Apprentissage par Différence Temporelle (TD) en Apprentissage par Renforcement

1. Introduction à l'Apprentissage par Différence Temporelle (TD)

1.1 Définition

L'apprentissage par différence temporelle (TD) est une technique qui combine les concepts des méthodes de Monte Carlo et de la programmation dynamique. TD permet d'estimer la valeur d'un état en utilisant les **récompenses observées** et les **estimations des valeurs futures**, sans attendre la fin des épisodes.

1.2 Rappel de l'Équation de Bellman

Avant de plonger dans TD, rappelons l'**équation de Bellman**, qui exprime la valeur d'un état s comme suit :

$$V(s) = \mathbb{E}[R_{t+1} + \gamma V(S_{t+1})]$$

Cette équation indique que la valeur d'un état s dépend de la récompense immédiate R_{t+1} et de la valeur estimée de l'état suivant S_{t+1}.

L'idée de TD est d'utiliser cette relation pour mettre à jour la valeur $V(s)$ de manière progressive, étape par étape.

—

2. Prédiction TD(0) : Estimation de Valeur

2.1 Objectif de TD(0)

TD(0) est une méthode de prédiction qui estime la valeur d'un état $V(s)$ en utilisant à la fois la récompense immédiate et la valeur de l'état suivant.

2.2 Équation de Mise à Jour TD(0)

L'équation de mise à jour TD(0) pour un état s est :

$$V(s) \leftarrow (1 - \alpha)V(s) + \alpha[R_{t+1} + \gamma V(S_{t+1})]$$

où :

- $(1 - \alpha)$ est le facteur d'**exploitation**, qui utilise la valeur actuelle,
- α est le **taux d'apprentissage**, qui ajuste le poids de la nouvelle estimation.

Note : Plus α est élevé, plus l'agent "écoute" les nouvelles expériences (exploration). Plus il est faible, plus il conserve les valeurs actuelles (exploitation).

—

3. Contrôle TD : SARSA et Q-Learning

Les méthodes de contrôle TD permettent à l'agent de choisir ses actions pour optimiser les récompenses.

3.1 SARSA (Sur-Politique)

SARSA est une méthode de contrôle **sur-politique** : elle met à jour $Q(s, a)$ en suivant l'action réellement prise par la politique actuelle.

Équation de Mise à Jour SARSA :

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})]$$

3.2 Q-Learning (Hors-Politique)

Q-Learning est une méthode de contrôle **hors-politique** : elle met à jour $Q(s, a)$ en utilisant la meilleure action possible dans l'état suivant, indépendamment de l'action choisie.

Équation de Mise à Jour Q-Learning :

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a')]$$

Différence entre SARSA et Q-Learning :

- **SARSA** : suit la politique actuelle et apprend en fonction des actions prises.
- **Q-Learning** : utilise l'action optimale pour chaque état, ce qui le rend plus agressif dans l'apprentissage.

—

4. Exploration et Exploitation avec la Politique Epsilon-Greedy

4.1 Pourquoi Epsilon-Greedy ?

Un défi en apprentissage par renforcement est de trouver un équilibre entre :

- **Exploration** : essayer de nouvelles actions pour découvrir de meilleures récompenses,
- **Exploitation** : utiliser les actions déjà connues pour maximiser les récompenses.

4.2 Politique Epsilon-Greedy

Avec une politique **epsilon-greedy**, l'agent :

- choisit une action aléatoire avec une probabilité ϵ (exploration),
- choisit l'action avec la meilleure valeur estimée avec une probabilité $1 - \epsilon$ (exploitation).

—

5. Implémentation en Python

5.1 TD(0) en Python

Voici une implémentation de l'algorithme TD(0) en Python :

```
import numpy as np

# D finir les  tats  et la politique
states = [0, 1, 2, 3, 4]
policy = {s: np.random.choice(['a', 'b']) for s in states}

# R gle de mise    jour TD(0)
def td_prediction(policy, episodes=100, alpha=0.1, gamma=0.9):
    V = np.zeros(len(states))
```

```
    for _ in range(episodes):
        state = np.random.choice(states)
        while state != 4:  #  tat  terminal
            next_state = np.random.choice(states)
            reward = np.random.randn()  # r compense al atoire
            V[state] = (1 - alpha) * V[state] + alpha * (reward + gamma * V[
                ↪ next_state])
            state = next_state
    return V

# Ex cution de TD(0)
value_function = td_prediction(policy)
print("Fonction de Valeur Estim e :", value_function)
```

5.2 Q-Learning en Python

Voici l'implémentation de Q-Learning avec une politique epsilon-greedy en Python.

```
from collections import defaultdict

# Q-Learning avec epsilon-greedy
def q_learning(episodes, alpha=0.1, gamma=0.9, epsilon=0.1):
    Q = defaultdict(lambda: np.zeros(len(actions)))
    for _ in range(episodes):
        state = np.random.choice(states)
        while state != 4:  #  tat  terminal
            action = epsilon_greedy_policy(Q, state, epsilon)
            next_state = np.random.choice(states)
            reward = np.random.randn()  # r compense al atoire
            best_next_action = np.argmax(Q[next_state])
            Q[state][actions.index(action)] = (1 - alpha) * Q[state][actions.index
                ↪ (action)] + \
                                              alpha * (reward + gamma * Q[
                                                  ↪ next_state][best_next_action])
            state = next_state
    return Q

# Ex cution de Q-Learning
Q_q_learning = q_learning(episodes=1000)
print("Valeurs Q Estim es (Q-Learning) :", Q_q_learning)
```

—

Conclusion

Les méthodes d'apprentissage par différence temporelle permettent d'estimer les valeurs d'état ou d'état-action sans avoir besoin d'un modèle complet de l'environnement. SARSA et Q-Learning sont deux algorithmes populaires pour le contrôle TD, chacun ayant des caractéristiques uniques :

- **SARSA** : apprend en suivant la politique actuelle.
- **Q-Learning** : apprend en utilisant l'action optimale, ce qui en fait une méthode plus "directe".

Ces algorithmes permettent à l'agent d'apprendre en explorant son environnement, tout en équilibrant exploration et exploitation grâce à une politique epsilon-greedy.