Laboratoire 2

Réseau de neurones profond avec apprentissage par renforcement (DQN)

Durée : 3 séances

Christophe Lamarche

Mise-à-jour: 23 mai 2021

1 Histoire

"**Les environnements virtuels et les jeux sont à notre avis la plateforme parfaite pour faire le développement et l'évaluation d'algorithme d'apprentissage machine.**"

Demis Hassabis (Co-fondateur et CEO de DeepMind) (Krieg et al., 2017)

La quête de résoudre un jeu de société a longtemps intéressé les mathématiciens. Le jeu d'échecs a été considéré comme un bon défi pour la résolution, car la quantité de mouvements possibles dans une partie étaient supérieurs aux capacités des ordinateurs de l'époque. (P., 2019) L'algorithme *minimax* de John Von Neumann (Von Neumann and O., 1953) a permis la première résolution du jeu d'échecs pouvant rivaliser avec un joueur moyen au milieu des années 1950.

La montée en popularité de l'apprentissage machine à l'aide des réseaux de neurones a permis à une collaboration avec l'apprentissage par renforcement. Cette alliance a produit des agents surpassant l'entendement humain. Pensons à *AlphaGo* de *DeepMind* qui a défait en 2016 le champion mondial de Go, Lee Sedol (Krieg et al., 2017). Ou bien, l'équipe d'*OpenAi* ayant réussi à produire un agent pouvant rivaliser avec des professionnels aux jeux vidéo *Dota 2* (OpenAI, 2017) et qui cible présentement le mode collaboratif 5 contre 5 de ce même jeu (OpenAI, 2019).

1.1 Exposé du problème

Pour le prochain événement d'un club étudiant, on vous confie le projet de produire un modèle pouvant rivaliser avec des participants au jeu *Breakout* de la plateforme *Atari*.

Mieux connu sous le nom *Atari Breakout*, ce jeu fut conçu originalement pour les arcades et fut exporté sur la console *Atari 2600*. Le but du jeu est simple : détruire le maximum de briques colorées en renvoyant une balle à l'aide d'une plateforme. Ce jeu fut conçu par Nolan Bushnell, Steve Bristow et Steve Wozniak en 1976.

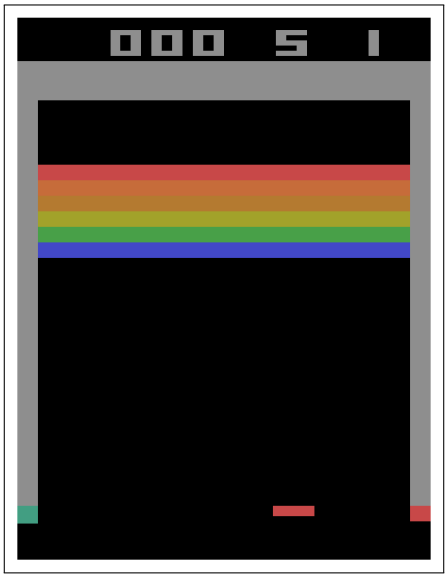

FIGURE 1 – Capture d'écran du jeu Atari Breakout

Par chance, ce jeu est disponible dans l'environnement *Gym* d'*OpenAI*. Cet environnement a été conçu pour développer et comparer des algorithmes d'apprentissage par renforcement (OpenAI, 2020a). Cet environnement est conçu spécifiquement pour le langage Python et possède une centaine d'environnements différents accessibles directement avec *Gym* et plusieurs autres via des tierces parties (OpenAI, 2020b). Cette librairie a été intégrée dans le milieu de l'intelligence artificielle par ses multiples références dans des papiers recherches (V. et al., 2014) et que cette librairie est offert sous la licence *MIT*.

Pour concevoir votre modèle, vous tomber sur le papier "*Human-level control through deep reinforcement learning*" (V. et al., 2015) qui met de l'avant une architecture et un algorithme pour produire un agent qui est capable de rivaliser en termes de performance à des joueurs humains pour les jeux de la plateforme *Atari* telle que le jeu *Breakout*. Selon cette recherche, il vous sera possible de produire un agent tel que demandé par votre client.

2 Préparation du notebook

2.1 Configurer le notebook

Ce laboratoire nécessite beaucoup de calcul pour l'entraînement du modèle qui nécessite un réseau neuronal convolutif. Pour réduire le temps de traitement, il est intéressant d'utiliser le processeur graphique (GPU) offert avec *Colaboratory*. Pour lier un GPU avec un notebook, il faut premièrement se diriger vers l'option "Paramètres de notebook" (*Notebook settings*) sous l'onglet "Édition" (*Edit*) figure 2.

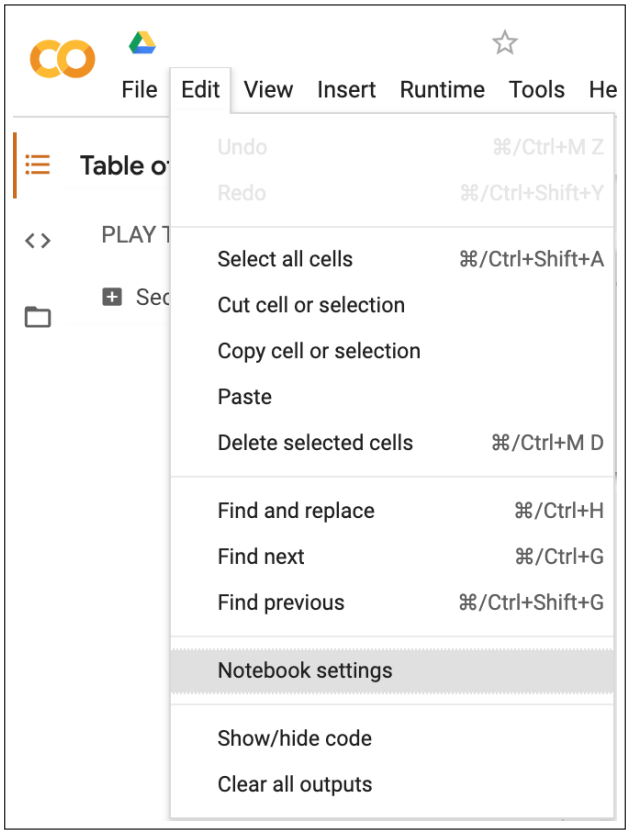

FIGURE 2 – Paramètres du notebook

Par la suite, sous le menu déroulant, sélectionner GPU tel que dans la figure 3. Après avoir sauvegardé les paramètres, connectez le notebook avec une instance de *Colaboratory*.

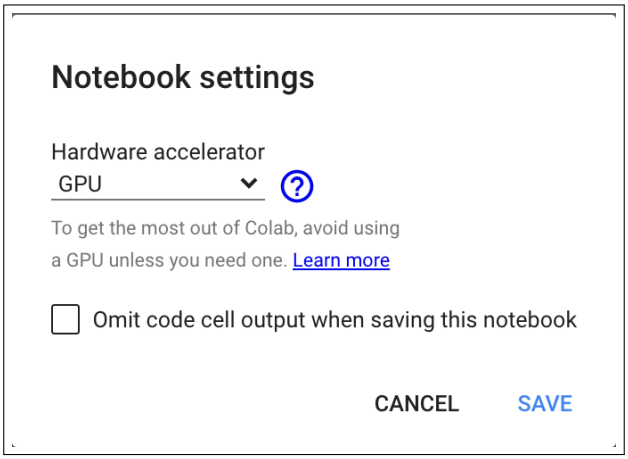

FIGURE 3 – Ajout d'un processeur graphique au notebook

Pour s'assurer que le processeur graphique est bien disponible, utilisez le segment de code 1 dans une cellule de code et faites-en la compilation.

```
%tensorflow_version 2.x
import tensorflow as tf
nom_gpu = tf.test.gpu_device_name()
if nom_gpu != '/device:GPU:0':
  raise SystemError('Aucun GPU trouve')
print('GPU trouve : {}'.format(nom_gpu))
```

Segment de codes 1 – Valider la présence du GPU

2.1.1 Librairies à importer

Pour faire ce laboratoire, quelques librairies devront être importées dans votre projet. Nous voulons utiliser la version 2 de la librairie Tensorflow à l'aide de la commande du segment de code 2.

```
%tensorflow_version 2.x
```

Segment de codes 2 – Sélectionner la version de Tensorflow

Si la commande du segment de code 2 est omise, un avertissement sera affiché lors de l'importation de la librairie "*Tensorflow*".

Les librairies à importer sont :

- tensorflow (*tf*)
- numpy (*np*)
- skimage.color
- cv2
- gym
- matplotlib.pyplot (*plt*)
- heapq
- json

Tel que dans le laboratoire 1, il est conseillé d'ajouter la commande "**%matplotlib inline**" au notebook pour simplifier l'affichage des graphiques.

Un exemple d'importation de librairies est disponible dans la description du premier laboratoire.

3 Conception

Pour simplifier la conception, l'agent est séparé en 2 modules.

— Le tampon d'expérience
— Deep Q Network (réseau neuronal)

3.1 Constantes du programme (hyperparamêtres)

Dans la conception d'algorithme d'apprentissage, machine, certains paramètres sont nécessaires d'être définis pour l'entraînement du réseau tel que le taux d'apprentissage, l'architecture du modèle et la taille du *minibatch* utilisée pour l'apprentissage.

Pour ce laboratoire, les constantes suivantes doivent être définies :

— À trouver dans la documentation
  — Quantité de départ pour le tampon d'expérience (*replay start size*)
  — Taille du tampon d'expérience (*replay memory size*)
  — Taille du minibatch (*minibatch size*)
  — Historique des états (*agent history length*)
  — Fréquence de mise à jour du réseau cible (*target network update frequency*)
  — Facteur de réduction (*discount factor*)
  — Quantité de répétition des actions (*action repeat*)
  — Valeur initiale de l'epsilon pour $\epsilon - greedy$ (*initial exploration*)
  — Valeur finale de l'epsilon pour $\epsilon - greedy$ (*final exploration*)
  — Index de la dernière image d'exploration $\epsilon - greedy$ (*final exploration frame*)
  — Nom de l'environnement (*sous la forme "NomDuJeu-vX"*)
— Offert par l'auteur
  — Taille de l'image préprocessée **(84, 84)** pixels
  — Nombre d'épisodes à effectuer **(3500)**

Certains de ces hyperparamêtres sont facilement disponibles dans les documentations en lien avec le projet. À l'aide des liens suivants, déterminez la valeur des hyperparamêtres.

— https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
— https://gym.openai.com/

3.2 Intégration de l'environnement

L'environnement gym d'OpenAI est conçu afin que tous les environnements possèdent la même interface de programmation (**API**). Avec le temps, cette interface est devenue l'interface par défaut qu'un environnement doit respecter pour être accueilli par la communauté d'apprentissage par renforcement.

3.2.1 Produire un environnement

À l'aide de la libraire "gym", il faut utiliser la fonction "make" du module avec le nom de l'environnement comme argument (segment de code 3).

```
env = gym.make(NOM_DU_JEU)
```

Segment de codes 3 – Création de l'environnement

3.2.2 Extraire les actions utilisées par l'environnement

Comme tous jeux, les actions entre les différents environnements diffèrent. Une des manières pour extraire les définitions des commandes sur un environnement "gym" est d'utiliser la suite de méthodes du segment de code 4.

```
NOMBRE_ACTION = len(env.unwrapped.get_action_meanings())
```

Segment de codes 4 – Extraire les actions utilisées par l'environnement

3.2.3 Interactions avec l'environnement

Pour obtenir l'image de l'environnement, deux méthodes sont disponibles. La première est d'effectuer une action à l'aide de la méthode **step**. Cette méthode retourne un "*tuple*" de quatre arguments : l'image résultante, la récompense, la valeur du drapeau de fin et un dictionnaire d'information dont le nombre de vies restant. La deuxième méthode est à l'aide de la méthode **render** qui fait un rendu sans le besoin d'action. Comparativement à **step**, **render** retourne seulement la dernière image produite par l'environnement. Il est à noter que la méthode **render** est possible qu'après avoir fait une action dans un nouvel environnement.

```
# Importer l'environnement Breakout-v0
env = gym.make('Nom de l'environnement')
# Repartir l'environnement a l'etat initial
env.reset()

# Liste des definitions des actions
env.unwrapped.get_action_meanings()

# Rendu de l'environnement
env.render(mode='rgb_array')

# Faire une action dans l'environnement
env.step(action, render_mode='rgb_array')
```

Segment de codes 5 – Initilisation Gym pour Atari Breakout v0

3.3 Tampon d'expérience

Le tampon d'expérience est l'un des deux modules devant être conçus pour produire l'algorithme DQN (V. et al., 2014). Pour bien concevoir ce module, nous allons mettre sur pied une classe. Cette classe aura comme but de mettre en mémoire des expériences de l'environnement.

3.3.1 Expérience

Le papier mettant en lumière l'algorithme du *DQN* fait aussi l'utilisation du principe de tampon d'expérience (*replay buffer*). L'apprentissage à l'aide d'expérience consécutive pour un agent *DQN* est inefficace (V. et al., 2014). Le tampon d'expérience fera la sélection d'expérience de manière aléatoire.

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

Une expérience est définie comme la composition de l'image d'origine (s_t), l'action choisie par rapport à cette image (a_t), la récompense résultante (r_t) et l'image produite (s_{t+1}) (V. et al., 2014).

Ces composantes sont offertes par l'environnement "gym" lorsqu'une action est effectuée avec la méthode "step".

3.3.2 Création de la classe "*TamponExperience*"

Python étant un langage orienté objet, les modules seront conçus sous la forme d'un objet. Avec Python, un objet se réfère lui-même à l'aide du mot "*self*". De plus, plusieurs méthodes dites *spéciales* sont réservées pour décrire des interactions souvent utilisées. Par exemple, la méthode "**__init__**" est appelée lors de la création d'un nouvel objet d'une classe pour son initialisation (Python, 2020). Cette méthode *spéciale* est celle-là plus couramment modifiée lors de l'écriture d'une classe.

Similaire à "**def**" pour la création d'une fonction, le terme "**class**" sert à indiquer la conception d'une nouvelle classe. Le segment de code 6 offre un exemple pour le début de la création d'une classe.

Pour ce laboratoire, le champ "*héritage*" sera laissé vide.

```
1 class Nom_de_la_classe(heritage):
2   [...]
```

Segment de codes 6 – Création d'une classe en Python

3.4 Fonction d'initialisation

À l'intérieur d'une classe, il est nécessaire de définir une méthode faisant l'initialisant d'un objet. Cette méthode est appelée au premier appel de la classe. Le segment de code 7 montre un exemple d'initialisation pour une classe fictive **Rectangle**.

```
1  class Rectangle():
2    def __init__(self, largeur, hauteur):
3      """
4      __INIT__
5
6      Initialisation d'un objet Rectangle
7
8      @Arguments :
9        largeur {float} : largeur du rectangle [en unite de distance]
10       hauteur {float} : hauteur du rectangle [en unite de distance]
11
12     @Return :
```

```
13          {Rectangle} : l'objet Rectangle ayant appeler la fonction.
14
15      ** Note : il n'est pas necessaire de mettre 'return self' pour __init__
16      """
17      self.largeur = largeur
18      self.hauteur = hauteur
19      self.perimetre = 2 * (self.largeur + self.hauteur)
20      self.aire = self.largeur * self.hauteur
21      return self
```

Segment de codes 7 – Initilisation d'une classe d'un rectangle

En Python, la mention **self** est un pointeur correspondant à l'objet même. **Self** est similaire au pointeur **this** dans d'autres language comme **c++** et **JavaScript**. En appelant une composante de **self**, il est possible de la définir (segment de code 7) et de l'utiliser (segment de code 8).

```
1 r = Rectangle(5,6)
2 print('Aire du rectangle:',r.aire) # >> Aire du rectangle: 30
```

Segment de codes 8 – Appel de la classe Rectange

3.5 Initialisation du tampon

Pour le tampon d'expérience, nous utilisons quatre tampons circulaires qui contiendront les valeurs d'actions, les récompenses, les images traités et les drapeaux de fins aux instant **t**.

Les tampons circulaires sont de la taille maximale du tampon d'expérience. L'allocation de ces tampons sont fait à l'aide de la fonction "numpy.empty". De cette manière, nous contournons les problématiques d'épuisement de mémoire pouvant être produit par le *collecteur de vidange (Garbage collector)*.

```
1 self.images = np.empty((taille_max_tampon, hauteur_image_traitee, largeur_image_traitee),
      dtype=np.uint8)
```

Segment de codes 9 – Allocation du tampon contenant les images

Le but du tampon est de contenir l'information, alors il est nécessaire d'utiliser le type pouvant correspondant aux données sans utiliser de l'espace mémoire superflu.

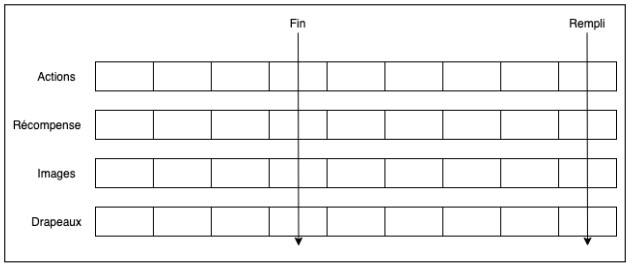

FIGURE 4 – Schématisation d'un tampon circulaire

Faites l'allocation des trois autres tampons

Pour interagir avec les tampons, nous devons utiliser trois variables pour définir l'état de ceux-ci. La première variable contient la taille maximale du tampon (*taille*). La deuxième variable indique combien de valeur est contenu dans le tampon à l'instant $\mathbf{t}$ (*compteur*) et d'une variable utilisée comme pointeur pour donner la position de la prochaine valeur à ajouter.

Afin de réduire l'utilisation de mémoire par le tampon d'expérience, il est intéressant de faire l'allocation d'espace mémoire pour la création des *minibatchs*. Pour cela, nous allons allouer l'espace pour trois matrices. Deux allant contenir des états qui seront traités par les réseaux neuronaux (segment de code 10) et une matrice contenant les indices utilisés pour la conception des états.

```
1 self.etat = np.empty((taille_minibatch, historique, hauteur_image_traitee,
      largeur_image_traitee), dtype=np.uint8)
```

Segment de codes 10 – Allocation de la matrice pour contenir les états

3.5.1 Traitement de l'image

La première modification qui influencera directement l'agent est le traitement de l'image pour réduire la complexité de l'image produite par l'environnement.

Selon l'article original faisant mention de l'architecture "*Deep Q-Network*" (V. et al., 2014), lors de leur conception avec les jeux de la plateforme *Atari 2600*, les images reçues par l'environnement sont réduites à une dimension de 84 pixels par 84 pixels en ton de gris avant d'être envoyées au modèle. Il est important que l'image traitée soit carrée pour être compatible avec les couches de convolution 2D au début du modèle.

Propriétés	Originale	Traitée
Taille	210 x 160 pixels	84 x 84 pixels
Couleur	RGB (128 couleurs)	Ton de gris (0,255)
Contenu	Totalité de l'écran	Section de jeu

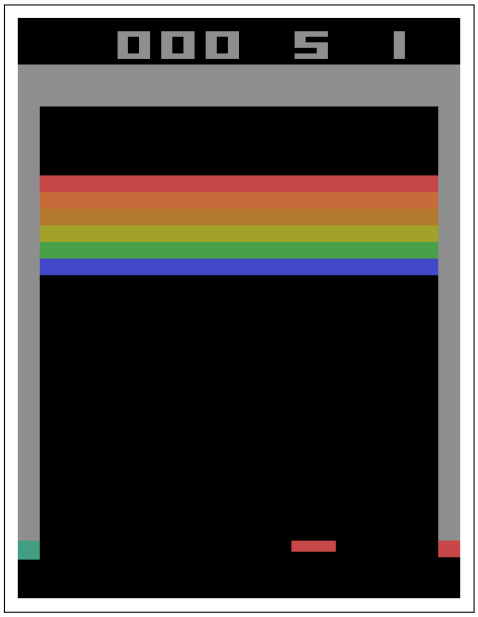

FIGURE 5 – Image tirée de l'environnement *Gym*

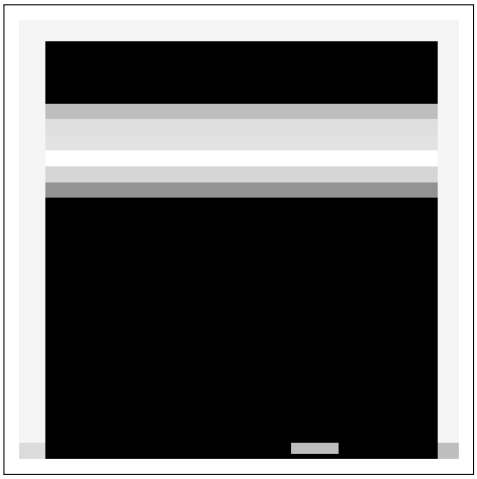

FIGURE 6 – Image traité pour le modèle.

Les fonctions qui sont à votre disposition pour faire le traitement sont :

- tensorflow.image.rgb_to_grayscale
- tensorflow.image.crop_to_bounding_box
- tensorflow.image.resize (avec la méthode **tensorflow.image.ResizeMethod.NEAREST**)
- la méthode de reshape de la matrice *numpy*

Pour ajouter une méthode dans une classe, il faut simplement ajouter une fonction à l'intérieur de la classe avec l'indentation correspondante .

```
class Rectangle()
  def __init__(self, largeur, hauteur):
    ...

  def calculer_aire(self):
    """
    CALCULER_AIRE

    Calculer l'aire d'un rectangle a l'aide d'un objet Rectangle.

    @Arguments :
      self {Rectangle} : Objet faisant l'appel de la methode

    @Return :
      {float} : Aire du rectangle correspondant
    """
    return self.largeur * self.hauteur
```

Segment de codes 11 – Définition d'une méthode statique

Lorsqu'une méthode offre des fonctionnalités similaires à un objet, mais ne nécessite pas l'utilisation d'attribut d'un objet, il est possible de définir une méthode statique à l'intérieur d'une classe. Ceci offre la possibilité d'appeler la fonction sans définir un objet auparavant. Pour définir une méthode statique, il suffit d'ajouter le décorateur "@staticmethod" avant la méthode.

```
class Rectangle()
  def __init__(self, largeur, hauteur):
    ...

  @staticmethod
  def calculer_aire(largeur, hauteur):
    """
    CALCULER_AIRE

    Calculer l'aire d'un rectangle a l'aide de la largeur et de la hauteur.

    @Arguments :
      largeur {float} : Largeur du rectangle
      hauteur {float} : hauteur du rectangle

    @Return :
      {float} : Aire du rectangle correspondant
    """
    return largeur * hauteur
```

Segment de codes 12 – Définition d'une méthode statique

Un exemple d'image du jeu est offert lors de la réinitialisation de l'environnement "gym" (env.reset())

3.5.2 Ajout d'une expérience dans le tampon

Pour ajouter une expérience à l'intérieur du tampon, il suffit de prendre les composantes d'une expérience (image, récompense, action, drapeau de fin) et de les insérer dans leur tampon respectif où le pointeur de donnée indique. Par la suite, les variables de contrôle du tampon sont mises à jour. Le compteur est incrémenté d'un incrément jusqu'à ce qu'il soit identique à la taille du tampon. Pour sa part, le pointeur est incrémenté d'un incrément et est restreint à l'intérieur de la taille du tampon (utiliser l'opération modulo %).

3.5.3 Déterminer les indices pour la création d'un état

La classe du tampon d'expérience est utilisée pour la manipulation des données. Il est donc nécessaire de définir une méthode pour extraire une suite d'index pour concevoir des états pour l'apprentissage du modèle. Pour ce faire, nous devons définir une boucle ayant comme but de produire une quantité précise d'indices aléatoires respectant des conditions précises.

Les conditions à respecter sont :

- Si l'index est plus grand que le pointeur, alors l'indice doit avoir au moins une différence minimum de la taille de l'historique avec un incrément de 1.
- Si le compteur n'est pas équivalent à la taille maximale du tampon, l'index doit être supérieur à l'historique avec un incrément de 1

— Il ne doit pas avoir de drapeaux de fin entre l'index choisie et les indexes contenus à l'intérieur de l'historique

La dernière condition s'assure que les valeurs qui seront à l'intérieur de l'état sont du même épisode. Lorsqu'un indice respecte toutes les conditions, ajouter cette valeur à l'intérieur de la matrice devant contenir les indices des états.

Le but de la tâche est de respecter la distribution uniforme d'une sélection aléatoire. Alors, il n'est pas nécessaire de s'assurer si un indice a déjà été sélectionné.

3.5.4 Création d'un état

Dans le cas de ce laboratoire, un état correspond à l'agrégation d'observation (image). Pour produire une sous-matrice, il est possible d'utiliser une indexation bornée. L'indexation bornée est faite en sorte que la borne inférieure est incluse et celle supérieure est exclue.

```
a = [[1,2,3],[4,5,6],[7,8,9]]
a[0:2] # >> [[1,2,3],[4,5,6]]
```

Segment de codes 13 – Indexation bornée

À l'aide de cette information, créer une méthode pouvant extraire les images d'un index précis et son historique.

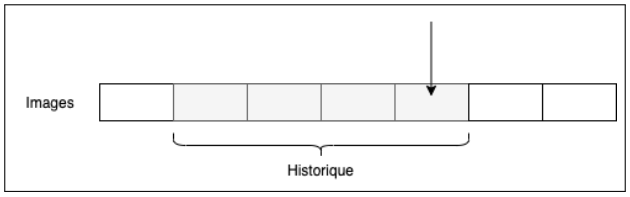

FIGURE 7 – Extraction d'un état

3.5.5 Création d'un minibatch

Le minibatch est le regroupement de plusieurs entrées afin que le modèle fasse leur traitement simultanément. Cette technique aide à réduire la quantité de bruit dans les gradients pouvant être produit par une donnée disparate.

```
def creer_minibatch(self):
  trouver_indices_valide(nombre_donnees_par_minibatch)

  # Une boucle enumerate sert a indique la position de la valeur sortie simultanement
  for i, index in enumerate(self.indices) :
    etats[i] = creer_etat(index-1)
    etats_suivant[i] = creer_etat(index)

```

```
9   return etats, actions[self.indices], recompenses[self.indices], etats_suivant, drapeaux
      [self.indices]
```

Segment de codes 14 – Création d'un minibatch

L'état offert par le segment de code 14 a toutefois la problématique qu'une couche peut avoir de la difficulté à traiter l'information temporelle de l'état, car il n'est pas situé à la dernière dimension de la matrice. Pour corriger cela, il suffit d'utiliser la fonction "numpy.transpose" avec l'argument "axis=(0,2,3,1)". De cette manière, nous prenons la deuxième dimension de la matrice et la transporte à la dernière dimension.

3.5.6 Produire un nouvel état à partir de l'état précédent

Cette dernière fonction du tampon d'expérience a comme but simple de prendre un état et d'y intégrer une nouvelle image traitée en retirant l'image la plus ancienne de l'état.

3.6 Modèle

Avec la gestion d'expérience compléter, il est maintenant nécessaire de construire le réseau de neurones. Le modèle utilisé dans l'algorithme *DQN* est divisible en quatre parties.

La première section du modèle est l'entrée du modèle. Cette couche définit l'entrée afin que *tensorflow* puisse définir l'espace nécessaire pour la conception du réseau neuronal lors de la compilation (*tensorflow.keras.Input*).

La deuxième section est de formater l'entrée. Dans le tampon d'expérience, nous gardons en mémoire l'intensité des pixels sous le format "uint8". Cette représentation a peu de valeur pour le réseau. Pour remédier à la situation, nous allons formater les pixels afin que les valeurs soit entre 0 et 1. Pour ce faire, nous utilisons la couche **tensorflow.keras.Lambda** pour diviser par 255.

La couche **Lambda** traite les données en entrées par une fonction personnalisée. Afin de faciliter l'écriture de cette couche, nous utilisons la création de fonctions à l'aide de la définition simplifiée *lambda*. *Lambda* retourne le résultat du traitement à droite de " :" (segment de code 15).
D'avantage d'information est disponible sous la documentation de la couche : https://www.tensorflow.org/api_ docs/python/t

```
1 nom_fonction = lambda entree: traitement(entree)
```

Segment de codes 15 – Utilisation de la définition par lambda

La troisième section est un réseau convolutif pour l'analyse de l'image. Les réseaux convolutifs sont souvent utilisés dans la catégorisation d'image (AlexNet, ResNet, ...).
Alors, cette section a comme but de déterminer par itérations les caractéristiques importantes du jeu par l'entremise de la sortie de l'environnement. Le réseau convolutif est constitué de 3 couches. Dans la librairie *tensorflow*, il existe trois types de couches de convolution. Dans le laboratoire, nous nous intéressons à la couche de convolution à deux dimensions (**Conv2D**).

Index	Profondeur de sortie (*filters*)	Dimension de noyau	Déplacement (*stride*)	Fonction d'activation
1	32	8×8	4	relu
2	64	4×4	2	relu
3	64	3×3	1	relu

TABLE 1 – Description des couches de convolution

Afin de faire le lien entre une entrée bidimensionnelle et une couche totalement connectée, il est nécessaire d'ajouter une couche d'aplatissement (**Flatten**).

La dernière couche du modèle est un perceptron. Cette couche complètement connectée fait le lien entre l'observation des couches de convolution et les commandes de l'environnement. Le perceptron est constitué de 512 neurones terminés par la fonction d'activation *relu* suivie de 4 neurones représentant les actions de l'environnement du jeu *Atari Breakout*.

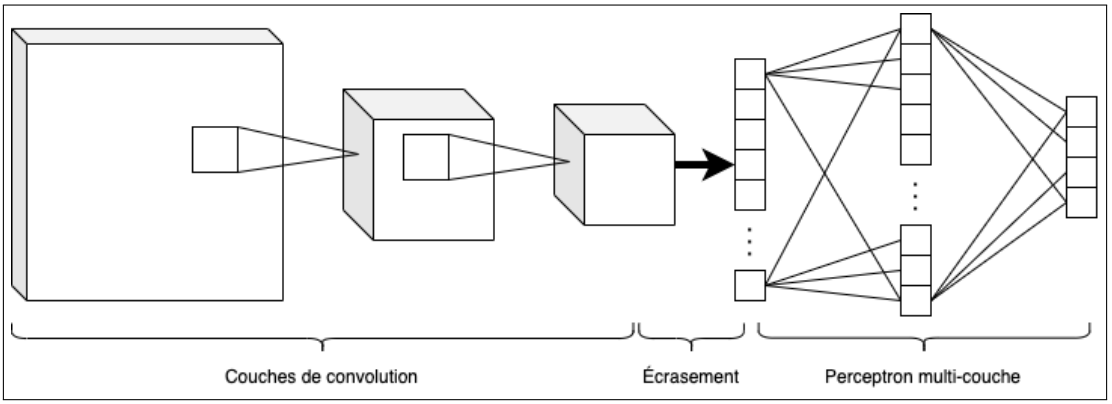

FIGURE 8 – Schéma du modèle

Dans ce laboratoire, le modèle utilise l'optimisateur Adams pour la mise à jour des poids et de la fonction de pertes Huber.

```
1 model.compile(optimizer=tf.keras.optimizers.Adam(1e-5), loss=tf.keras.losses.Huber())
```

Segment de codes 16 – Compilation du modèle

3.6.1 Initialisation du modèle

La conception du modèle devra être faite à l'intérieur d'une fonction afin de pouvoir répliquer le modèle aisément.

La fonction d'initialisation possède deux variables dans sa création. La première variable est la taille de l'image en entrée. La taille de l'image étant déjà définie dans le papier de recherche du DQN. Toutefois, il est intéressant d'offrir la possibilité de modifier ce paramètre dans le futur. La deuxième variable est le nombre d'actions possibles pour l'environnement. De cette manière, il sera possible de réutiliser l'algorithme sur différents jeux de l'environnement *Gym d'OpenAI*.

Avec les techniques utilisées dans le premier laboratoire, faites la création du réseau de neurones.

3.6.2 Entraînement du réseau

La méthode d'entraînement diverge de celui du premier laboratoire en utilisant un aspect davantage manuel au procédé. Dans ce laboratoire, nous utilisons la fonction de perte de l'algorithme **DQN** pour mettre à jour les poids.

$$L(\theta) = E_{s,a,r,s'}[(r + \gamma max_{a'}\hat{Q}_{\theta'}(s', a') - Q_\theta(s, a))^2]$$

Cette équation possède trois segments distinctifs. Le premier segment est $G(r, s', a') = r+\gamma max_{a'}\hat{Q}_{\theta'}(s', a')$ qui correspond à la valeur Q voulue pour le réseau de neurones. Cette équation est la somme entre la valeur Q du réseau cible et de la récompense de l'action obtenue. Le deuxième segment est $Q_\theta(s, a)$ qui correspond à la valeur Q obtenue par le réseau de neurones interagissant avec l'environnement. Finalement, nous calculons la distance euclidienne (L2) entre les deux valeurs. Le préfix $E_{s,a,r,s'}$ indique que le résultat obtenu correspond à la valeur attendue.

Dans le cas du laboratoire, nous utilisons l'outil *GradientTape* de *Tensorflow* pour capter le gradient du réseau de neurones. De cette manière, il sera possible d'analyser précisément la fonction de perte de l'algorithme DQN.

Dans le Segment de code 17, nous utilisons la représentation "*one-hot*" pour cibler la valeur Q obtenus par le réseau de neurones. La représentation "*one-hot*" correspond à une matrice unidimensionnelle dont toutes les valeurs sont forcées à zéro sauf l'indice voulu qui est forcé à un.

Par exemple, la représentation "*one-hot*" de la valeur 3.5 avec une matrice avec une profondeur de 5 correspond à la matrice $\begin{bmatrix}0 & 0 & 0 & 1 & 0\end{bmatrix}$.

Pour faciliter la conception de l'entraînement du modèle, nous utilisons l'outil "GradientTape" offert par Tensorflow qui fait automatique le calcul des gradients pour la mise à jour des poids. De plus, nous remplaçons la fonction de pertes "MSE" par la fonction **Huber** qui offre une forme similaire, mais dont la pente devient constante lorsque l'erreur est trop grande. Cela aide à réduire la possibilité de divergence du modèle.

```
with tf.GradientTape() as tape :
  # Informe quoi gradientTape doit analyser
  tape.watch(self.model.trainable_variables)

  Qs = model.predict(states)
  Q = tf.reduce_sum(Qs * tf.one_hot(actions, self.nbr_action),axis=-1)

  # Calcul des pertes
  pertes = tf.reduce_mean(self.model.loss(G, Q))

gradients = tape.gradient(pertes, self.model.trainable_variables)
self.model.optimizer.apply_gradients(zip(gradients, self.model.trainable_variables))
```

Segment de codes 17 – Utilisation de GradientTape

Avec l'information offerte par le Segment de code 17, nous pouvons en déduire que la fonction d'entrainement nécessite comme arguments l'expérience, la valeur Q du réseau de comparaison et d'un accès au réseau de neurones du modèle.

3.6.3 Transfert des valeurs des poids entre deux modèles

Les méthodes faisant la lecture des poids d'un modèle *Tensorflow* sont **get_weights** et **set_weights**. À partir de ces méthodes, faites la conception de la méthode pouvant transférer les poids du modèle principal vers le modèle comparatif.

3.6.4 Choix d'une action

L'algorithme du DQN (V. et al., 2014) utilise $\epsilon - greedy$ pour l'exploration et l'évaluation du modèle.

- Si une valeur aléatoire entre 0 et 1 est inférieure à la valeur epsilon (ϵ)
  - Retourne une action aléatoire
- Sinon
  - Retourne l'argmax du résultat du modèle pour l'état.

3.7 Jouer un épisode

Les étapes pour jouer un épisode avec l'algorithme DQN (V. et al., 2014).

1. réinitialiser l'environnement
2. Aggréger l'image initiale pour produire un état
3. Tant que le drapeau de fin n'est pas levé
   (a) si un nombre "t" d'actions a été vu, transférer les poids du modèle principal vers le modèle de comparaison
   (b) prendre une action avec $\epsilon - greedy$
   (c) effectuer l'action sur l'environnement
   (d) ajouter les résultats de l'action dans le tampon
   (e) incrémenter le nombre "t" d'actions vues
   (f) réduire la valeur d'epsilon (ϵ)
   (g) faire l'apprentissage du modèle en utilisant un minibatch du tampon
   (h) l'état est mis à jour avec la nouvelle image traitée obtenue à l'aide de l'action précédente.

Il est recommandé de faire la sommation des récompenses offerte par l'environnement pour déterminer les performances du modèle.

Avec cette information, produisez une fonction implémentant le jeu d'un épisode.

Pour faciliter votre conception, il est conseillé que cette fonction reçoit en argument :

— l'environnement
— la valeur "t"
— le tampon d'expérience
— le modèle principal
— le modèle comparatif
— la valeur epsilon (ϵ)

En fin de conserver les modifications avec certains de ces arguments, il est nécessaire de renvoyer les valeurs :
— la valeur "t" actualisée
— la valeur epsilon (ϵ)
— le résultat de l'épisode

3.8 Algorithme DQN

La dernière étape à la conception du laboratoire est de joindre tous les modules pour produire l'algorithme du DQN (V. et al., 2014).

1. Initilisation du tampon d'expérience afin d'attendre la borne inférieure en effectuant que des actions aléatoires.
2. Initilisation du modèle principal et du modèle comparatif.
3. Copier les poids du modèle principal vers le modèle comparatif.
4. Pour un nombre M d'épisodes
   (a) jouer une partie

4 À mettre dans le rapport

Avec le rapport, il faut remettre le notebook ('.ipynb') et le dossier "modeles"

Pour télécharger le fichier du notebook, cliquez l'option "Télécharger .ipynb" (*"Download .ipynb"*) sous l'onglet fichier.

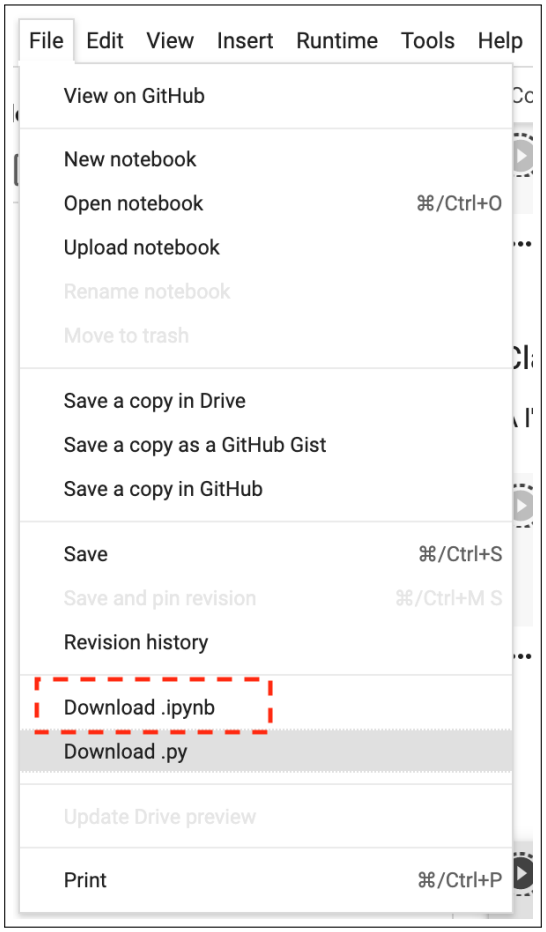

FIGURE 9 – Télécharger le notebook

Afin de télécharger le dossier "modeles", il faut le compresser auparavant. Pour se faire, ouvrer une nouvelle cellule de code et insérer la commande ci-dessous.

```
!zip -r modeles.zip ./modeles
```

Segment de codes 18 – Compresser un fichier sur une session

Faites attention au point d'exclamation [!] au début de la commande

Par la suite, il est possible de télécharger en se dirigeant sur l'icône de fichier dans la barre de gauche du notebook tel que représenté dans la figure 10.

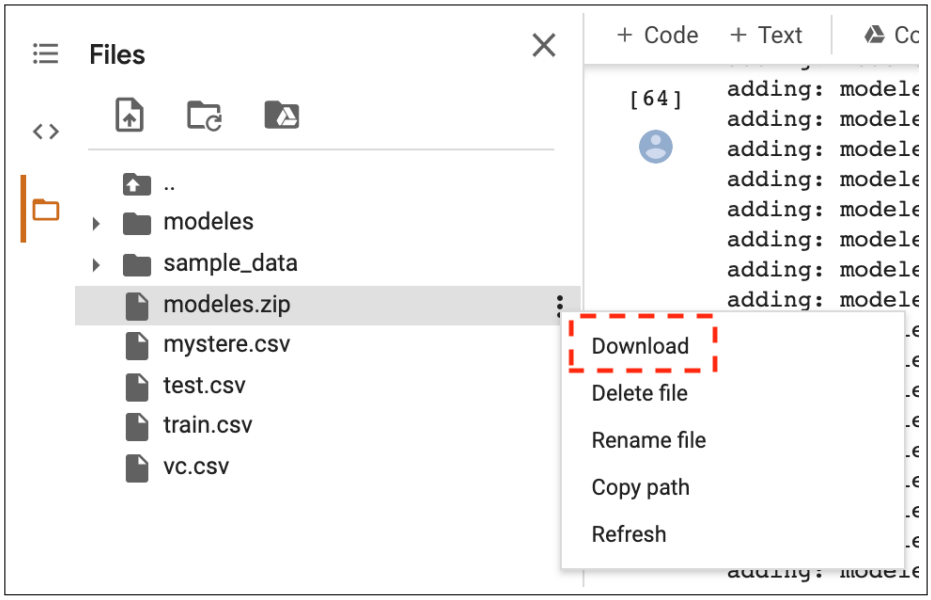

FIGURE 10 – Télécharger le dossier compressé

TABLE 2 – Contenu du rapport

Section	Taille minimum	Pondération
Introduction	3 lignes	5%
Théorie	5 lignes	20%
Résultats	5 lignes	40%
Conclusion	5 lignes	35%

4.1 Description des sections

4.1.1 Introduction

Dans l'introduction, il faut décrire le but du laboratoire et de résumer la procédure du laboratoire.

4.1.2 Théorie

Dans la section théorique, il est nécessaire d'expliquer les théories et les concepts qui sont mis de l'avant par le laboratoire.

4.2 Résultats

La section des résultats doit inclure les graphiques, les tableaux obtenus durant le laboratoire. Il est nécessaire d'inclure une figure décrivant l'entrainement pour chaque modèle. De plus, il est nécessaire d'expliquer le résultat observé à l'intérieur de la figure.

4.2.1 Conclusion

Finalement, la conclusion doit contenir un simple retour sur le but, une évaluation des résultats et une ouverture sur l'amélioration du laboratoire.

Références

G. Krieg, K. Proudfoot, and J. Rosen. Alphago : The movie, 2017. https://www.youtube.com/watch?v=WXuK6gekU1Y.

OpenAI. Dota 2, 8 2017. https://openai.com/blog/dota-2/.

OpenAI. Openai five, 8 2019. https://openai.com/projects/five/.

OpenAI. Gym, 2020a. https://gym.openai.com/.

OpenAI. Gym environments, 2020b. https://gym.openai.com/envs.

Gebhardt P. The history of chess ai, 2 2019. https://blog.paessler.com/the-history-of-chess-ai.

Python. Data model : Special method names, 2020. https://docs.python.org/3/reference/datamodel.html#basic-customization.

Mnih V., Kavukcuoglu K., Silver D., Graves A., Antonoglou I., Wierstra D., and Riedmiller M. Playing atari with deep reinforcement learning. *Journal of Machine Learning Reasearch*, (15) :1929–1958, 6 2014. https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf.

Mnih V., Kavukcuoglu K., Silver D., and Rusu A. A. Human-level control through deep reinforcement learning. *Nature*, (15) :529–542, 2 2015.

J. Von Neumann and Mogenstern O. *Theory of Games and Economic Behavior*. Princeton University Press, The address, 2 edition, 1953. ISBN 9780691041834.