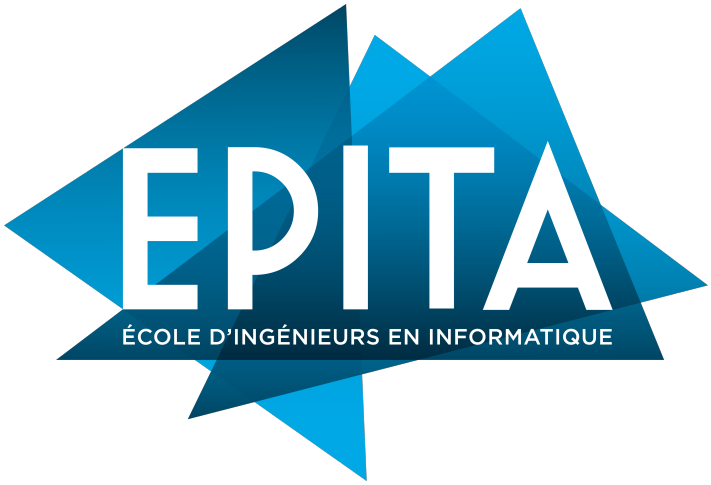
PONG DUAL DQN

Développement d'un agent d'apprentissage par renforcement pour jouer au Pong

Auteurs :
Eithan Nakache - Camil Ziane
Promotion 2025



EPITA
École pour l'Informatique et les Techniques Avancées

Table des matières

1 Introduction

Ce projet vise à entraîner un agent capable de maîtriser Pong grâce à un Deep Q-Network (DQN), combinant Q-Learning et réseaux de neurones pour prendre des décisions à partir d'images brutes. L'environnement de jeu est simulé à l'aide de la bibliothèque Gymnasium. Ce rapport décrit la méthodologie, les résultats et les défis rencontrés.

2 Méthodologie

Pour entraîner notre agent à jouer à Pong, nous avons utilisé **Gymnasium**, qui nous a fournit l'environnement *PongDeterministic-v4*.

2.1 Prétraitement des images

Comme les images brutes du jeu sont grandes et colorées, nous les avons transformées pour simplifier le travail de l'agent :

- Conversion des images en niveaux de gris.
- Redimensionnement à une taille plus petite (80x64), dans le but de retirer les parties inutiles de l'image telles que le score affiché.
- Normalisation des pixels (valeurs entre 0 et 1).

2.2 L'algorithme DQN

Le cœur du projet repose sur le **Deep Q-Network (DQN)**. Cet algorithme utilise un réseau de neurones pour prédire les meilleures actions à prendre. Les éléments clés incluent :

- Une stratégie *epsilon-greedy* : au début, l'agent explore avec des actions aléatoires, puis devient plus stratégique.
- Un *replay buffer* : l'agent stocke ses expériences (état, action, récompense, nouvel état) et les rejoue pour mieux apprendre.

2.3 Réseau de neurones et Dual DQN

Le réseau de neurones est conçu selon l'architecture **Dual DQN**, une amélioration du DQN classique. Cette architecture introduit une séparation entre deux aspects fondamentaux :

- **Stream de valeur ($V(s)$) :** il estime la qualité globale d'un état indépendamment des actions.
- **Stream d'avantage ($A(s,a)$) :** il mesure l'impact relatif de chaque action dans cet état.

La valeur Q pour une action donnée est alors calculée comme suit :

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a') \right)$$

Cette architecture permet de mieux gérer les états neutres (où toutes les actions ont un effet similaire) et améliore la stabilité de l'apprentissage.

2.4 Gestion des deux réseaux (Policy et Target)

Pour assurer une convergence stable, l'algorithme DQN utilise deux réseaux distincts :

- **Réseau principal (Policy) :** Entraîné pour prédire les valeurs Q des actions possibles à partir des expériences de l'agent.
- **Réseau cible (Target) :** Copie périodique du réseau principal, utilisé pour calculer les cibles stables lors de la mise à jour des poids.

Le réseau cible est synchronisé avec le réseau principal à des intervalles réguliers (par exemple, toutes les 1000 étapes). Cette approche réduit les oscillations et rend l'apprentissage plus robuste, car les cibles de mise à jour sont moins volatiles.

2.5 Avantages

L'implémentation du Dual DQN avec deux réseaux distincts présente plusieurs bénéfices :

- **Découplage des évaluations :** Meilleure séparation entre l'évaluation d'un état et celle des actions.
- **Stabilité accrue :** Grâce au réseau cible, les mises à jour des poids sont plus stables.
- **Apprentissage efficace :** Gestion des états neutres et meilleure généralisation dans des environnements complexes.

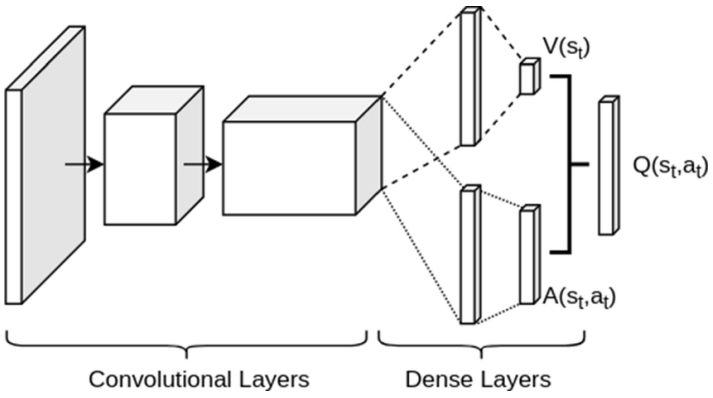


Figure 1 – Schéma de l'architecture Dual DQN avec séparation des streams

3 Résultats

L'agent a été entraîné sur une carte graphique RTX 4070 pendant environ 1h43. La Figure 2 présente l'évolution des performances au cours de l'entraînement avec les récompenses cumulées, le nombre d'étapes, la diminution de l'epsilon et la convergence de la perte totale.

Récompenses cumulées : Les récompenses cumulées augmentent rapidement entre les épisodes 100 et 300. L'agent progresse en apprenant à mieux interagir avec l'environnement. Après l'épisode 500, les récompenses se stabilisent autour de 20, indiquant que l'agent a acquis une stratégie efficace.

Nombre d'étapes : Le nombre d'étapes par épisode augmente progressivement jusqu'à atteindre un plateau après environ 300 épisodes. Cela signifie que l'agent prolonge les épisodes en prenant de meilleures décisions.

Epsilon et perte : L'epsilon, qui contrôle la balance entre exploration et exploitation, diminue au fil de l'entraînement, permettant à l'agent de se concentrer davantage sur les stratégies déjà apprises. En parallèle, la perte totale diminue fortement au début, puis converge vers des valeurs faibles, indiquant que le modèle est devenu stable et précis.

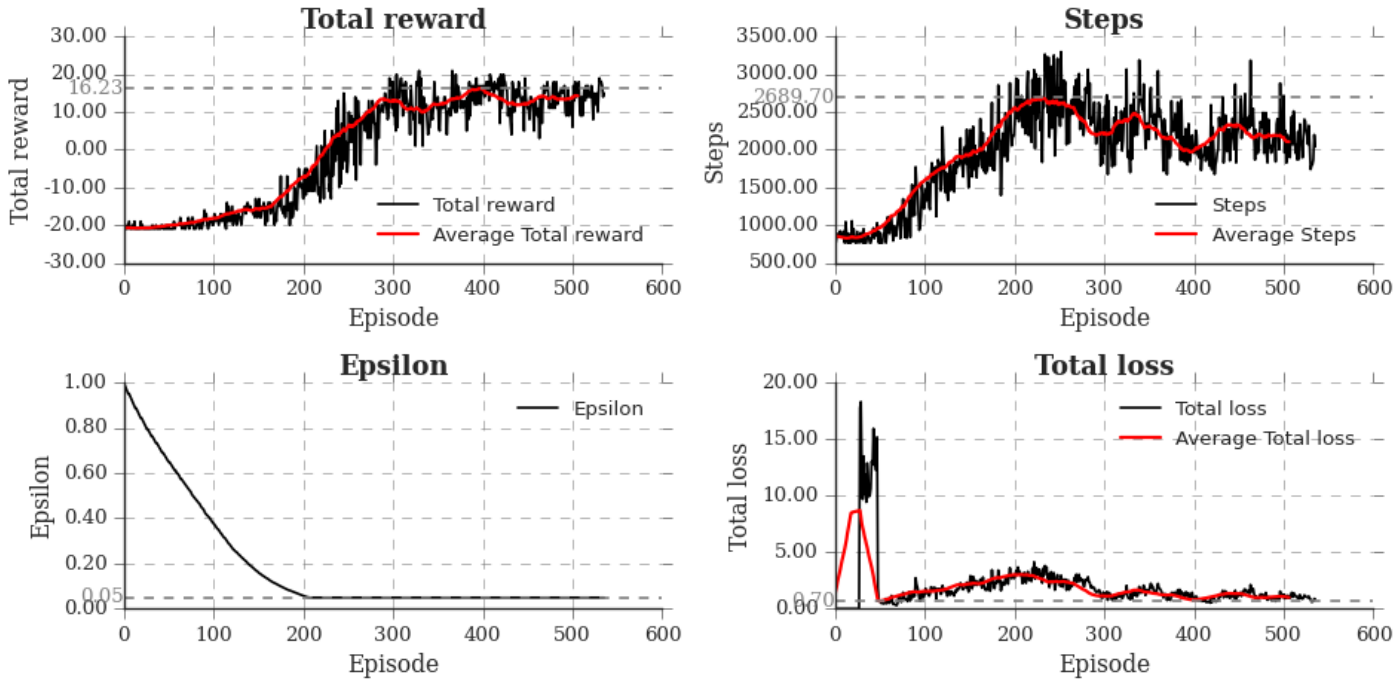


Figure 2 – Performances de l'agent au cours de l'entraînement : récompenses cumulées, nombre d'étapes, epsilon et perte totale.

Pour plus de détails consultez le dépôt GitHub.