Université Mohamed Premier Oujda
École Nationale de l'Intelligence Artificielle et du Digital Berkane
Année universitaire : 2024 / 2025

Filière : IA
Prof : Mohamed Khalifa BOUTAHIR

MACHINE LEARNING II – TP 1

Objectif :

L'objectif de ce TP est de se familiariser avec les outils essentiels du Reinforcement Learning (RL), notamment OpenAI Gym. Les étudiants vont explorer comment interagir avec un environnement RL et exécuter des actions avant d'implémenter un algorithme d'apprentissage dans les séances suivantes.

A) Partie 1 : Présentation des Bibliothèques Clés

1. OpenAI Gym

- OpenAI Gym est une bibliothèque permettant de simuler des environnements interactifs pour tester des algorithmes de RL.
- Un environnement Gym est défini par un ensemble **d'états, d'actions, de récompenses et d'un critère de fin**.

📌 Installation de Gym

```
pip install --upgrade gymnasium pygame numpy
```

📌 Création d'un environnement

```
import gymnasium as gym

env = gym.make("CartPole-v1", render_mode="human")
env.reset()
```

B) Partie 2 : Exercices Pratiques avec OpenAI Gym

- Exercice 1 : Découverte et Exploration d'un Environnement Gym

Objectif : Comprendre la structure d'un environnement Gym en explorant ses propriétés et ses actions possibles.

✓ Instructions :

1. Afficher l'espace d'actions et l'espace d'observations.
2. Exécuter une boucle de simulation avec des actions aléatoires pendant 100 itérations.

3. Observer les valeurs des observations retournées.

```
print(f"Espace d'actions : {env.action_space}")
print(f"Espace d'observations : {env.observation_space}")

for _ in range(100):
    action = env.action_space.sample()
    observation, reward, done, _, _ = env.step(action)
    print(f"Action : {action}, Observation : {observation}, Reward :
{reward}")
    if done:
        env.reset()

env.close()
```

- Exercice 2 : Manipulation des Observations et Récompenses

Objectif : Comprendre comment récupérer les observations et les récompenses lors de l'interaction avec l'environnement.

✓ Instructions :

1. Prendre une action et récupérer les valeurs retournées (`observation`, `reward`, `done`).
2. Afficher ces valeurs et analyser leur signification.
3. Faire plusieurs essais et noter les variations des observations et des récompenses.

- Exercice 3 : Contrôle Manuel de l'Agent

📌 **Objectif :** Permettre à l'utilisateur de contrôler manuellement l'agent pour mieux comprendre l'effet des actions.

✓ Instructions :

1. Demander à l'utilisateur d'entrer une action (`0` ou `1`).
2. Exécuter l'action dans l'environnement et afficher les nouvelles observations.
3. Répéter l'opération jusqu'à la fin de l'épisode.
4. Afficher la durée totale de l'épisode avant qu'il ne se termine.

- Exercice 4 : Évaluation des Performances d'une Politique Aléatoire

📌 **Objectif :** Mesurer la durée moyenne d'un épisode lorsqu'un agent prend des actions aléatoires.

✓ Instructions :

1. Faire exécuter à l'agent des actions aléatoires pendant plusieurs épisodes (ex : 10 épisodes).
2. Calculer la durée moyenne avant que l'épisode ne se termine.
3. Comparer les résultats entre plusieurs exécutions.

Université Mohamed Premier Oujda
École Nationale de l'Intelligence Artificielle et du Digital Berkane
Année universitaire : 2024 / 2025

Filière : IA
Prof : Mohamed Khalifa BOUTAHIR

MACHINE LEARNING II – TP 2

Objectif :

L'objectif de ce TP est de mettre en pratique les concepts fondamentaux de l'apprentissage par renforcement en explorant l'algorithme Q-Learning. À travers une série d'exercices progressifs, les étudiants vont apprendre à implémenter cet algorithme, comprendre l'impact des stratégies d'exploration et d'exploitation, et analyser la convergence des valeurs Q. L'environnement FrozenLake de OpenAI Gym servira de terrain d'expérimentation, permettant d'illustrer concrètement comment un agent apprend à optimiser ses décisions grâce aux mises à jour successives de sa Q-table.

Exercice 1 : Exploration de l'Environnement FrozenLake

Instructions :

1. Charger l'environnement **FrozenLake-v1** de OpenAI Gym.
2. Afficher les informations de l'espace d'états et d'actions.
3. Exécuter une boucle où l'agent prend des actions aléatoires pendant plusieurs épisodes.
4. Observer les observations et les récompenses obtenues.

```
import gymnasium as gym
import numpy as np

# Charger l'environnement FrozenLake
env = gym.make("FrozenLake-v1", is_slippery=True)
…
```

Exercice 2 : Implémentation de la Q-Table et Initialisation

Instructions :

1. Créer une **Q-Table** de dimension `(nombre d'états x nombre d'actions)`, initialisée à 0.
2. Afficher la Q-Table avant l'apprentissage.
3. Vérifier que chaque état a une liste de valeurs associées aux actions possibles.

```
# Initialisation de la Q-Table
q_table = …
# Affichage de la Q-Table initiale
print("Q-Table initialisée :")
print(q_table)
```

Exercice 3 : Implémentation du Q-Learning avec Mise à Jour

Instructions :

1. Définir les **hyperparamètres** : taux d'apprentissage (alpha), facteur de discount (gamma), epsilon pour l'exploration.
2. Mettre à jour la **Q-Table** en appliquant la règle de mise à jour du Q-Learning :

$$Q(s,a) = Q(s,a) + \alpha\,[R + \gamma \max Q(s',a') - Q(s,a)]$$

3. Exécuter plusieurs épisodes et observer l'évolution de la table.

```
# Paramètres
alpha = 0.1 # Taux d'apprentissage
gamma = 0.99 # Facteur de discount
epsilon = 1.0 # Exploration initiale
epsilon_decay = 0.995 # Décroissance d'epsilon
num_episodes = 5000 # Nombre d'épisodes

# Boucle d'apprentissage
for episode in range(num_episodes):
…
```

Exercice 4 : Évaluation des Performances de l'Agent

Instructions :

1. Lancer 100 épisodes en utilisant uniquement l'action optimale (argmax(Q[s, a])).
2. Mesurer le taux de réussite de l'agent (nombre de fois où il atteint l'objectif).
3. Comparer les performances avec celles obtenues dans l'Exercice 1 (actions aléatoires).

```
num_test_episodes = 100
successes = 0
for _ in range(num_test_episodes):
    …
```

Université Mohamed Premier Oujda
École Nationale de l'Intelligence Artificielle et du Digital Berkane
Année universitaire : 2024 / 2025

Filière : IA
Prof : Mohamed Khalifa BOUTAHIR

MACHINE LEARNING II – TP 3

Objectif de TP - Optimisation des Feux de Circulation avec Apprentissage par Renforcement

L'objectif de ce TP est d'explorer l'optimisation des feux de circulation à l'aide de l'apprentissage par renforcement. Les étudiants vont :

- Découvrir un environnement simulé de gestion du trafic.
- Implémenter Q-Learning et SARSA pour apprendre une politique optimale.
- Comparer les résultats des deux algorithmes à l'aide de graphiques et d'évaluations quantitatives.

Exercice 1 : Découverte de l'Environnement

Instructions :

1. Téléchargez le fichier de l'environnement (*env-traffic.py*) depuis Google Classroom.
2. Installez les dépendances nécessaires si ce n'est pas encore fait.
3. Importez et exécutez l'environnement pour tester son fonctionnement.

```
from env_traffic import TrafficEnvironment

env = TrafficEnvironment()
state = env.reset()

for _ in range(10):
    action = 0  # Garder le feu tel qu'il est
    next_state, reward = env.step(action)
    print(f"État : {next_state}, Récompense : {reward}")
```

Exercice 2 : Implémentation de Q-Learning

Instructions :

1. Initialisez une Q-Table pour stocker les valeurs des actions pour chaque état.
2. Implémentez l'algorithme Q-Learning, en mettant à jour la Q-Table à chaque itération.
3. Utilisez une stratégie ε-greedy pour gérer l'exploration/exploitation.
4. Exécutez l'apprentissage sur plusieurs épisodes et Affichez la Q-Table finale après l'entraînement.

Le Q-Learning suit la mise à jour suivante : $Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$

```
# Initialisation de la Q-Table
q_table = np.zeros((10, 10, 10, 10, 2))

def train_q_learning(env, episodes=1000, alpha=0.1, gamma=0.9, epsilon=1.0,
decay=0.995):
……
```

Exercice 3 : Implémentation de SARSA

Instructions :

1. Créez une nouvelle Q-Table pour SARSA.
2. Mettez en œuvre l'algorithme SARSA, qui met à jour la Q-Table avec la valeur de l'action effectivement choisie.
3. Utilisez une stratégie ε-greedy.
4. Affichez la Q-Table finale et comparez avec celle de Q-Learning.

SARSA suit la mise à jour suivante : $Q(s,a) \leftarrow Q(s,a) + \alpha\left[R + \gamma Q(s',a') - Q(s,a)\right]$

```
def train_sarsa(env, episodes=1000, alpha=0.1, gamma=0.9, epsilon=1.0,
decay=0.995):
    for episode in range(episodes):
…
```

Exercice 4 : Analyse et Visualisation des Résultats

Instructions :

1. Générez un graphique montrant l'évolution des récompenses au fil des épisodes.
2. Comparez la rapidité d'apprentissage entre les deux algorithmes.
3. Affichez les meilleures politiques apprises.
4. Ajoutez les scores enregistrés pendant l'apprentissage.
5. Interprétez le graphique : Quel algorithme apprend plus vite ?

```
plt.plot(q_learning_rewards, label="Q-Learning")
plt.plot(sarsa_rewards, label="SARSA")
plt.xlabel("Épisodes")
plt.ylabel("Récompense Cumulative")
plt.legend()
plt.show()
      …
```

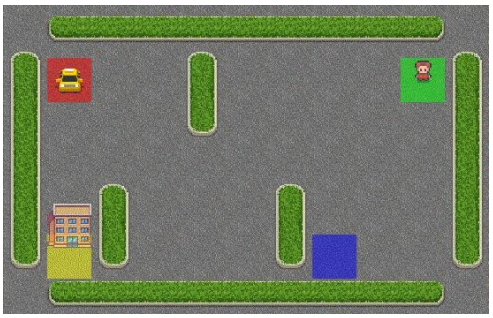
Université Mohamed Premier Oujda
École Nationale de l'Intelligence Artificielle et du Digital Berkane
Année universitaire : 2024 / 2025

Filière : IA
Prof : Mohamed Khalifa BOUTAHIR

MACHINE LEARNING II – TP 4

Objectif du TP :

L'objectif de ce TP est de familiariser les étudiants avec l'implémentation de l'algorithme **Proximal Policy Optimization (PPO)**. À travers ce TP, les étudiants apprendront à construire une **table de politiques**, à **mettre à jour la valeur des états** et à **entraîner un agent** à résoudre le problème de transport de passagers dans l'environnement **Taxi-v3**.



Exercice 1 : Initialisation de l'environnement et des structures de données

- Initialiser l'environnement **Taxi-v3** et afficher **le nombre d'états et d'actions**.
- Créer une **table de politique** où chaque état a une probabilité égale pour chaque action.
- Créer une **table de valeurs** initialisée à zéro.
- Ajouter un affichage des premières lignes de policy_table et value_table.

```
import gymnasium as gym
import numpy as np

# Initialisation de l'environnement
env = gym.make("Taxi-v3")

# Nombre d'états et d'actions
state_size = env.observation_space.n
action_size = env.action_space.n

.....
```

Exercice 2 : Exploration et collecte d'épisodes

- Faire exécuter un **agent aléatoire** dans l'environnement **pendant 20 épisodes**.
- Afficher les **actions exécutées et les récompenses obtenues**.

```
state, _ = env.reset()
for t in range(20):
    .....
```

Exercice 3 : Mise à jour de la politique avec PPO

L'algorithme PPO optimise la politique $\pi\theta$ en maximisant la fonction suivante avec un terme de clipping pour éviter des mises à jour trop brutales :

$$L(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right]$$

Étapes clés de la mise à jour PPO

1. **Calcul des récompenses cumulées** Rt (discounted rewards).
2. **Calcul de l'avantage** At=Rt - V(st).
3. **Mise à jour de la politique avec clipping** : Ajuster $\pi\theta$ en respectant les contraintes de PPO.
4. **Mise à jour de la fonction de valeur** V(s).

- Calculer **les récompenses cumulées** (discounted rewards).
- Mettre à jour la **fonction de valeur** pour chaque état visité.
- Mettre à jour **la politique avec PPO** (en respectant le clip).
- Ajouter une mise à jour de value_table[state] avec une learning rate.

```
gamma = 0.99
lr_policy = 0.1
clip_epsilon = 0.2

episode_states = [state1, state2, ...]  # Liste des états
episode_actions = [a1, a2, ...]  # Liste des actions
episode_rewards = [r1, r2, ...]  # Liste des récompenses
......
```

Exercice 4 : Évaluation de l'agent après entraînement

- Tester l'agent entraîné **pendant 20 épisodes**.
- Comparer **les performances** avant et après entraînement.

```
num_eval_episodes = 20
total_rewards = []

for ep in range(num_eval_episodes):
    state, _ = env.reset()
    total_reward = 0

    ..........
```

Université Mohamed Premier Oujda
École Nationale de l'Intelligence Artificielle et du Digital Berkane
Année universitaire : 2024 / 2025

Filière : IA
Prof : Mohamed Khalifa BOUTAHIR

Machine Learning II – TP 5

Objectif du TP :

L'objectif de ce TP est de découvrir l'utilisation pratique de la bibliothèque TensorFlow Agents (TF-Agents) pour entraîner un agent d'apprentissage par renforcement dans un environnement simple.

À travers ce TP, vous allez manipuler les composants fondamentaux d'un agent RL : environnement, réseau, agent, buffer, politique et entraînement.

Exercice 1 : Préparer l'environnement et les outils

Consignes :

- Installez les bibliothèques nécessaires (tf-agents, tensorflow, etc.).
- Créez l'environnement CartPole-v0 à l'aide de suite_gym de TFAgent.
- Affichez les spécifications de l'environnement (observation et action specs).
- Testez l'environnement avec un acteur aléatoire et observez les épisodes.

Exercice 2 : Création du réseau et de l'agent

Consignes :

- Créez un réseau de neurones pour approximer les Q-values (QNetwork).
- Définissez un agent DQN à l'aide de DqnAgent, en précisant :
  - Optimiseur
  - Stratégie de politique
  - Fonction de perte
- Initialisez les variables du modèle.

Exercice 3 : Entraînement et évaluation

Consignes :

- Créez un replay buffer pour stocker les expériences.
- Implémentez une boucle de collecte de données (policy.collect_policy).
- Implémentez la boucle d'entraînement et affichez régulièrement la reward moyenne.
- Évaluez les performances de l'agent après l'entraînement.

Université Mohamed Premier Oujda
École Nationale de l'Intelligence Artificielle et du Digital Berkane
Année universitaire : 2024 / 2025

Filière : IA
Prof : Mohamed Khalifa BOUTAHIR

Machine Learning II – TP 6

Objectif du TP

Dans ce TP, nous allons entraîner un agent intelligent à jouer au célèbre jeu **Super Mario Bros** en utilisant la bibliothèque **TorchRL**. L'objectif est de découvrir comment appliquer un algorithme d'apprentissage par renforcement profond dans un environnement visuel complexe. À la fin, chaque groupe devra générer une **vidéo du gameplay de leur agent entraîné**.

Exercice 1 : Préparation de l'environnement

Objectif : Installer les librairies nécessaires et préparer l'environnement Mario.

Instructions :

- Installer les bibliothèques nécessaires : `gym-super-mario-bros`, `torch`, `torchrl`, `numpy`, `opencv-python`, etc.
- Vérifier que l'environnement **SuperMarioBros-v0** fonctionne correctement avec `gym.make`.
- Rendre l'environnement compatible avec TorchRL.

```
pip install gym-super-mario-bros==7.4.0 nes-py opencv-python torch
torchvision torchrl
```

Testez le rendu avec un `env.render()` et assurez-vous que Mario apparaît à l'écran.

Exercice 2 : Prétraitement et transformation de l'environnement

Objectif : Appliquer les wrappers nécessaires pour que l'agent reçoive des observations optimisées.

Instructions :

- Convertir les images en niveaux de gris et les redimensionner (84x84).
- Appliquer le `FrameStack` pour conserver une mémoire courte (ex : 4 images).
- S'assurer que les observations sont converties en **tensors Torch**.

Cela permet à l'agent d'avoir une meilleure compréhension de l'environnement visuel, tout en réduisant la charge de calcul.

Exercice 3 : Implémentation et entraînement de l'agent

Objectif : Créer un agent basé sur un **Deep Q-Network (DQN)** à l'aide de TorchRL et commencer l'entraînement.

Instructions :

- Définir un modèle CNN simple pour traiter les frames du jeu.
- Utiliser TorchRL pour créer un policy learner avec l'algorithme DQN.
- Entraîner l'agent sur un nombre limité d'épisodes pour observer l'amélioration.

Logguez les récompenses moyennes par épisode pour suivre la progression. Vous pouvez sauvegarder le modèle après quelques centaines d'épisodes.

Exercice 4 : Évaluation et génération d'une vidéo

Objectif : Faire rejouer l'agent entraîné et enregistrer une vidéo de sa performance.

Instructions :

- Charger le modèle entraîné et faire jouer Mario pendant un épisode complet.
- Capturer les frames à l'aide de `cv2.VideoWriter` ou `imageio`.
- Générer une vidéo `.mp4` ou `.gif` et la sauvegarder dans le répertoire du TP.

```
import imageio
with imageio.get_writer('mario_agent.mp4', fps=30) as video:
    for frame in frames:
        video.append_data(frame)
```