École Centrale de Lyon

MOD 3.2 - Deep Learning IA
Option Informatique
Rapport

TD3 – Apprentissage par renforcement

Élèves :
Thibaud MIQUEL

Enseignant :
Thomas DUBOUDIN

Table des matières

Table des figures

1 Présentation du TD

1.1 Objectif du TD

L'objectif de ce TD est de mettre en oeuvre l'apprentissage par renforcement. Nous allons utiliser le toolkit Gym qui propose plusieurs types d'environnements de simulation. Nous nous intéresserons en particulier à l'environnement Cartpole et à celui du jeu Breakout d'Atari.

1.2 Présentation des 2 sujets

1.2.1 Cart pole

CartPole, également connu sous le nom de pendule inversé, est un jeu dans lequel vous essayez d'équilibrer une perche le plus longtemps possible. On suppose qu'à l'extrémité de la perche, il y a un objet qui la rend instable et très susceptible de tomber. Le but de cette tâche est de déplacer le chariot de gauche à droite de manière à ce que le poteau puisse rester en équilibre (dans un certain angle) le plus longtemps possible.

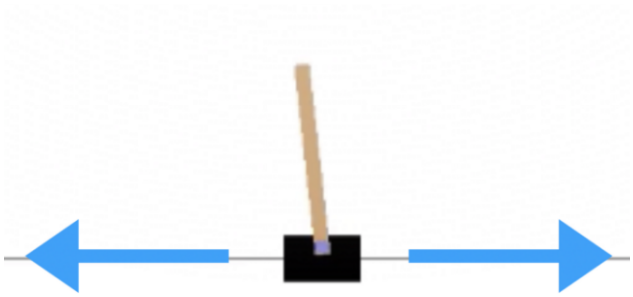


FIGURE 1 – Interface de CartPole

Pour contrôler le chariot on utiliser un apprentissage par renforcement Q-learning basé sur un réseau neurones avec une simple couche cachée du type fully-connected layer. L'entrée du réseau sera directement l'état du chariot à l'instant t soit : la position du chariot, la vitesse du chariot, la angle du pendule et la vitesse du pendule au sommet.

1.2.2 Atari - Break out

Breakout est un jeu d'arcade, commercialisé par Atari Inc. en 1976.

Dans Breakout, plusieurs lignes de « briques » sont alignées dans le haut de l'écran de jeu. Une balle traverse l'écran, rebondissant sur les côtés et le haut. Lorsque la balle touche une brique, elle rebondit et la brique disparaît. Le joueur possède une raquette qu'il peut déplacer latéralement afin de faire rebondir la balle vers le haut. Il perd une vie quand la balle touche le bas de l'écran.

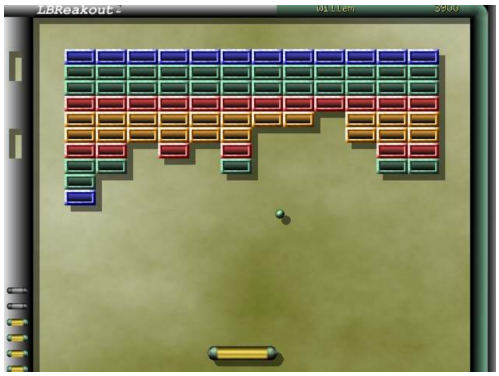


FIGURE 2 – Interface du jeu LBreakout (variante de Breakout)

Le contexte d'utilisation d'apprentissage par renforcement ici est qu'à partir d'une entré qui correspond à l'observation d'une séquence de 4 images consécutives afin d'avoir l'information de mouvement d'obtenir en sortie l'action à prendre, ie : se déplacer à gauche ou a droite.

Pour effectuer cela on utilisera ici ce qu'on appelle un DQ-learning (Deep Q-learning) signifiant que l'on aura plusieurs couches au réseau de neurones basé sur un réseau convolutif afin de pouvoir traiter directement les images.

2 Utilisation d'apprentissage par renforcement

2.1 Principes du reinforcement learning

L'apprentissage par renforcement consiste, pour un agent autonome (robot, etc.), à apprendre les actions à prendre, à partir d'expériences, de façon à optimiser une récompense au cours du temps.

L'agent est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure à l'agent une récompense, qui peut être positive ou négative. L'agent cherche, au travers d'expériences itérées, une stratégie optimale, c'est à dire qui maximise la somme des récompenses au cours du temps.

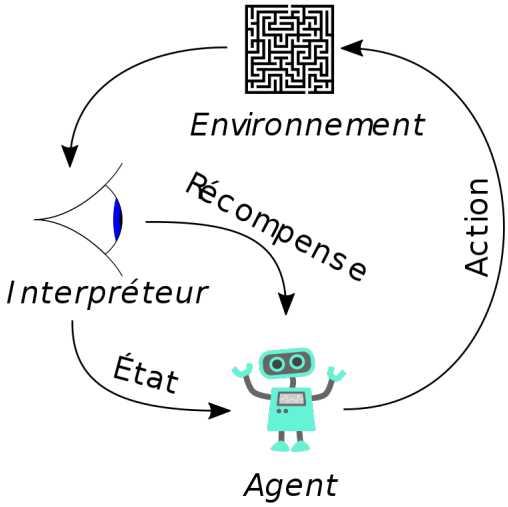


FIGURE 3 – Schématisation du fonctionnement de reinforcement learning

2.1.1 Spécificité du Q-learning et du DQ-learning

le Q-learning est une technique d'apprentissage par renforcement. Cette technique ne nécessite aucun modèle initial de l'environnement. La lettre 'Q' désigne la fonction qui mesure la qualité d'une action exécutée dans un état donné du système.

Cette méthode fonctionne par l'apprentissage d'une fonction notée Q(s,a) qui permet de déterminer le gain potentiel, c'est-à-dire la récompense sur le long terme, apportée par le choix d'une certaine action "a" dans un certain état "s" en suivant une stratégie optimale. Lorsque cette fonction Q est connue ou apprise par l'agent, la stratégie optimale peut être construite en sélectionnant l'action à valeur maximale pour chaque état, c'est-à-dire en sélectionnant l'action a qui maximise la valeur Q(s,a) quand l'agent se trouve dans l'état s.

Dans un environnement reinforcement learning, pour un état (S) et une action (A), il y aura une valeur Q associée qui est maintenue dans une table. Mais lorsque le nombre d'états possibles est trop important, les valeurs Q associées à ces paires état-action explosent en nombre. C'est pourquoi on peut remplacé cette table Q par une fonction comme un réseau de neurones profond. D'où le nom de Deep-Q-Learning.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

FIGURE 4 – Pseudo code du Deep Q learning

2.2 Présentation des fonctions communes aux 2 sujets

- **ReplayMemory :** Cette classe permet de rejouer la mémoire afin d'effectuer le nouveau mouvement du chariot. les fonctions sont l'initialisation, l'addition de la nouvelle position du chariot, le renvoie de la longueur de la mémoire, l'échantillonnage d'un batch de la mémoire.
- **Agent :** Cette classe va permettre la gestion de l'agent (ici du chariot) afin de l'initialiser, de sélectionner l'action à effectuer ainsi que le processus d'entraînement du réseau de neurones.
  - **init :** On initialise tous les paramètres de l'agent, les paramètres d'entraînement, les paramètres de sélection d'une action et l'initialisation du réseau de neurones. Le seul argument ici est l'environnement.
  - **select action :** On décide dans cette fonction si l'on va utiliser l'action prédite ou si on réalise une action aléatoire. Ce choix se fait en fonction d'un seuil, si une valeur tirée aléatoirement entre 0 et 1 est supérieure au seuil défini alors on sélectionne l'action renvoyée par la stratégie (le réseau de neurones) qui a le meilleur score. Sinon on renvoie une action aléatoire. Le seul argument ici est l'état du chariot.
  - **optimize model :** Dans cette fonction on va optimiser les paramètres du réseau de neurones. Le déroulement est le suivant : à partir d'un batch d'un triplet donné : state, action, reward (pris dans dans le tuple : Transition stockant tous ces triplets plus le next_state), on va d'abord calculer Q pour l'état courant à partir state et action dans le triplet précédent. On calcule ensuite Q prédit de l'état suivant. Enfin on calcul le Q cumulé attendu à partir du Q prédit que l'on vient de calculer et de la reward de l'état actuel, ce cumulé correspondant à l'état attendu. On utilise ensuite la perte de huber et on compare Q prédit et Q cumulé puis on optimise le sparamètres du réseau de neurones via une descente de gradient.
  - **train policy model :** Ici on réalise la fonction l'entraînement du réseau de neurones sur tous les épisodes.
  - **test :** Ici on va sélectionner l'action à faire à chaque épisode et l'envoyer à l'environ-

nement afin de visualiser l'action.

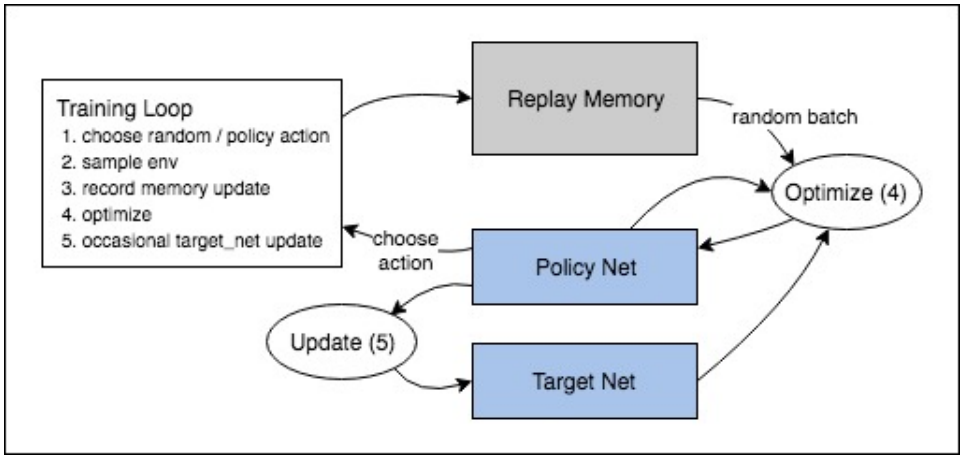


FIGURE 5 – Diagramme illustrant tout le processus d'entraînement

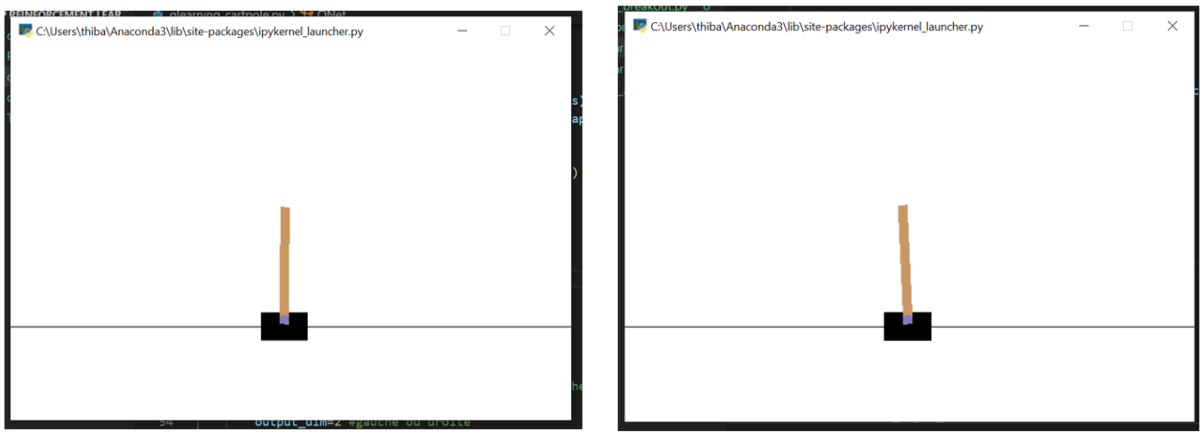
2.3 Présentation des fonctions spécifiques aux 2 sujets

- **Cart pole - QNet :**
  - **init :** Définition d'un réseau avec une couche cachée de 256 neurones. On a 4 entrée et 2 sorties.
  - **forward :** Passe avant du réseau de neurones avec des fonctions d'activation linéaire.
- **Breakout - DQN :**
  - **init :** Définition du réseau avec 3 couches de convolution chacune suivie d'une batch normalization. On utilise des filtres de taille 5 pixels. On utilisera 16 filtres pour la première couche, 32 filtres pour la deuxième, 64 pour la troisième et on fini par une couche fully connected. On a ici 4 images en entrée et 2 sorties.
  - **forward :** Passe avant du réseau de neurones avec une fonction d'activation linéaire pour la couche fully-connected.
- **Breakout - Agent :**
  - **process :** Cette fonction permet de convertir les images RGB de breakout en image de nuance de gris

2.4 Résultats obtenus et observations

2.4.1 CartPole

Concernant CartPole on peut observer comment se "corrige" le chariot afin de garder l'équilibre. En faisaint varier le nombre d'épisode on peut voir cette séquence pendant plus ou moins longtemps, pour des valeurs importantes de nombre d'épisodes >1000 il devenait nécessaire d'effectuer les calculs sur GPU.



(a) (b)

FIGURE 6 – Chariot dans 2 états différents dans un même test

2.4.2 Breakout

La réalisation de l'apprentissage renforcé concernant Breakout n'a pas pu être finalisé car l'environnement Breakout proposé par gym ne fonctionne pas... Malgré un passage à des versions antérieures de gym ou l'utilisation de différentes versions de l'environnement Breakout je n'ai pas réussi à faire apparaître cet environnement. Même indépendamment de l'algorithme d'apprentissage apr renforcement en essayant simplement de faire apparaître l'environnement Breakout cela ne fonctionne pas...

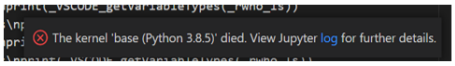


FIGURE 7 – Message d'erreur de VSC pour l'environnement Breakout

3 Conclusion

Pour conclure ce TD on a pu voir qu'avec l'environnement CartPole de gym l'apprentissage par renforcement fonctionne particulièrement bien pour maintenir le pendule en équilibre sur le chario, notamment grâce au choix de stratégie optimale par un réseau de neurones à une seule couche cachée. Cependant pour effectuer les calculs il est préférable voir nécessaire d'utiliser un GPU afin d'accélérer le temps de calcul.

Malgré le non fonctionnement de breakout on peut imaginer que l'utilisation d'un réseau de neurones profond à partir de couche convolutionnelle pour traiter en entrée 4 images consécutives correspondant à l'état de l'agent dans son environnement est la solution adaptée comme stratégie optimale d'apprentissage.

Malgré les dysfonctionnements nous avons pu ici programmer avec succès un algorithme d'apprentissage par renforcement de type DQN à partir de pytorch.