DQN Reinforcement Learning

Param Dave

Décembre 2023

Sommaire

1 Introduction

Ce projet est une ré-implémentation des algorithmes et concepts proposés dans le papier *Human-level control through deep reinforcement learning* par **DeepMind** dans le cadre du cours de Reinforcement Learning à **EPITA**.

L'objectif principal de ce projet consiste à mettre en oeuvre un modèle de DQN capable de s'entraîner et jouer sur les jeux ATARI de **Gym OpenAI**.

Nous allons tout d'abord nous interesser aux preprocessing des frames du jeu, ensuite des algorithmes d'entraînement, puis des résultats obtenus. Enfin nous relèverons les difficultés et contraintes de ce projet et les possibles pistes d'améliorations.

2 Preprocessing

Le preprocessing se découpe en plusieurs étapes. Travailler directement avec les images brutes de l'Atari 2600, qui sont des images de 2103 par 160 pixels avec une palette de 128 couleurs, peut être exigeant en termes de calculs et de besoins en mémoire.

Il faut appliquer une étape de prétraitement de base visant à réduire la dimensionnalité de l'entrée et à traiter certains artefacts de l'émulateur Atari 2600.

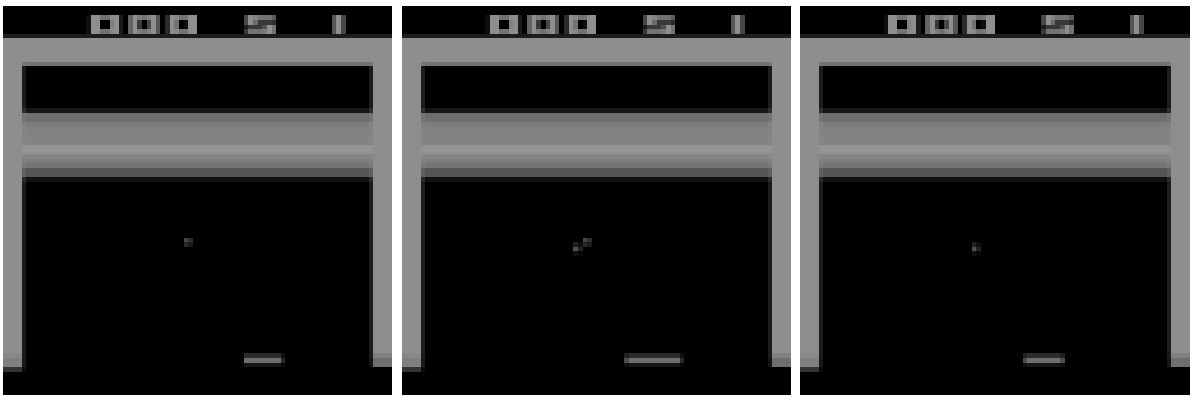
Tout d'abord, pour coder une seule image, nous prenons la valeur maximale pour chaque pixel en couleur sur l'image en cours de codage et l'image précédente. C'était nécessaire pour éliminer le scintillement présent dans les jeux où certains objets n'apparaissent que dans les images paires tandis que d'autres n'apparaissent que dans les images impaires, un artefact causé par le nombre limité de sprites que l'Atari 2600 peut afficher à la fois.

Deuxièmement, il faut ensuite extraire le canal Y, également connu sous le nom de luminance, de l'image RGB et le redimensionner à 84 par 84. La fonction w de l'algorithme 1 décrit ci-dessous applique ce prétraitement aux m images les plus récentes et les empile pour produire l'entrée de la fonction Q, où m est égal à 4, bien que l'algorithme soit robuste à différentes valeurs de m (par exemple, 3 ou 5).

Pour simplifier le code, certains wrappers de **Gym OpenAI** sont disponible tels que:

- **GrayScaleObservation**: donne les frames en grayscale
- **ResizeObservation**: donner aux frames une certaine dimension (84x84 ici)
- **FrameStack** permet de donner directement des stacks de n frames

Voici un exemple arbitraire de prétraitement:



3 Algorithme

La majorité de l'algorithme d'entraînement se trouve dans la fonction *play_and_train* du fichier **train_model.py** et tout le fichier **dqn_agent.py**.

Nous lançons un nombre arbitraire d'épisode pour l'entraînement. Pour chaque épisode, tant qu'il n'est pas fini, nous évaluons la prochaine action et stockons les résultats nécessaires dans le *replay_buffer*. L'action est déterminée soit aléatoirement (exploration) soit en utilisant le modèle DQN.

Nous avons également un deuxième modèle de DQN qui ne s'update pas à chaque étape. Ce modèle est nécessaire pour assurer la stabilité de notre modèle et éviter que cela diverge.

Une fois que le *replay_buffer* est plein, nous updatons le deuxième modèle.

Nous réitérons cet algorithme en boucle.

Les actualisations des modèles de DQN sont effectuées de cette manière:

```
qvalues_online = self.online_net(states).gather(1, actions.unsqueeze(1))

qvalues_next = self.target_net(next_states).max(1)[0].detach()

qvalues_target = rewards + (1 - dones) * self.gamma * qvalues_next
```

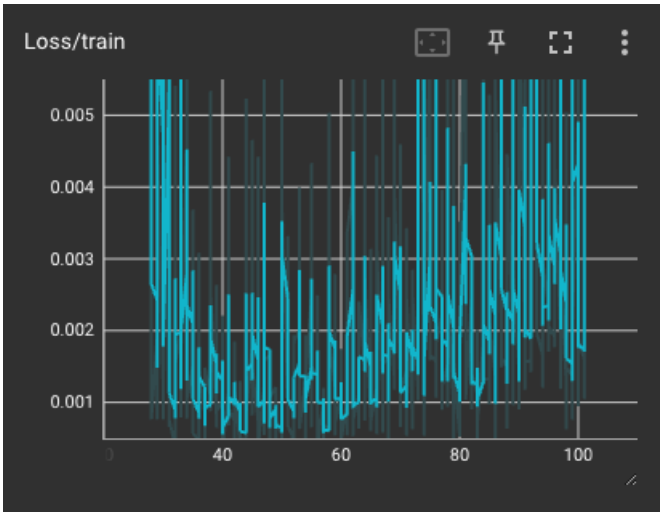
La fonction à miniser est la **Huber Loss** entre les *qvalues* des deux DQN.

4 Résultats

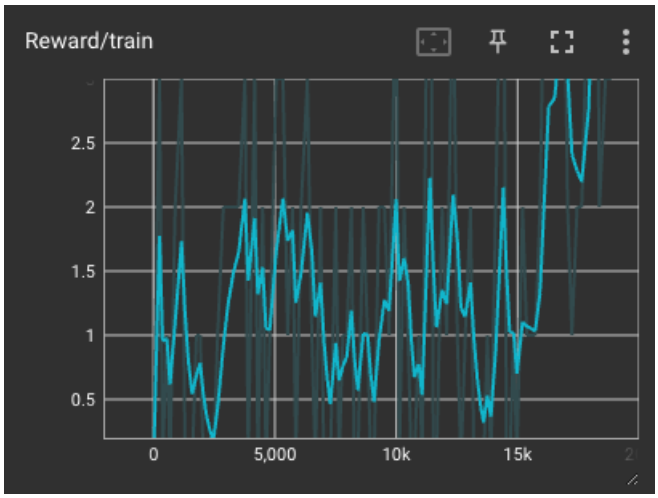
Pour afficher et analyser les résultats, j'ai utilisé **TensorBoard**.

Malheureusement les résultats n'atteint jamais 20 et varie énormément entre 0 et 12. De plus, cela se produit après 10 heures d'entraînement.
Sur 30 minutes d'entraînement sur le jeu **Breakout-v5**, voici les différentes métriques observées:

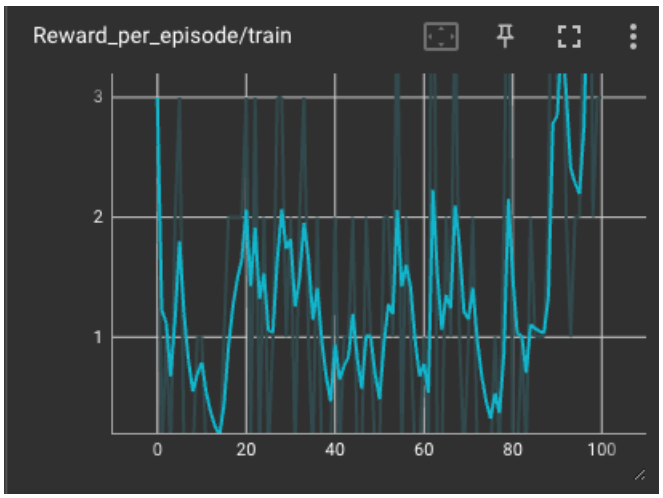
La loss:



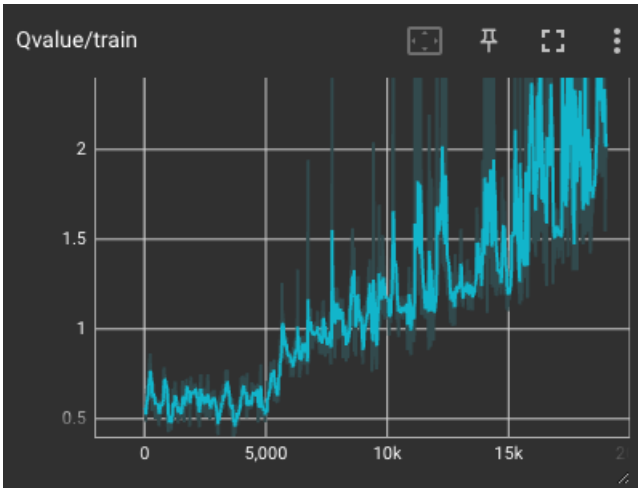
La reward en continu:



La reward par épisode:



Et enfin, la qvalue:



Au bout de 5 heures d'entraînement, nous observons que le modèle n'apprend plus et dans le pire cas, diverge.

5 Améliorations

Nous observons donc que les résultats ne sont pas satisfaisant et très loin des résultats de **DeepMind**. Voici les différentes améliorations possibles:

- Passer au GPU: En effet avec le GPU nous iront beaucoup plus vite et nous auront moins de temps d'entraînement pour les mêmes résultats
- Rescale les hyperparamètres: Étant donné que ma machine a une puissance de calcul moins forte, downscale les hyperparamètres pourrait être une solution. Cependant, nous ne pourrons pas avoir des résultats comme **DeepMind**
- Essayer sur d'autres jeu: j'ai éssayé sur **Breakout** et **Boxing** (pour avoir des récompenses négatives). Cependant, essayer sur d'autres jeux pourrait mieux m'aider à améliorer mon algorithme et modèle