Intelligence Bio-Inspirée
Travaux Pratiques

Compte-rendu des travaux pratiques sur l'apprentissage profond par renforcement

Réalisé par

SUBLET Gary – 11506450
VILLERMET Quentin – 11507338

Polytech Lyon
Université Claude Bernard Lyon 1

2020

Table des matières

1 Réplicabilité

1.1 Environnement virtuel

La totalité des expérimentations suivantes ont été réalisées avec Python 3.7, au sein de *notebooks jupyter* et de scripts *.py*.

Afin de favoriser la facilité de reproduction de ces programmes, nous évoluons dans un environnement virtuel, où les modules python peuvent être plus aisément contrôlés.
Nos dépendances sont listées dans le fichier `requirements.txt` à la racine du projet.

Un guide permettant la mise en place de ce *virtualenv* est présent dans le `Readme.md` du dépôt GitHub.

2 Deep Q-network sur CartPole

2.1 Début

2.1.1 Question 1 – Agent aléatoire CartPole-v1

Le code de l'agent aléatoire est entièrement contenu dans le fichier `cartpole.py`, trouvable dans les sources du projet.

Il est presque entièrement basé sur l'exemple des créateurs de `gym` présent sur le GitHub du module.

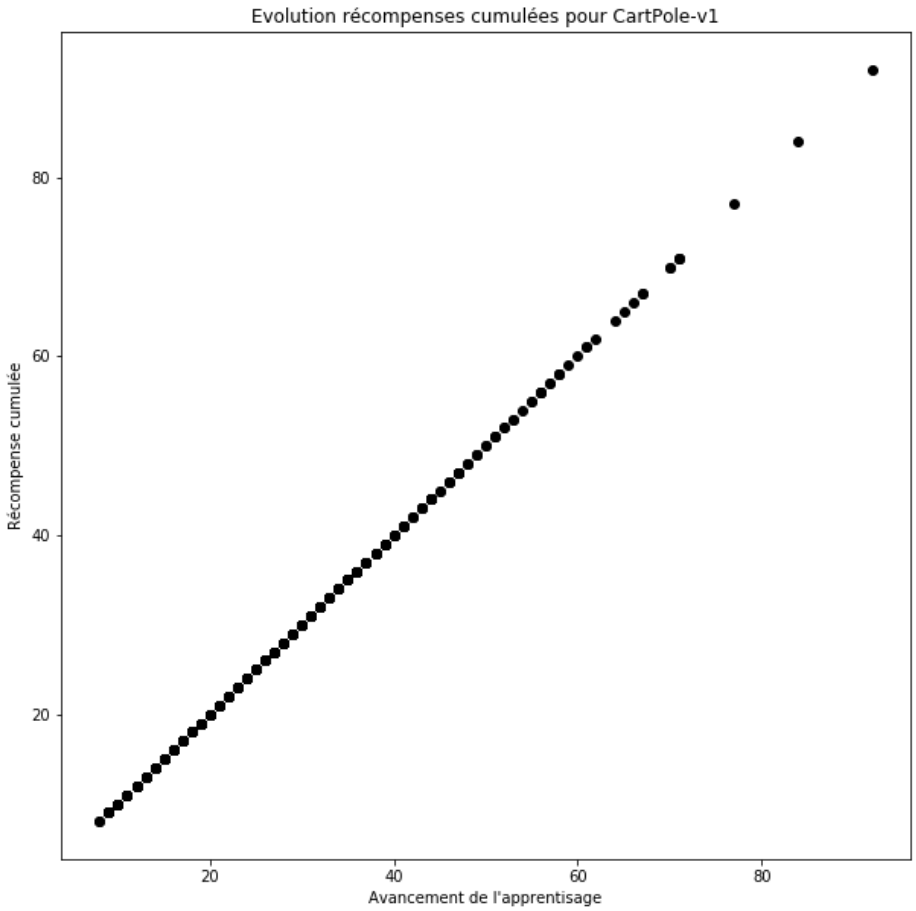
2.1.2 Question 2 – Évaluation des récompenses

NB : En plus de l'affichage demandé, nous avons pris l'initiative de créer une visualisation de la récompense cumulée au fil des épisodes dénotant en quelque sorte le « succès » de la session d'apprentissage.

Nous avons opté pour un affichage `matplotlib`, avec *mean-pooling* pour alléger le graphe en cas de grand nombre d'épisodes lorsque nécessaire.

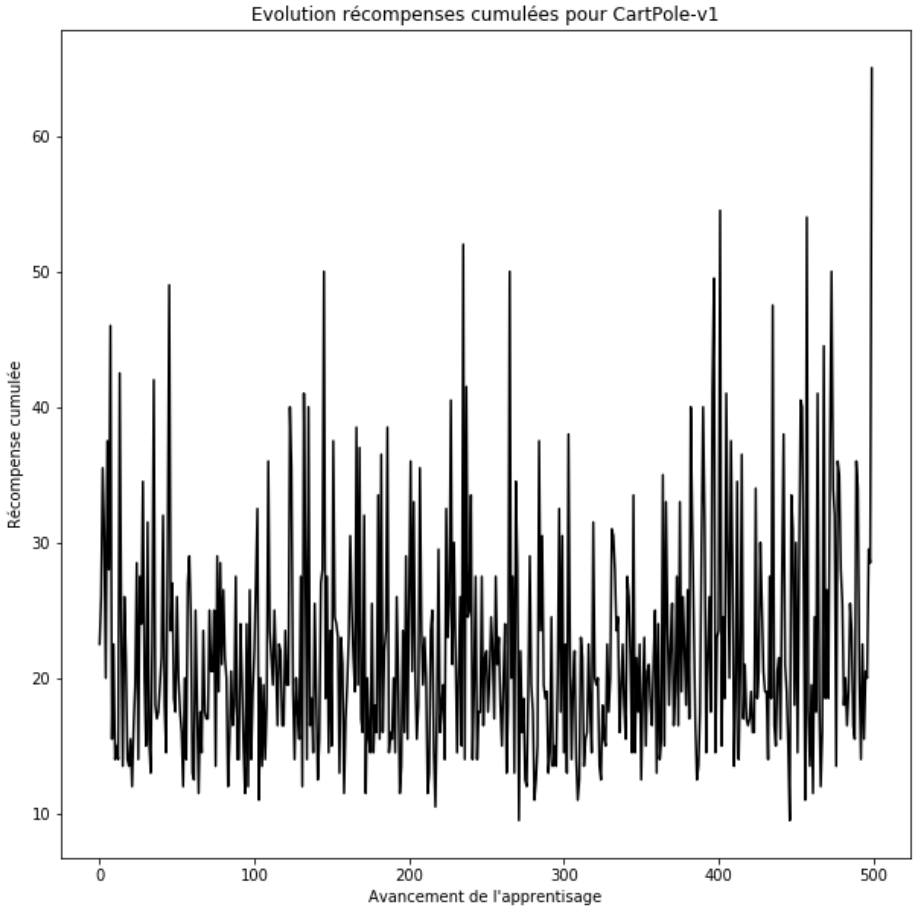
Dans notre premier cas, soit celui d'un agent aléatoire sur l'environnement *CartPole-V1*, le graphe des récompenses cumulées en fonction du nombre d'interactions aboutit forcément à un nuage de points suivant la fonction $f(x) = |x|$.

On constate peu d'*outliers* (qu'on considèrera ici comme ayant $x > 60$), en raison du caractère aléatoire des décisions.



Ce résultat est sans surprise, puisque toute interaction entraîne une récompense de 1 dans cet environnement.

La deuxième visualisation n'est pour le moment pas très utile non plus, puisque notre agent n'apprend pas et agit de façon aléatoire.



Ces deux affichages prendront leur pertinence plus tard dans le développement, lorsque notre agent commencera à apprendre de son environnement.

Il nous indiqueront respectivement la rapidité d'apprentissage des épisodes individuels, et l'efficacité générale de l'apprentissage en fonction du temps passé (*i.e.* des épisodes).

2.2 Experience replay

2.2.1 Question 3 – Buffer de stockage d'expériences

Le code du buffer *experience replay* est dans le fichier `deeprl_agent.py`, plus particulièrement la fonction `experience` de la classe `Agent`.
Cette question marque le début du code des notebooks jupyter.

2.2.2 Question 4 – Sampling de mini-batch

À partir de cette question, nous basculons sur un système de notebooks *Jupyter* pour une implémentation plus rapide des tests.
La récupération de mini-batch de données du buffer se fait donc dans le fichier `experience_replay.ipynb`.

2.3 Deep Q-learning

L'implémentation du Q-learning se fait dans la continuation de la partie précédente, donc dans le fichier `experience_replay.ipynb`. Sauf indication du contraire, c'est au sein de ce notebook que le reste du TP se déroulera.

2.3.1 Question 5 – Réseau de neurones

Pour notre premier exemple, on construit donc un réseau avec 4 neurones d'entrée et 2 neurones de sortie.

En effet, dans l'environnement *CartPole-v1*, l'observation est de type `Box(4,)` et le domaine des actions est un `Discrete(2)` (en l'occurence 0 et 1 pour bouger à gauche et à droite respectivement).

Ces dimensions expliquent la taille de notre réseau.

2.3.2 Question 6 – Approximation des Q-valeurs

Les deux stratégies d'exploitation proposées sont implémentées dans la classe *Agent*, qui est dotée des fonctions `epsilongreedy(self, ob, epsilon)` et `boltzmann(self, ob, tau)` qui renvoient l'action choisie, qui est soit une action aléatoire d'exploration, soit le choix déterminé par la politique d'après les Q-valeurs prédites par le réseau.

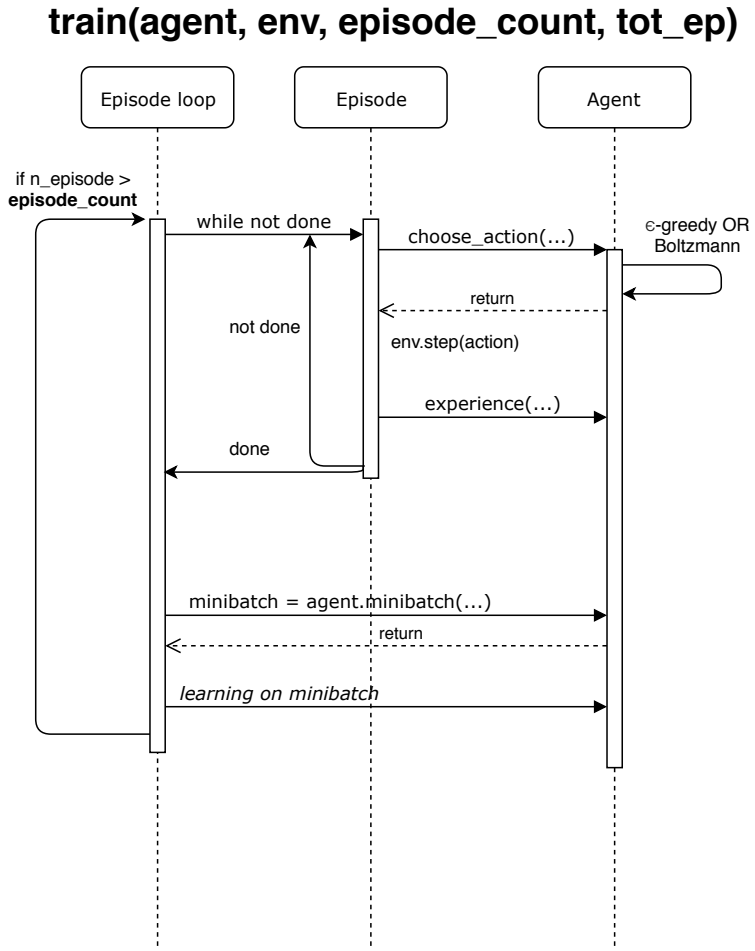
Pour la politique ϵ-greedy, il s'agit de la meilleure action avec une probabilité $(1 - \epsilon)$. Pour Boltzmann, le choix est stochastique mais tend malgré tout à choisir la Q-valeur maximale le plus souvent.

2.3.3 Question 7 – Apprentissage

À ce stade du développement, notre agent utilise désormais toute une panoplie de fonctions ; à chaque fois qu'il doit choisir une action, il fait appel à `choose_action(ob, reward, done)`, qui détermine les Q-values avant de déterminer (selon la méthode d'exploration actuelle) l'action choisie.

Après avoir effectué son action, celle-ci est appliquée à l'environnement par le biais de la fonction `step()`, après quoi on fait appel à l'*experience replay*. Pour ce faire, on utilise la fonction `experience()` de la classe *Agent*.

À la fin de notre épisode, on utilise `minibatch()` pour récupérer des expériences passées de l'agent, puis on apprend sur chacune des expérience de cette mini-batch.



Ci-dessus, un diagramme de séquence simplifié de l'entraînement d'un Agent.

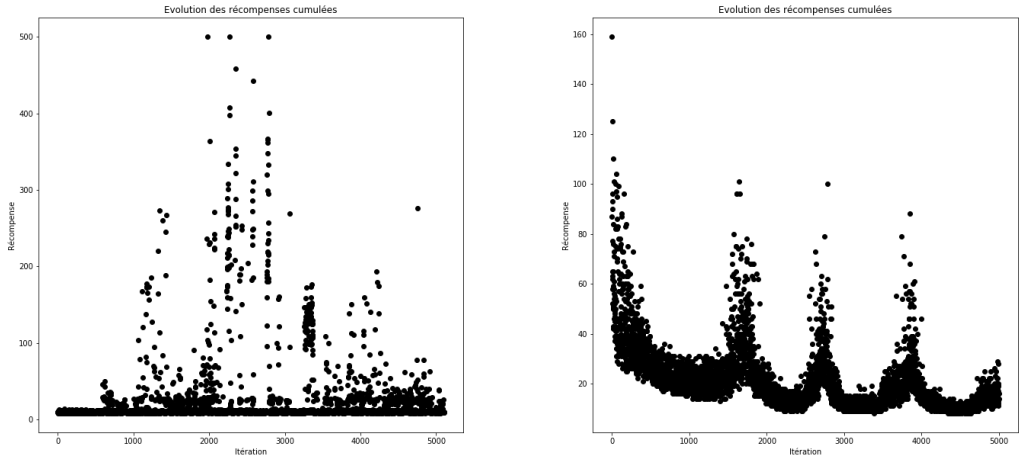
2.3.4 Question 8 – Target Network

Le *target network* a été implémenté selon la solution « N-étapes ». On calculera donc la Q-valeur ciblée avec ce nouveau réseau, qui sera mis à jour toutes les N itérations d'apprentissage.

Ce N se traduira chez nous par l'hyper-paramètre `target_sync`, et la fonction `sync_target()` d'*Agent* sera appelée en temps voulu.

2.4 Étude des résultats finaux sur CartPole

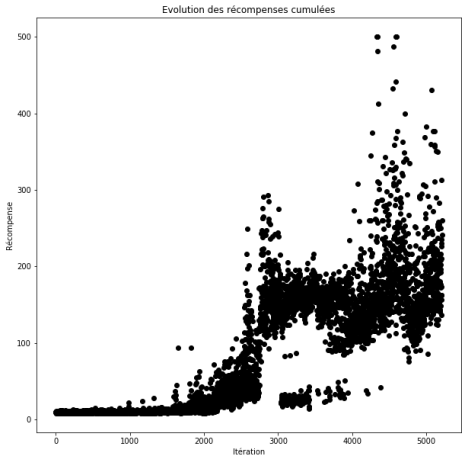
L'apprentissage de notre agent sur l'environnement *CartPole-v1* s'est soldé par un certain succès. Nos premiers pas donnaient parfois un comportement peu optimisé, parfois une politique plus solide, mais après quelques temps et une session d'apprentissage plus longue, on commence à distinguer une réelle progression.

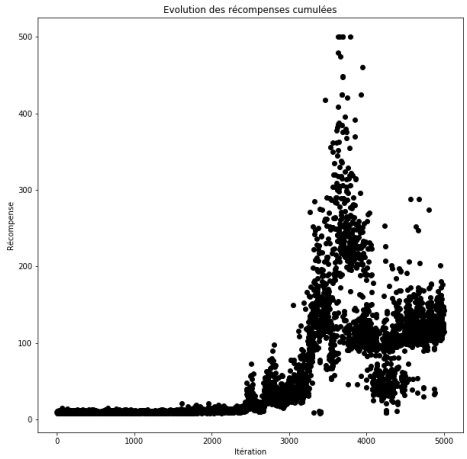


Ci-dessus, quelques-un de nos premiers succès représentés graphiquement. Sur seulement 5 000 itérations, l'agent peut atteindre une récompense de 500 (qui est la récompense maximale de l'environnement) plusieurs fois.

Cependant, on ne constate pas de réelle amélioration sur le long terme de son comportement. L'évolution de sa performance semble au mieux erratique, et on ne pouvait pas se satisfaire d'un tel agent.

Après quelques modifications des hyper-paramètres, toujours sur 5 000 itérations, on aboutit à de meilleures tendances :





Cette fois, l'agent semble mieux « retenir » ses bonnes performances passées. On constate déjà une tendance qu'on retrouvera par la suite, celle de « pallier » de récompenses cumulées dans le graphique.

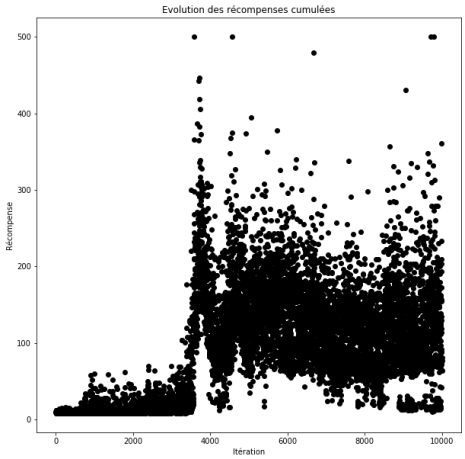
En effet, l'agent semble avoir des performances variables, même après un certain apprentissage, mais les récompenses des épisodes tombent le plus souvent dans des fourchettes déterminées.

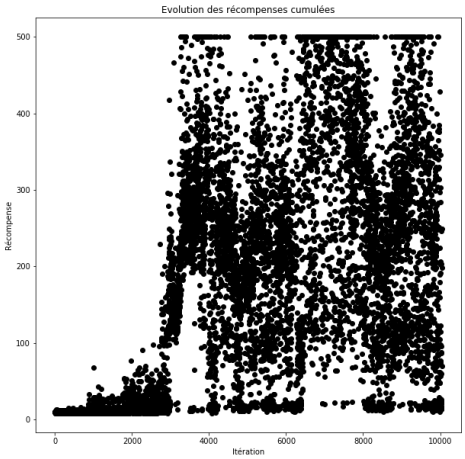
Il est difficile de déterminer ci ces palliers sont dûs à la nature de l'environnement ou, au contraire, si l'agent lui-même en est la source.
Quoi qu'il en soit, nous avons ensuite réalisé de nouvelles sessions d'apprentissage, cette fois en passant à 10 000 itérations.

Ces nouvelles sessions ont non seulement plus d'itérations, mais sont aussi les premières à implémenter le *target network* de la Question 8.

Là où les résultats ci-dessus tendent à l'instabilité (en particulier, le deuxième cas présente un apprentissage jusqu'à 500 de récompense suivi d'une retombée assez décevante), l'ajout du deuxième réseau permet de viser une cible stable pendant au moins `target_sync` itérations.

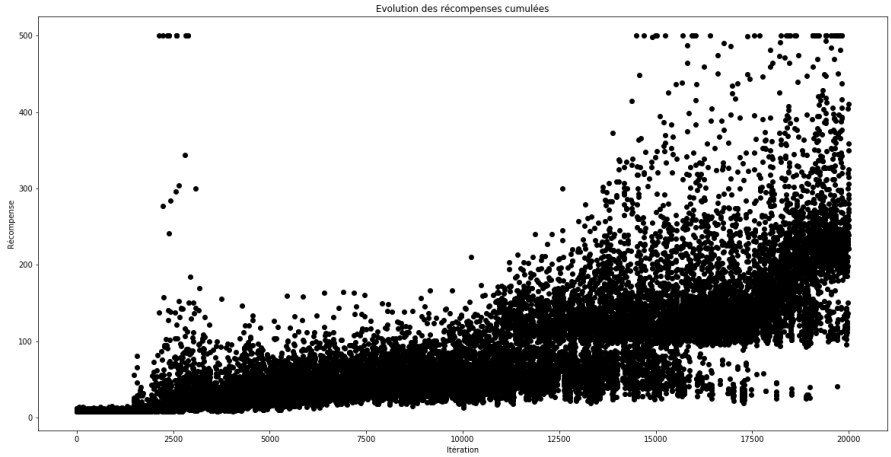
Ces deux améliorations nous permettent d'obtenir des comportements plus satisfaisants, qui sont illustrés ci-dessous.

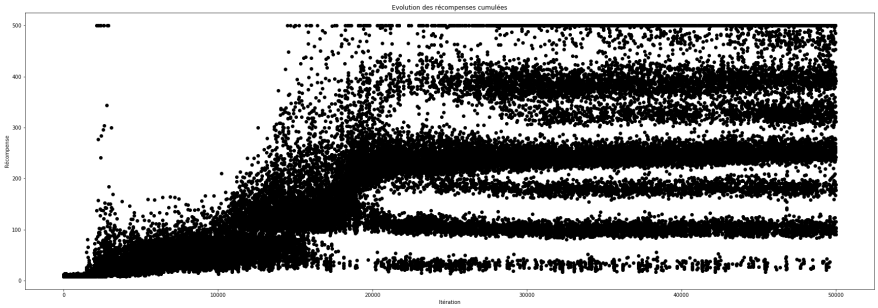




L'ensemble de ces résultats ont été obtenus avec $\eta = 0.01$ et un batch_size de 32. Le phénomène de palliers est ici peu visible, mais se remarque plus facilement sur une session encore plus longue.

Ci-dessous, notre dernière session, de respectivement 20 000 et 50 000 itérations :





Les deux images représentent la même session, à différents niveaux d'avancement. On constate mieux le phénomène de palliers, et, en particulier, après quelques dizaines de milliers d'itérations, l'agent obtient des résultats plutôt déterminés.

Beaucoup d'épisodes obtiennent un score de 500, et la plupart des autres obtiennent soit zéro (probablement à cause d'une exploration déstabilisant le bâton), soit une récompense contenue dans une fourchette assez restreinte, représentant (d'après notre hypothèse) les politiques de l'agent.

Ces résultats finaux nous paraîssent assez satisfaisants mais, le but ultime de cet agent étant d'évoluer dans l'environnement *Atari Breakout*, il nous faut adapter légèrement notre système pour opérer une transition vers ce nouveau jeu.

3 Breakout Atari

Sauf indication du contraire, nous évoluons toujours dans le même notebook, soit `experience_replay.ipynb`.

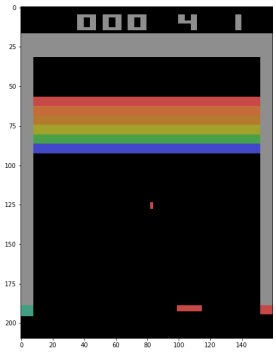
3.1 Question 1 – Construction de l'environnement

L'ajout de ce nouvel environnement se fait dans un bloc dédié (présent juste sous le bloc d'initialisation de *CartPole-v1*). On utilise le wrapper *AtariPreprocessing* pour obtenir les paramètres décrits dans l'énoncé.

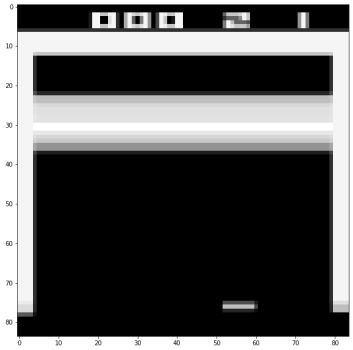
En l'occurence, `grayscale_obs=True`, `frame_skip=4` et `screen_size=84`.

De plus, afin d'avoir une meilleure appréhension de la trajectoire et vitesse de la balle, nous utilisons aussi *FrameStack*, qui va réunir les 4 dernières frames jouées.

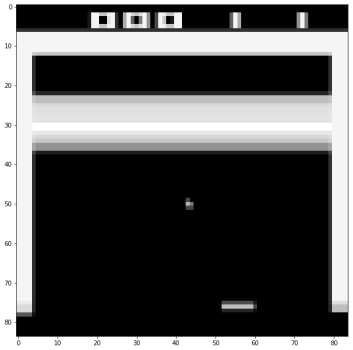
Ainsi nous passons d'une image RGB de 210×160 pixels :



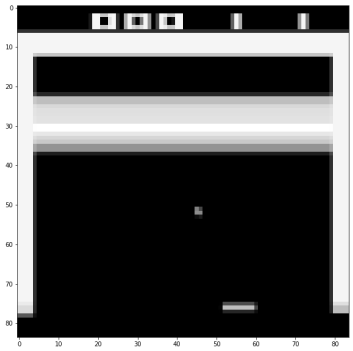
A une série de 4 frames de 84×84 pixels, en noir et blanc :



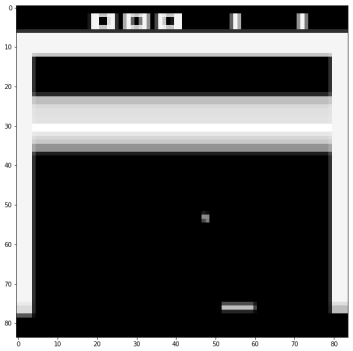
(a) Frame 1



(b) Frame 2



(c) Frame 3



(d) Frame 4

3.2 Question 2 – Adaptation du code

Peu de commentaires sont à faire sur cette adaptation. Les détails du code sont comme toujours présents dans le notebook.

3.3 Question 3 – Remplacement par un CNN

Afin de simplifier l'implémentation du réseau convolutionnel, nous avons ré-utilisé du code produit lors du TP1 d'intelligence bio-inspiré. Durant celui-ci, nous avions réalisé pour une question bonus un tel réseau. Sa classe dédiée est contenue dans un bloc du notebook.

On peut y retrouver le paramétrage du réseau, qui est le suivant :
⟶ Couche 1
- outchannel = 32
- kernel = 8
- stride = 4

⟶ Couche 2
- outchannel = 64
- kernel = 4
- stride = 2

⟶ Couche 3
- outchannel3 = 64
- kernel3 = 3
- stride1 = 1

3.4 Question 4 – Optimisation des hyper-paramètres

La partie « Methods » de l'article, à partir de la page 6, détaille certains éléments et hyper-paramètres, dont le pré-processing (déjà aligné sur l'article depuis la Question 1).

La structure du réseau convolutionnel, détaillée dans la Question 3), est elle aussi déjà calquée sur celle des chercheurs, tout comme la taille des mini-batch de 32 et le taux d'update du *target network* (10 000).

La taille du buffer d'*experience replay*, elle, doit être adaptée. Nous utilisions jusqu'à présent seulement 10 000 de taille maximale, comparée aux 1 000 000 de l'article.

Ayant ajusté nos paramètres, nous tentons alors de lancer nos sessions d'appren-

tissage sur ce nouvel environnement.

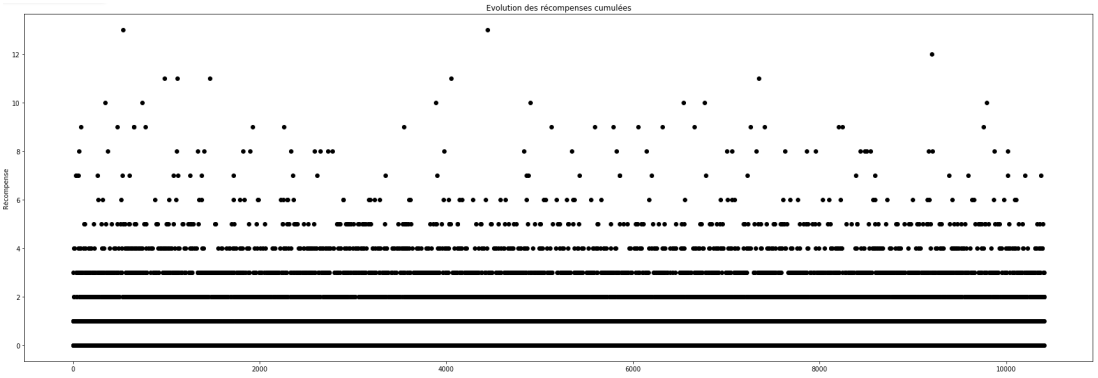
3.5 Question 5 – Vidéos du comportement et réseau

Malgré les hyper-paramètres optimisés, notre agent semble avoir des difficultés lors de son apprentissage. La plupart de nos premières sessions se terminant avec l'agent collé contre le mur (ce qui s'avère fonctionner dans certains cas particuliers), nous avons cherché une parade à ce comportement peu optimisé.

Nos premières tentatives consistaient à favoriser l'exploration et une vision plus long-terme de l'apprentissage.

En premier lieu, nous avons tenté d'utiliser la stratégie ϵ-greedy avec un ϵ non pas fixé à 0.05 mais décroissant en partant de 1, comme l'a été fait dans l'article référencé précédemment.
Ainsi, l'agent est forcé à une exploration aléatoire durant ses premiers pas d'apprentissage. Cette stratégie n'a malheureusement pas porté ses fruits.

Toutes nos tentatives d'apprentissage se sont soldées par un échec, l'agent ayant atteint un grand maximum de 13 points. Il semble s'être coincé dans le premier comportement augmentant légèrement sa récompense, en se collant systématiquement au côté de l'espace de jeu.



Malgré tout, nous avons enregistré comme demandé des vidéos du comportement de l'agent dans le dossier `vid`, et nos modèles de réseaux de neurones dans le dossier `model`.

Ces performances peu satisfaisantes auraient potentiellement pu être améliorées avec plus de temps, ou tout du moins plus de temps d'apprentissage.