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 méthode 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

L'**équation de Bellman** 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}.

TD utilise cette relation pour mettre à jour la valeur $V(s)$ de manière progressive.

—

2. Prédiction TD(n) : Généralisation

2.1 Évolution de TD(0) à TD(n)

La méthode TD peut être généralisée avec le paramètre n, qui représente le nombre d'étapes futures utilisées pour calculer la mise à jour de la valeur d'un état.

- **TD(0)** : Utilise uniquement la récompense immédiate et la valeur de l'état suivant.
- **TD(1)** : Prend en compte la récompense sur deux étapes (la récompense immédiate et la suivante).
- **TD(2)** : Considère trois étapes, c'est-à-dire deux transitions dans le futur.
- **TD(n)** : Généralise le nombre d'étapes, où n représente la profondeur des futures transitions prises en compte.

2.2 Formule Générale de TD(n)

La mise à jour de TD(n) pour un état s est donnée par :

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha \left(\sum_{k=0}^{n-1} \gamma^k R_{t+k+1} + \gamma^n V(S_{t+n}) \right)$$

où :

- R_{t+k+1} est la récompense reçue à chaque étape jusqu'à n,
- γ est le facteur de discount (ou d'actualisation),
- $V(S_{t+n})$ est la valeur de l'état atteint après n étapes.

2.3 Convergence de TD(n)

Lorsque $n \to \infty$, TD(n) devient équivalent aux **méthodes de Monte Carlo**. En effet, plus n est grand, plus TD(n) prend en compte un nombre important d'étapes futures, jusqu'à inclure un épisode complet. Cela signifie que TD(n) converge vers la vraie valeur d'un état, calculée comme la moyenne des retours sur tous les épisodes.

Conclusion : TD(0) utilise une seule étape, tandis que TD(n) avec $n \to \infty$ prend en compte un épisode complet, rejoignant ainsi les méthodes de Monte Carlo.

—

3. Exemple de TD(1) et TD(2)

3.1 TD(1) : Utilisation de Deux Étapes

L'équation de TD(1) est la suivante :

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha[R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2})]$$

Dans TD(1), on utilise la récompense immédiate R_{t+1}, la prochaine récompense R_{t+2}, et la valeur de l'état S_{t+2}.

3.2 TD(2) : Utilisation de Trois Étapes

L'équation de TD(2) est :

$$V(s) \leftarrow (1-\alpha)V(s) + \alpha[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V(S_{t+3})]$$

TD(2) utilise trois récompenses (jusqu'à R_{t+3}) et se projette jusqu'à S_{t+3}.

—

4. Contrôle TD : SARSA et Q-Learning

4.1 SARSA (Sur-Politique)

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

Équation de Mise à Jour SARSA :

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

4.2 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 entreprise.

Équation de Mise à Jour Q-Learning :

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

4.3 Différence entre SARSA et Q-Learning

- **SARSA** : suit la politique actuelle.
- **Q-Learning** : utilise l'action optimale pour chaque état.

—

5. Politique Epsilon-Greedy et Exploration/Exploitation

5.1 Pourquoi Epsilon-Greedy ?

La politique epsilon-greedy permet d'équilibrer :

- **Exploration** : essayer des actions aléatoires avec une probabilité ϵ,
- **Exploitation** : choisir l'action avec la meilleure valeur estimée avec une probabilité $1 - \epsilon$.

—

6. Implémentation en Python de TD(0), TD(1) et TD(2)

6.1 TD(1) et TD(2) en Python

Voici un code pour TD(1) et TD(2) qui montre comment calculer des valeurs en utilisant plusieurs étapes de prévision.

```
import numpy as np

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

# R gle de mise    jour TD(n)
def td_n(policy, episodes=100, alpha=0.1, gamma=0.9, n=1):
    V = np.zeros(len(states))
    for _ in range(episodes):
        state = np.random.choice(states)
        while state != 4:  #  tat  terminal
            rewards = []
            next_state = state
            for i in range(n):
                next_state = np.random.choice(states)
                reward = np.random.randn()  # r compense al atoire
                rewards.append(reward)
            total_return = sum(gamma**k * rewards[k] for k in range(n)) + gamma**n
                ↪  * V[next_state]
            V[state] = (1 - alpha) * V[state] + alpha * total_return
            state = next_state
    return V

# Ex cution de TD(1) et TD(2)
V_td1 = td_n(policy, episodes=100, alpha=0.1, gamma=0.9, n=1)
V_td2 = td_n(policy, episodes=100, alpha=0.1, gamma=0.9, n=2)
print("Valeurs␣TD(1)␣:", V_td1)
print("Valeurs␣TD(2)␣:", V_td2)
```

—

Conclusion

La méthode TD(n) permet de moduler le nombre d'étapes prises en compte lors de la mise à jour des valeurs d'état. Lorsque $n \to \infty$, TD(n) devient équivalent à une méthode de Monte Carlo, car elle considère un épisode complet.

Ainsi :

- **TD(0)** : utilisation d'une seule étape.
- **TD(n)** : utilisation de plusieurs étapes, avec une convergence vers Monte Carlo pour des n élevés.

TD offre une flexibilité pour les problèmes d'apprentissage par renforcement, permettant un équilibre entre la précision et la rapidité d'apprentissage.