Tabular Q-Learning (TD0)

Overview:
This code implements the Tabular Q-Learning algorithm, a popular off-policy Temporal Difference (TD) method for solving Reinforcement Learning problems. It is designed for environments *with* **discrete state and action spaces.** Q-Learning relies on updating the Q-value estimates using a greedy policy derived from the learned Q-values.

Why Tabular methods?

Tabular methods like Q-learning, SARSA, and Expected SARSA were developed to address the **limitations of dynamic programming (DP)** methods in reinforcement learning (RL) problems. While DP methods such as policy iteration and value iteration are foundational, they come with certain drawbacks that tabular methods aim to overcome.

Here are the key reasons tabular methods were developed despite the existence of DP:

1. No Requirement for a Model of the Environment

- **Dynamic Programming:** Requires a complete and accurate model of the environment, specifically the **state transition probabilities** and the **reward function**. This is often impractical for real-world problems where the model is unknown or too complex to specify.
- **Tabular Methods:** Are **model-free**, meaning they do not need explicit knowledge of the environment's dynamics. Instead, they learn directly from interactions (e.g., state, action, reward, and next state). This makes tabular methods more applicable to real-world scenarios.

2. Sample-Based Updates

- **Dynamic Programming:** Requires a **synchronous update** of the value function for all states, which assumes access to the full environment model and can be computationally expensive. In other words, in each iteration the algorithm updates state-value or action-value for all states (or state/action Paris). In addition to convergence time that can be longer for DP algorithm, in many cases we may have not seen some states when we are going to update or in practical situation some states might be unknown yet.

- **Tabular Methods:** Use **sample-based updates**, relying on individual interactions with the environment. For instance:
    - Q-Learning updates the value for one state-action pair at a time based on sampled rewards and transitions.
    - This makes them much more efficient in environments where full environment information is unavailable or infeasible to compute.
    -

3. Suitability for Online Learning

- **Dynamic Programming:** Is fundamentally **batch-oriented**, requiring multiple sweeps over the entire state space to converge to an optimal policy. This is not practical for online or real-time applications.
- **Tabular Methods:** Are inherently **online algorithms** that update the value function incrementally during each interaction with the environment. This allows them to operate effectively in real-time scenarios.

Q Learning Algorithm:

1. Initialize the Q-table for all state-action pairs with arbitrary values (e.g., zeros).
2. Repeat for a number of episodes:
    - Initialize the starting state.
    - For each step of the episode:
        - Choose an action using the epsilon-greedy policy.
        - Take the action, observe the reward, and transition to the next state.
        - Update the Q-value using the formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

        - Update the current state to the next state.
3. Continue until the agent converges to the optimal policy.

Parameters Explained:

- **α (Learning Rate):** Determines the step size for updating Q-values.
- **γ (Discount Factor):** Reflects the importance of future rewards.
- **ε (Epsilon for Exploration):** Governs the likelihood of random action selection for exploration.

Output:

- A trained Q-table containing the optimal Q-values for each state-action pair.
- Optionally, visualization of the learned policy or performance metrics.

Applications:
Q-Learning is widely used in discrete RL problems like grid-world environments, simple games, or scenarios where the agent learns to navigate or perform tasks efficiently.

Tabular Expected SARSA

Overview:
This code implements the Tabular Expected SARSA algorithm, an on-policy Temporal Difference (TD) method for solving RL problems. Expected SARSA updates the Q-values based on the expectation of the next state-action values, considering the agent's policy.

Expected Sarsa Algorithm:

1. Initialize the Q-table for all state-action pairs with arbitrary values.
2. Repeat for a number of episodes:
    - Initialize the starting state.
    - For each step of the episode:
        - Choose an action using the policy (e.g., epsilon-greedy).
        - Take the action, observe the reward, and transition to the next state.
        - Compute the expected Q-value for the next state:

$$\mathbb{E}[Q(s', a')] = \sum_{a} \pi(a'|s')Q(s', a')$$

        - Update the Q-value using the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \mathbb{E}[Q(s', a')] - Q(s, a)\right]$$

        - Update the current state to the next state.
3. Continue until the agent converges to a policy.

Parameters Explained:

- **α (Learning Rate):** Determines the step size for updating Q-values.
- **γ (Discount Factor):** Reflects the importance of future rewards.
- **ε (Epsilon for Exploration):** Governs the likelihood of exploration under the epsilon-greedy policy.

Output:

- A trained Q-table representing the expected Q-values for each state-action pair.
- Visualization of the learned policy or evaluation metrics, if included.

Applications:
Expected SARSA is particularly useful in scenarios where reducing variance is critical. It can be applied to discrete RL problems like navigation tasks, simple games, or problems with stochastic dynamics.