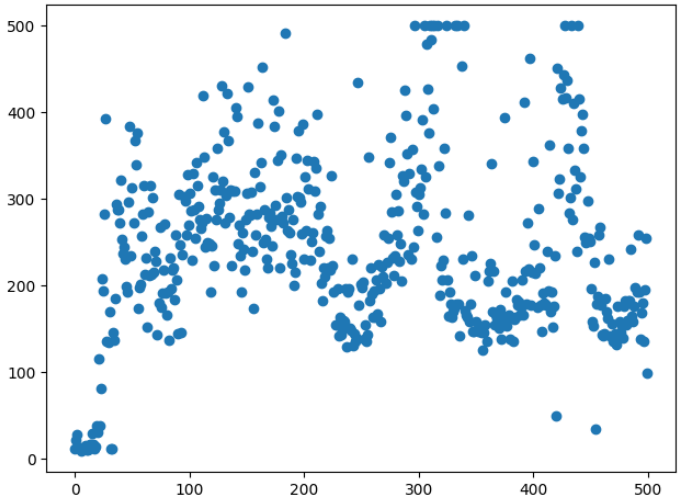
Pour chaque phase d'entrainement, l'agent demande un sample du buffer puis passe toutes les données de chaque état dans le modèle qui nous retourne les Q valeurs correspondantes où on sélectionne les valeurs correspondant aux actions effectuées à la suite de ces mêmes états. On calcule la Q valeur que l'on attendait grâce aux formules de Bellman en passant les états suivants dans un autre modèle et grâce au reward enregistré, si l'état est le dernier de l'épisode grâce au boolean "done" on ne calcule que la reward. Puis une fois ces deux valeurs calculées pour le nombre de valeurs de step définies grâce a "BATCH_SIZE" on calcule la loss et on fait remonter les gradients puis on met à jour les paramètres du modèle.

1.5 Clonage et mise à jour du réseau neuronal

Au niveau du clonage nous avons donc cloné le même réseau pour calculer les Q valeurs des états suivants. Pour cela notre agent a donc deux modèles identiques, mais lors de l'entrainement, seul celui qui calcule les Q valeur pour savoir quelle action effectuer est mis à jour. Pour mettre à jour l'autre modèle on choisit de le faire au nombre d'épisodes, c'est-à-dire que tous les N épisodes (où N peut être modifié grâce à la variable "UPDATE") on copie les poids du modèle dans le modèle cloné.

1.6 Les paramètres et résultats

Au niveau de la fonction d'optimisation, on a choisi Adam puis on a fait une étude empirique pour savoir quel taux d'apprentissage et quel "weight decay" permettraient d'avoir les meilleurs résultats. Avec les paramètres suivants: le taux d'apprentissage a 0.001, le "weight decay" a 1e-5, le batch size à 100, la taille du buffer à 100 000, l'epsilon qui démarre à 0.9 et qui descend à 0.05, le gamma a 0.9, le nombre d'épisodes à 1000 et l'update tous les 10 épisodes ce qui nous a donné le résultat pour l'apprentissage suivant:



CartPole Apprentissage : total des récompenses en ordonnée pour chaque épisode en abscisse

2 Deep Q-network sur Vizdoom

2.1 Les partie qui n'ont pas changé

Pour cette partie nous avons conservé le même buffer et la même architecture de code notamment de l'entrainement et de l'agent. L'entrainement a quelque peu changé au niveau de la création du batch ou au lieu de faire passer des tenseurs de taille [taille_état,taille_du_batch] on a dû concaténer les différentes images.

2.2 Le pre-processing

Pour ce qui est du pre-processing nous avons utilisé le code fourni dans le TP utilisant Skimage. Toutes les images contenues dans le buffer sont passées par le pre-processing avant d'y être ajoutées.

2.3 Le réseau de neurones convolutionnel

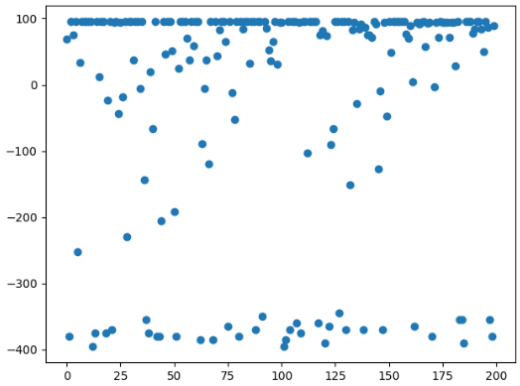
Pour ce qui est du réseau de neurones, nous avons donc dû changer pour pouvoir être plus efficaces et analyser les images. Nous avons donc utilisé trois couches de convolution, une utilisant 16 filtres puis les deux autres 32, que l'on alterne avec trois couches de batch normalisation puis nous finissons par une couche de neurone, ou l'on calcule la taille de neurone en entrée en fonction de la taille de l'image à l'entrée du réseau, diminué par son passage par les couches de convolution et en sortie le nombre d'actions possibles. Nous n'avons pas changé la fonction d'activation qui est toujours Relu.

2.4 La sauvegarde des poids du modèle

Comme l'apprentissage est devenu long nous avons dû implémenter la sauvegarde des poids du modèle pour ne pas avoir à apprendre à chaque fois que l'on veut seulement un rendu donc pour cela le paramètre "SAVE" permet de sauvegarder les poids quand l'apprentissage est fini, nommer avec le paramètre "SAVE_NAME" et enfin le paramètre "USE_SAVE" permet s'il est à True d'utiliser la sauvegarde et de ne pas entrainer le modèle il va chercher à l'emplacement "SAVE_NAME". Un fichier contenant des poids pour chacun des environnements testés (CartPole, VizdoomBasic, VizdoomCorridor et VizdoomTakeCover) est enregistré.

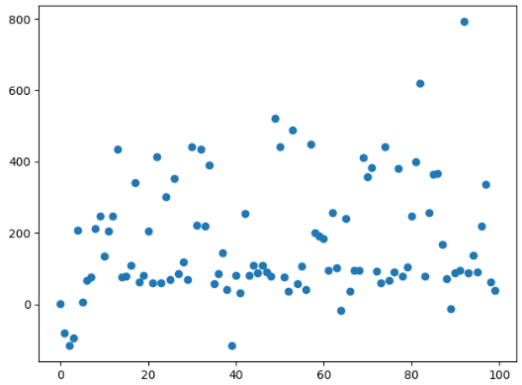
2.5 Les paramètres et résultats

Nous avons testé trois environnements de Vizdoom: VizdoomBasic, VizdoomCorridor et VizdoomTakeCover. Pour chacun d'eux nous avons gardées certains paramètres: le taux d'apprentissage a 0.001, le "weight decay" a 1e-3, le batch size a 5, la taille du buffer a 100 000, l'epsilon qui démarre a 0.9 et qui descend a 0.05, le gamma a 0.9, le nombre d'épisodes a 100 sauf pour l'environnement VizdoomBasic qui demande moins d'itérations donc permet de faire plus d'épisodes et l'update tous les 2 épisodes. Pour VizdoomBasic avec un taux d'apprentissage a 0.001. Les résultats pour ces trois environnements sont différents:

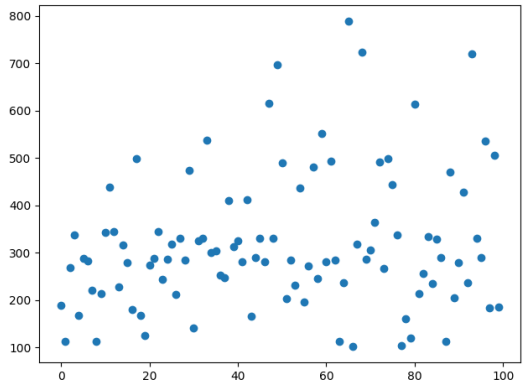


VizdoomBasic Apprentissage : total des récompenses en ordonnée pour chaque épisode en abscisse

On peut voir ici que l'agent a appris, car la plupart des récompenses par épisode se trouvent dans le positif cependant, il reste des cas où il n'arrive pas à tirer sur la cible, une probable amélioration serait d'augmenter le temps d'apprentissage. La taille du batch et la résolution de l'image, ce qui aurait pour effet d'augmenter significativement le temps d'exécution sur nos machines.



VizdoomCorridor Apprentissage : total des récompenses en ordonnée pour chaque épisode en abscisse



VizdoomTakeCover Apprentissage : total des récompenses en ordonnée pour chaque épisode en abscisse

Dans les deux autres environnements, l'apprentissage est limité, on voit que l'agent s'améliore, mais c'est assez faible. Il y a plusieurs raisons à ça, premièrement la complexité de l'environnement pour le corridor il y a sept actions possibles et pour le take cover il faut éviter un grand nombre de tirs. Deuxièmement, il faudrait augmenter la taille du batch, la résolution des images et le nombre d'épisodes pour augmenter la précision et enfin il faudrait implémenter le frame stacking, ce qui pourrait augmenter grandement l'apprentissage, car avoir une suite d'images permettrait à l'agent de mieux comprendre le mouvement d'évitement pour l'environnement take cover par exemple.

Pour les trois environnements Vizdoom nous avons filmé l'évaluation de l'agent sur 5 épisodes, pour l'environnement basique, l'agent réussi quatre fois à tirer sur l'ennemi étant apparu en face de lui puis sur le cinquième épisode il se décale à droite, puis tire sur l'ennemi. Pour l'environnement "corridor" on voit que l'agent comprend que la meilleure façon de maximiser sa récompense et de courir tout droit, mais il n'essaie pas de tirer sur les ennemies et enfin sur l'environnement take cover l'agent se précipite sur le mur de gauche et esquive le plus souvent le premier tir, mais ne fait en général pas plus.