# RL Paper implementation Report 1

kevin-brian.n-diaye

December 4, 2023

## 1 Introduction

Nous allons un papier de recherche venant de DeepMind s'intitulant: Human-level control through deep reinforcement learning.

Il se base sur un agent de QLearning avec epsilon variable mais utilise un réseau de neurones à pour faire ses inférences.

Le papier montre que le modèle atteint des facultés surhumaines à l'issue de l'entrainement.

Cependant, malgré une implémentation fidéle du modèle, je n'ai pas réussi à obtenir des résultats similaires aux chercheurs de chez DeepMind.

## 2 Modèle

Ce modèle contient un réseau de neurones composé de trois couches de convolution suivi de 2 couches Dense pour donner une action dans notre espace de jeu. Le réseau de neurones est doté d'un clone qui servira pour la descente de gradient.

Pour pouvoir apprendre, il se sert donc de sa mémoire et de son modèle jumeau donc les paramètres sont mis à jour à intervalles réguliers.

Lorsqu'il faut effectuer une descente de gradient pour entrainer le réseau, la mémoire rentre en jeu et fourni une quantité fixé de tuple contenant:

1. L'état s à l'instant t: $s_t$
2. L'action prise à l'instant t: $a_t$
3. La récompense liée à l'action: $r_t$
4. Le nouvel état engendrée: $s_{t+1}$

Le rôle du modèle clone va être d'essayer d'inférer la meilleure action $a_t$ pour maximiser la récompense. Cela nous donnera des labels sur lesquels on peut effectuer une descente de gradient (RMSProp dans ce papier).

Comme vous venez de comprendre, la sauvegarde des états précédents dans une replay memory est cruciale pour l'apprentissage du réseau de neurones. Cependant, au fur et à mesure, que la taille de la mémoire augmente (fixée à 1e6 dans le papier), notre consommation en VRAM augmente aussi. On se heurte alors au soucis de l'hardware qui nous pousse à vouloir compresser cette mémoire et revoir les hyperparamètre pour mieux s'adapter à nos contraintes.

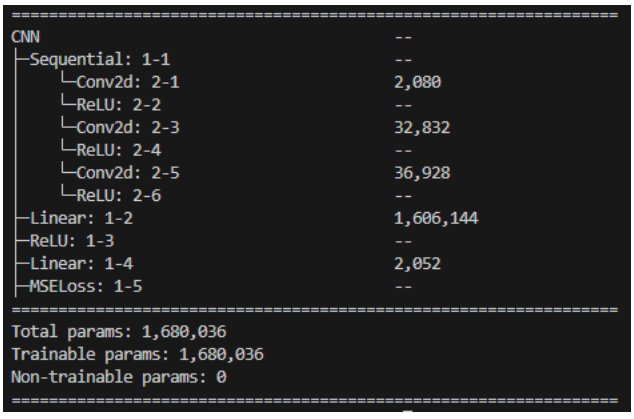


Figure 1: Structure du réseau de neurones

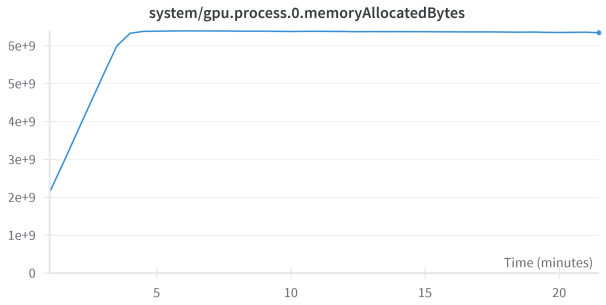


Figure 2: Mon GPU avec 6Go de VRAM au bout de 20mn

Le papier décrit alors une méthode de preprocessing des états pour pouvoir conserver de la mémoire. Les états sont récupérés tous les 4 frames et l'action est répétée sur les 4 prochaines. On note que Gym, la librairie que nous utilisons, le fait déjà dans les jeux utilisés. Les états qui sont des images sont ensuites passées en grayscale puis rétréci sur (84, 84).

On passe donc de (210, 160, 3, 4) pour quatres écrans de 210x160 en rgb à (84, 84, 1, 4).

D'autres méthodes comme le sampling sur frames pairs et impairs sont expliqué mais ne sont plus pertinentes grâce à Gym.

## 2.1 Hyperparamètres

Les hyperparamètres du papier ont été repris tout en diminuant la taille de la mémoire pour pouvoir maintenir l'entrainement.

```
RANDOM_SEED = 42
MINIBATCH_SIZE = 32
REPLAY_SIZE = 1e5
UPDATE_FREQUENCY = 10000
GAMMA = 0.99
RMS_LEARNING_RATE = 2.5e-4
RMS_GRADIENT_MOMENTUM = 0.95
RMS_ESP = 0.01
TAU = 0.005
MAX_EPOCHS = 1000
MAX_ACTION = 3600
# 6 (action per second) * 600 (seconds per game, 10mn of game max) = 3600 actions per game
```

Figure 3: Hyperparamètres reprenant ceux du papier

# 3 Expérimentation

Cette section va décrire les hyperparamètres qui ont été modifiés et pourquoi.

## 3.1 Fonction de perte

Le papier est parti sur une MSE mais des pertes qui seraient moins sensible aux outliers comme la SmoothL1Loss [1] pourraient mener à des modèles plus stables. Il est aussi mentionné que la perte est clip entre -1 et 1 pour obtenir un modèle plus robuste.

## 3.2 Update modèle jumeau

Le modèle jumeau peut se mettre à jour de deux façons:

1. Lui assigner les paramètre du modèle d'origine périodiquement
2. Utiliser la soft_update à chaque étape [2]

[1] https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html#torch.nn.SmoothL1Loss
[2] https://arxiv.org/pdf/1509.02971.pdf

# 4 Résultats

J'ai entrainé le modèle sur deux jeux:

1. Breakout-v5
2. Boxing-v5

Et voici les résultats.

## 4.1 Breakout-v5

Voici le premier entrainement complet obtenu sur Breakout. Tous les paramètres étaient ceux du papier:

1. Loss: MSE
2. Update: Périodique

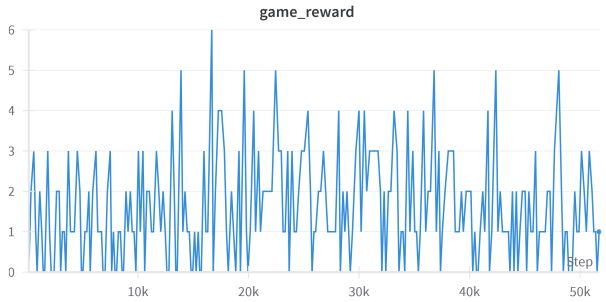


Figure 4: Peu d'apprentissage par ici

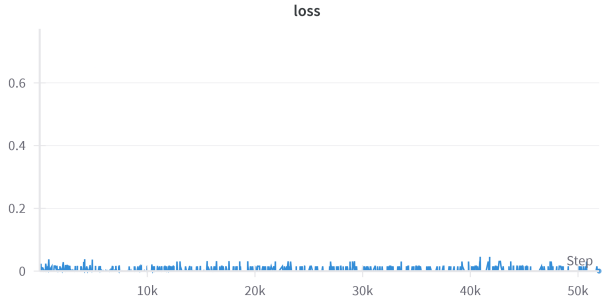


Figure 5: Loss = MSE, soft_update=False

## 4.2 Boxing-v5

Voici le premier entrainement complet obtenu sur Boxing. Tous les paramètres étaient ceux du papier:

1. Loss: MSE
2. Update: Périodique

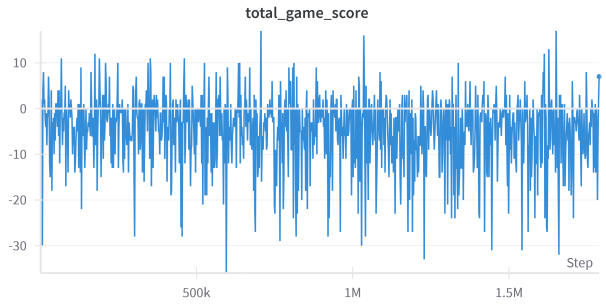


Figure 6: Loss = MSE

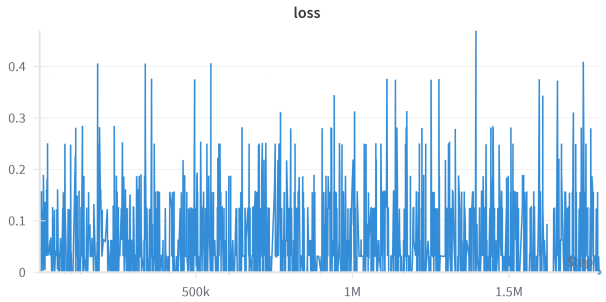


Figure 7: La loss varie violemment

# 5 Analyse et Conclusion

Etant donné que j'ai eu peu de résultats concluants sur Breakout, j'ai décidé d'observer le modèle en action. Je me suis rendu que le modèle se mettait dans un coin et ne bougeait pas de cette endroit jusqu'à la fin de la partie. Cela s'applique aussi à Boxing.

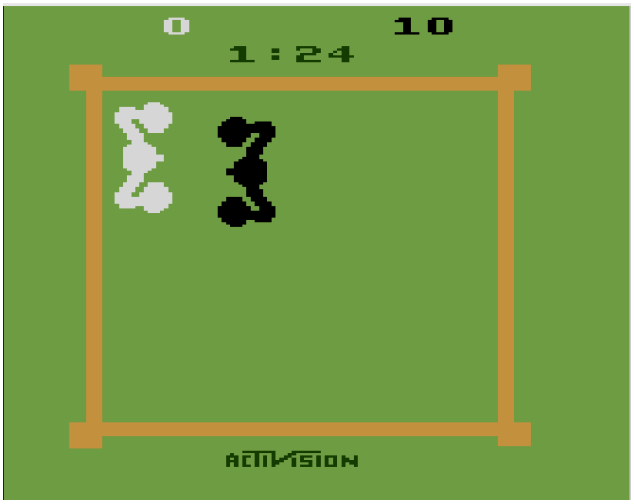


Figure 8: Le modèle (joueur blanc) est coincé

Ceci crée une situation où s'il bouge sa reward baisse car il prend des coups critiques durant son déplacement. Il est donc récompensé dans son comportement. La partie avec une reward relativement loin de 0 peuvent être expliqués par les quelques coups opportuns lancés depuis le coin.

On conclut donc qu'au vu du champs des possibles dans les environnement de jeux, le modèle prend énormément de temps à trouver un comportement risqué qui apporte gros.

Cela explique l'immense taille de la mémoire ainsi que les temps d'entrainements donnés dans le papier.

De plus, on remarque que les meilleurs résultats sont obtenus lorsque les techniques permettant un modèle plus robuste ne sont pas utilisées. Le modèle a plus de chance de réussir s'il fluctue grandement durant ses débuts.