Reinforcement Learning CSDO8011

Monte Carlo prediction

- Monte Carlo prediction is a technique used in reinforcement learning (RL) to estimate the value function of a Markov decision process (MDP) based on sampled episodes
- Monte Carlo prediction works:
- **Generate Episodes**: The first step is to generate episodes by following the policy of interest. The policy could be deterministic or stochastic. The agent interacts with the environment, selecting actions based on the policy and observing the resulting states and rewards until the episode terminates.
- **Sample Episode Returns**: For each state encountered during an episode, the return is calculated. The return is the total discounted sum of rewards obtained from that state until the end of the episode. Mathematically, the return Gt from time step t can be calculated as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots + \gamma^{T-t-1} R_T$$

- Where:
- Rt is the reward received at time step t,
- T is the time step at which the episode terminates,
- γ is the discount factor, 0≤γ≤1, used to discount future rewards.

Monte Carlo prediction: Advantages and Disadvantages

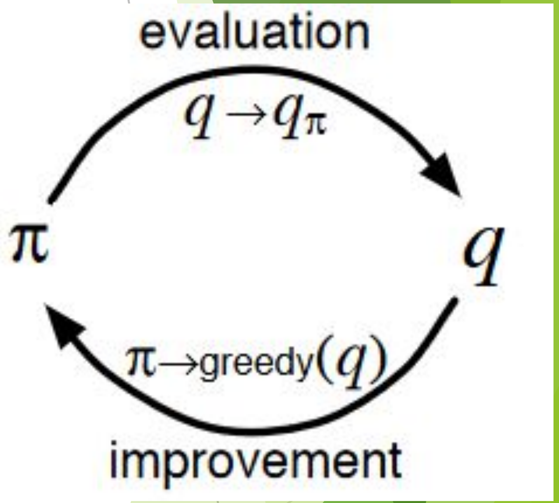
- Advantages:
- It's model-free, it doesn't require knowledge of the underlying dynamics of the environment.
- It can handle stochastic environments and policies.
- It converges to the true value function as the number of episodes approaches infinity, according to the law of large numbers.
- Disadvantages:
- It requires complete episodes to be generated, which may not always be feasible in certain environments.
- It tends to have higher variance compared to other prediction methods like temporal difference learning, especially when dealing with long episodes.
- It can be computationally expensive, especially if many episodes are needed to obtain accurate estimates.

Monte Carlo estimation of action values

- **Target of Estimation:**
  - **Monte Carlo Estimation of Action Values:** In this approach, the goal is to estimate the value of taking a specific action in a given state, i.e., the action-value function Q(s,a). The agent aims to learn the expected return (cumulative rewards) from taking each action in each state.
- **Value Function Update:**
  - **Monte Carlo Estimation of Action Values:** The action-value estimates are updated based on the returns observed for specific state-action pairs. Each update is targeted at improving the estimate of the value of taking a particular action in a particular state.
- **Application:**
  - **Monte Carlo Estimation of Action Values:** This approach is particularly useful in settings where the agent needs to evaluate and improve its policy by understanding the quality of different actions in various states. It is commonly used in reinforcement learning algorithms like Monte Carlo Control and Q-learning.

Monte Carlo control

- Monte Carlo control is a reinforcement learning technique aimed at learning an optimal policy by combining Monte Carlo methods with policy iteration. Its primary goal is to find the optimal policy that maximizes the cumulative rewards an agent can obtain in an environment.
- The Monte Carlo control process:
- **Initialization**:
  - Initialize the state-value function V(s) and the action-value function Q(s,a) arbitrarily or to some initial values.
- **Policy Evaluation**:
  - Use Monte Carlo methods to estimate the value function (either state-value function V(s) or action-value function Q(s,a)) under the current policy.
  - Generate episodes by following the current policy, collect returns, and update the value function estimates based on the observed returns.

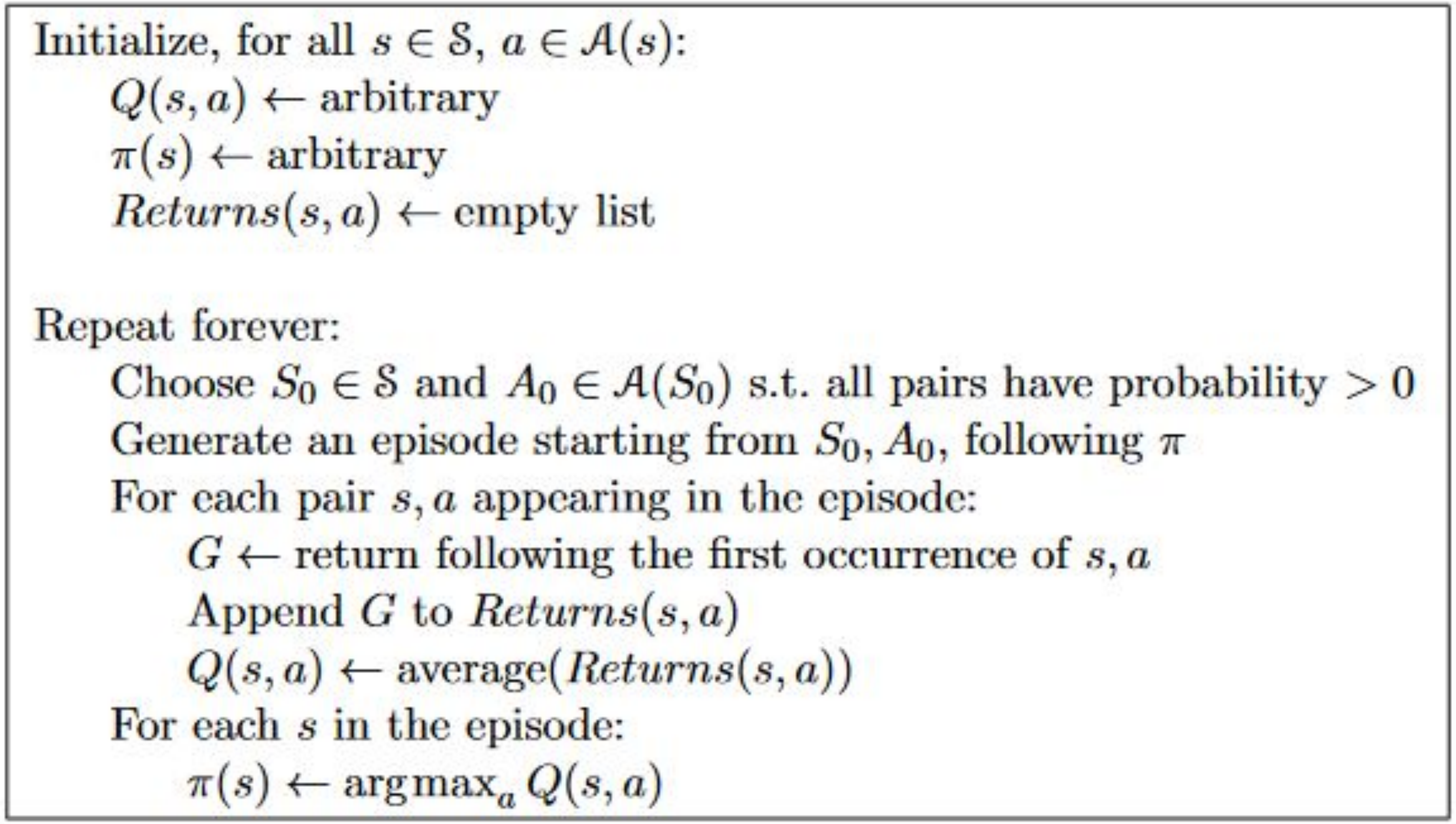


Cont…

- **Policy Improvement:**
  - Improve the policy based on the updated value function estimates. This is typically done by selecting the greedy policy with respect to the current value function estimates.
  - If the agent follows an ε-greedy policy, it may occasionally choose non-greedy actions with probability ε to encourage exploration.
- **Repeat Policy Evaluation and Improvement:**
  - Iterate between policy evaluation and policy improvement steps until convergence criteria are met (e.g., the policy stabilizes or the value function estimates converge).

$$\pi_0 \xrightarrow{E} q_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} q_{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \cdots \xrightarrow{I} \pi_* \xrightarrow{E} q_*,$$

- **Convergence:**
  - Monte Carlo control algorithms typically converge to an optimal policy under certain conditions, such as the Markov property and infinite exploration.
- **Termination:**
  - Terminate the algorithm once the optimal policy is found or after reaching a predefined number of iterations.



```
Initialize, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$:
    $Q(s, a) \leftarrow$ arbitrary
    $\pi(s) \leftarrow$ arbitrary
    $Returns(s, a) \leftarrow$ empty list

Repeat forever:
    Choose $S_0 \in \mathcal{S}$ and $A_0 \in \mathcal{A}(S_0)$ s.t. all pairs have probability $> 0$
    Generate an episode starting from $S_0, A_0$, following $\pi$
    For each pair $s, a$ appearing in the episode:
        $G \leftarrow$ return following the first occurrence of $s, a$
        Append $G$ to $Returns(s, a)$
        $Q(s, a) \leftarrow$ average$(Returns(s, a))$
    For each $s$ in the episode:
        $\pi(s) \leftarrow \arg\max_a Q(s, a)$
```

Temporal-Difference (TD) Learning

- Temporal-Difference (TD) learning is a type of reinforcement learning method
- It is used to predict outcomes, make decisions, or control agents in environments
- TD learning combines aspects of dynamic programming (looking ahead to estimate future rewards) and Monte Carlo methods (learning from actual experience)

Key Aspects of TD Learning

- **Prediction:**
  - TD learning is often used for predicting the value function (expected cumulative future rewards) associated with a given policy
  - It learns to predict the expected cumulative future rewards from any given state
- **Bootstrapping:**
  - One distinctive feature of TD learning is its use of bootstrapping, where value estimates are updated based on other value estimates
  - Instead of waiting until the end of an episode (as in Monte Carlo methods), TD methods update estimates at each time step based on the current estimate and the immediately observed reward
- **Temporal-Difference Error:**
  - At each time step, TD methods compute a temporal-difference error, which is the difference between the predicted value of the current state and a slightly updated estimate based on the observed reward and the estimated value of the next state
  - This error is used to adjust the value estimate for the current state, bringing it closer to the target value

Con…

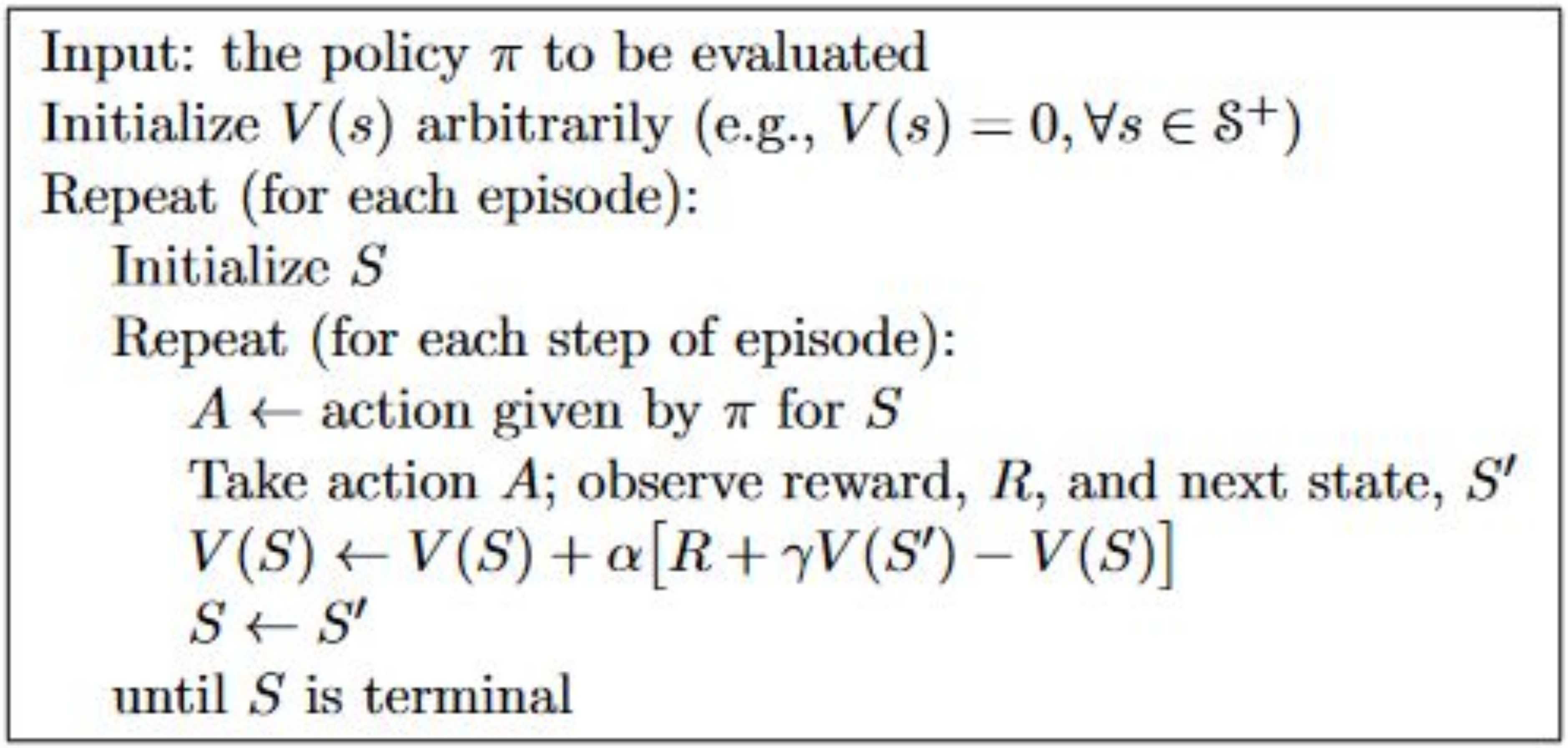
- **TD(0) and TD(λ):**
  - TD(0), also known as one-step TD, updates the value estimate based solely on the current state and the next state's estimated value
  - TD(λ) is a more generalized version that incorporates multiple time steps by weighting each TD error according to a parameter λ (lambda), which controls the balance between one-step updates and multi-step updates
- **On-Policy and Off-Policy Learning:**
  - TD learning can be applied in both on-policy and off-policy settings.
  - On-policy TD methods update the value function based on the current policy being followed by the agent
  - Off-policy TD methods can learn from experiences generated by following a different (possibly exploratory) policy while still estimating the value function for the target policy

The update rule used in TD prediction

- TD involves adjusting the value estimate for a state, based on the observed reward and the estimated value of the next state
- The basic outline of the update rule:
- **TD Error (δ):**
  - At each time step, TD prediction computes a temporal-difference error (δ),
  - Where $\delta = R_{t+1} + \gamma \times V(S_{t+1}) - V(S_t)$
  - δ represents the discrepancy between the predicted value of the current state and the updated estimate based on the observed reward and the estimated value of the next state
  - The temporal-difference error is calculated as the difference between the observed immediate reward plus the estimated value of the next state and the current estimate of the value for the current state.
- **Update Equation:**
  - The update equation for TD(0) prediction is typically expressed as: $V(S_t) \leftarrow V(S_t) + \alpha \times \delta$
  - Where:
    - $V(S_t)$ is the current estimate of the value for state S_t.
    - α is the learning rate, a constant representing the step size for updating the value estimate

Cont…

- **Effect of the Learning Rate (α):**
  - The learning rate α controls the size of the update to the value estimate
  - A higher learning rate means that the value estimate will be adjusted more aggressively based on each observed transition
  - Conversely, a lower learning rate results in slower updates, potentially providing more stability but requiring more data to converge
- **Convergence:**
  - With sufficient data and under certain conditions (e.g., the Markov property holds), TD(0) prediction tends to converge to the true value function
  - Convergence refers to the process by which the estimated value function approaches the true value function as more data is collected and the learning rate and discount factor are appropriately chosen



```
Input: the policy $\pi$ to be evaluated
Initialize $V(s)$ arbitrarily (e.g., $V(s) = 0, \forall s \in \mathcal{S}^+$)
Repeat (for each episode):
    Initialize $S$
    Repeat (for each step of episode):
        $A \leftarrow$ action given by $\pi$ for $S$
        Take action $A$; observe reward, $R$, and next state, $S'$
        $V(S) \leftarrow V(S) + \alpha\big[R + \gamma V(S') - V(S)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

Compare Monte Carlo and TD Prediction

- **TD Prediction (Temporal Difference):**
  - TD prediction methods update value estimates based on a combination of bootstrapping and sampling.
  - They utilize the current estimate of the value function to update the estimate towards a target that incorporates the observed reward and the estimated value of the next state.
  - TD(0) is one of the most common forms, where the update is based solely on the current state and the next state, without considering the entire trajectory or episode.
  - TD methods are typically more sample-efficient than Monte Carlo methods because they update value estimates at each time step, using the current estimate of the value function.
  - TD methods are particularly useful in situations where the entire trajectory or episode is not known in advance.
- **Monte Carlo Prediction:**
  - Monte Carlo prediction methods update value estimates based on complete episodes of experience.
  - They wait until the end of an episode to update the value estimates based on the total observed return (sum of rewards) from that episode.
  - Monte Carlo methods do not use bootstrapping; instead, they rely entirely on the observed returns from complete episodes.
  - Monte Carlo methods can be computationally expensive, especially when episodes are long or when many episodes are required to obtain accurate estimates.
  - They require the full trajectory or episode to be completed before any value estimates are updated.

SARSA (State-Action-Reward-State-Action)

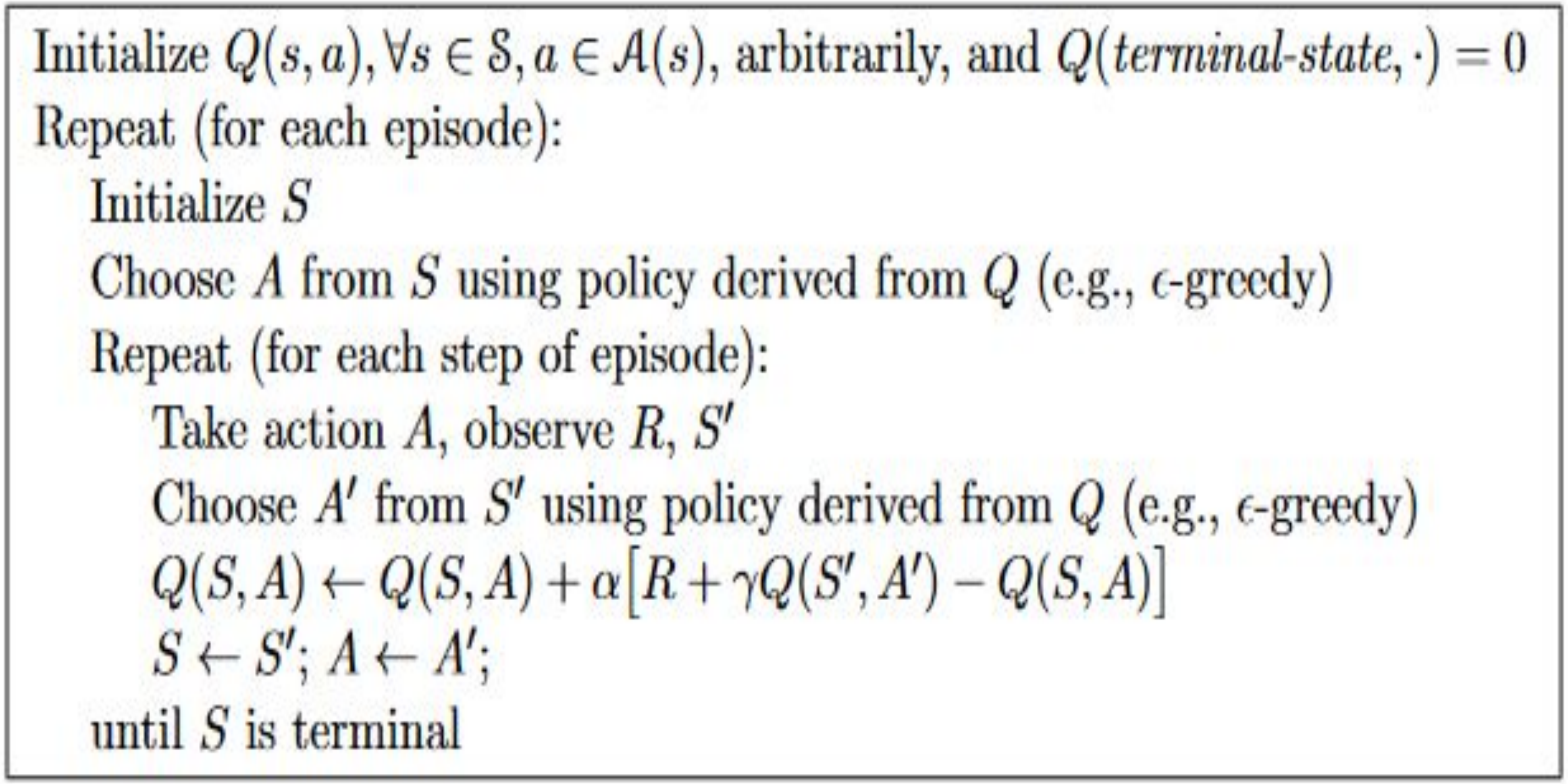
- Sarsa (State-Action-Reward-State-Action) is an on-policy Temporal-Difference (TD) control algorithm used in reinforcement learning for estimating the optimal policy and value function for a given environment
- In Sarsa, the policy is updated based on the action-value function (Q-function), which represents the expected return of taking a specific action in a particular state and following a specific policy thereafter

Sarsa: On-policy TD control

- **Initialization:**
  - Initialize the action-value function Q(s,a) arbitrarily for all state-action pairs (s,a)
  - Choose an initial state s based on the environment's initial conditions.
  - Select an action a using an exploration strategy (e.g., ε-greedy) based on the current policy derived from Q(s,a)
- **Iterative Update:**
- Repeat the following steps until convergence or a predetermined stopping criterion is met
  - Take action a in state s based on the current policy and observe the next state s′ and the reward r received
  - Select the next action a′ using the current policy derived from Q(s′,a′)
  - Update the action-value function using the Sarsa update rule: $Q(s,a) \longleftarrow Q(s,a)+\alpha \times (r+\gamma \times Q(s',a')-Q(s,a))$
  - Update the current state s and action a to the next state s′ and action a′

Cont…

- **Policy Improvement:**
  - Update the policy based on the updated action-value function, typically using an ε-greedy strategy to balance exploration and exploitation
- **Convergence:**
  - Sarsa continues to update the action-value function and policy iteratively until it converges to the optimal policy or a satisfactory approximation of it
- **Exploration vs. Exploitation:**
  - Sarsa balances exploration (trying different actions to discover their values) and exploitation (choosing the best-known action) through its policy, which gradually shifts from exploration to exploitation as the agent learns more about the environment



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
    Repeat (for each step of episode):
        Take action $A$, observe $R$, $S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        $Q(S,A) \leftarrow Q(S,A) + \alpha[R + \gamma Q(S',A') - Q(S,A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    until $S$ is terminal
```

Q-learning

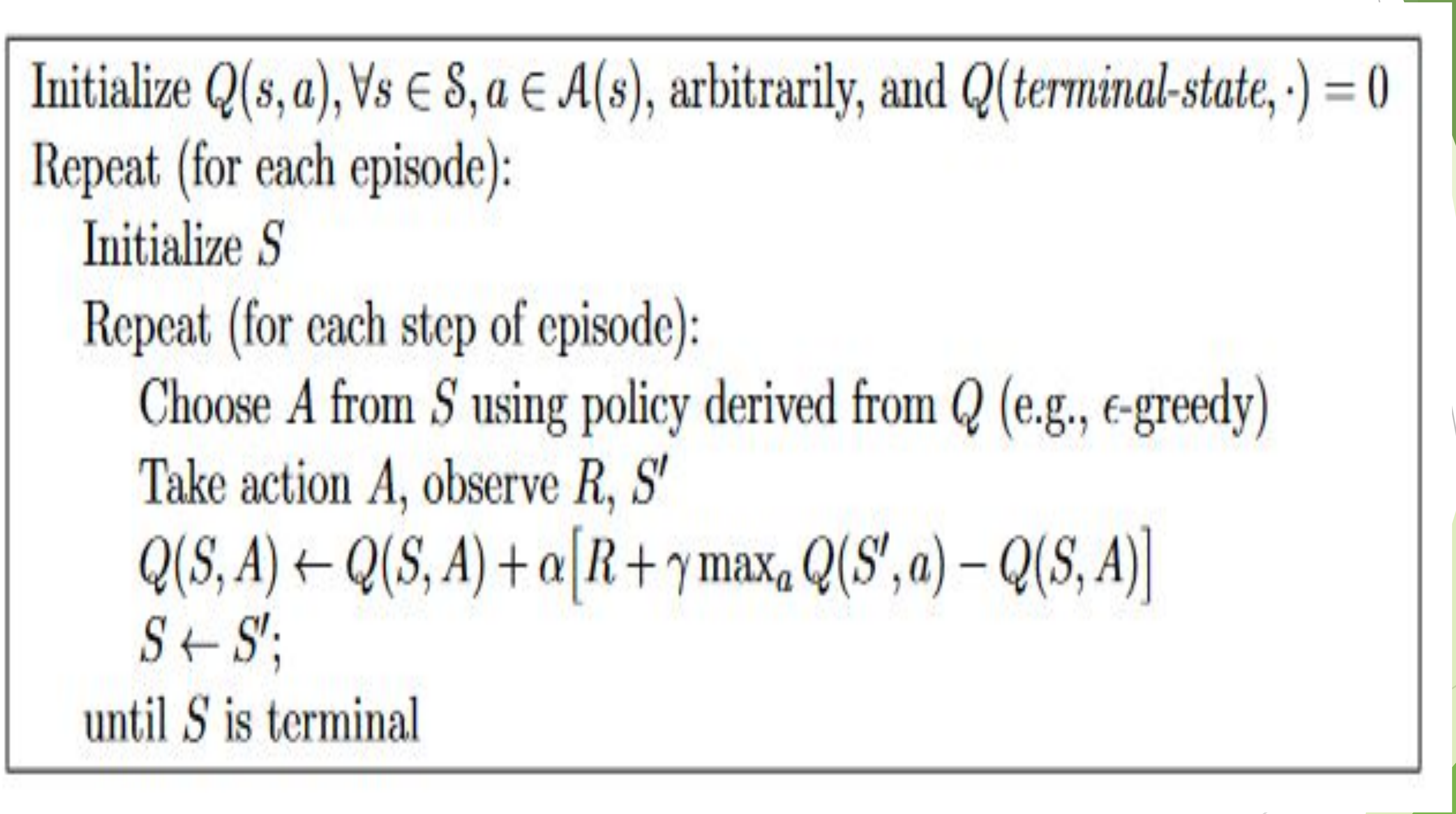
- Q-learning is an off-policy Temporal-Difference (TD) control algorithm used in reinforcement learning for estimating the optimal policy and value function for a given environment
- In Q-learning, the agent learns to estimate the maximum expected return (value) for each state-action pair directly, without needing to follow a specific policy
- This makes Q-learning an off-policy algorithm because it learns from experiences generated by following one policy while simultaneously estimating the value function for another (often, the optimal) policy

Q-learning: Off-policy TD control

- **Initialization:**
  - Initialize the action-value function Q(s,a) arbitrarily for all state-action pairs (s,a)
- **Iterative Update:**
- Repeat the following steps until convergence or a predetermined stopping criterion is met:
  - Choose an initial state ss based on the environment's initial conditions
  - Repeat until termination of episode or a maximum number of steps:
    - Select an action a using an exploration strategy (e.g., ε-greedy) based on the current Q-function
    - Take action a in state s and observe the next state s′ and the reward r received
    - Update the action-value function using the Q-learning update rule: $Q(s,a) \leftarrow Q(s,a)+\alpha\times(r+\gamma\times \text{maxa}' Q(s',a')-Q(s,a))$
  - Update the current state s to the next state s′

Cont…

- **Convergence:**
  - Q-learning continues to update the action-value function iteratively until it converges to the optimal action-value function, $Q*$, which represents the maximum expected return for each state-action pair under an optimal policy
- **Policy Extraction:**
  - After learning the optimal action-value function $Q*$, the optimal policy can be derived by selecting the action with the highest value for each state: $\pi*(s)=\text{argmaxa}Q*(s,a)$
  - The derived policy represents the actions that maximize the expected return from each state according to the learned Q-function.



```
Initialize $Q(s,a), \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$, arbitrarily, and $Q(\textit{terminal-state}, \cdot) = 0$
Repeat (for each episode):
   Initialize $S$
   Repeat (for each step of episode):
      Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
      Take action $A$, observe $R$, $S'$
      $Q(S,A) \leftarrow Q(S,A) + \alpha\left[R + \gamma \max_a Q(S',a) - Q(S,A)\right]$
      $S \leftarrow S'$;
   until $S$ is terminal
```

Example

- Consider an undiscounted Markov Reward Process with two states A and B. The transition matrix and reward function are unknown, but agent have observed two sample episodes:

- $A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow$ terminate
- $B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow$ terminate
- In the above episodes, sample state transitions and sample rewards are shown at each step, e.g. $A + 3 \rightarrow A$ indicates a transition from state A to state A, with a reward of +3. Using first-visit Monte-Carlo evaluation, estimate the state-value function V (A),V (B).Using every-visit Monte-Carlo evaluation, estimate the state-value function V (A),V (B). Draw a diagram of the Markov Reward Process that best explains these two episodes. Show rewards and transition probabilities on the diagram.

Example

- Consider an undiscounted Markov Reward Process with two states A and B. Agent have observed 8 sample episodes of experience as:

- A, 0, B, 0
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 1
- B, 0
- The first episode started in state A, transitioned to B with a reward of 0, and then terminated from B with a reward of 0. The other seven episodes start from B and terminate immediately. What is V (A), V (B)? Draw a diagram of the Markov Reward Process. Show rewards and transition probabilities on the diagram.

Example

- Apply Q-Learning to following scenario: Consider a grid world environment with the following layout: States: S0, S1, S2, S3 Actions: Up, Down, Left, Right. Rewards: - Transition to S3 gives a reward of +10 (terminal state). All other transitions give a reward of -1. Assume the following parameters: Discount factor (γ) = 0.9. Learning rate (α) = 0.5. Scenario: The agent starts in state S0 and takes the following sequence of actions: Moves Right to S1. Moves Down to S3 (terminal state).

Example

- Apply SARSA to following scenario: Consider a grid world environment with the following layout: States: S0, S1, S2, S3 Actions: Up, Down, Left, Right Rewards: - Transition to S3 gives a reward of +5 (terminal state). Transition to S2 gives a reward of -2. All other transitions give a reward of -1. Assume the following parameters: Discount factor (γ) = 0.8, learning rate (α) = 0.6, Scenario: The agent starts in state S0 and takes the following sequence of actions: Moves Right to S1. Moves Down to S2. Moves Right to S3 (terminal state).