Temporal Difference

Understanding Temporal Difference Learning

Temporal difference (TD) Learning

Temporal difference (TD) learning is a reinforcement learning algorithm that estimates the value function of a policy and updates the estimate based on the difference between predicted and actual rewards from a state transition. It is more efficient than the Monte Carlo method as it can update its estimates after every time step and is better at handling incomplete sequences of information than the dynamic programming method.

Recap

The TD learning algorithm was introduced by Richard S. Sutton in 1988. The reason the TD method became popular is that it combines the advantages of DP and the MC method. But what are those advantages?

The Dynamic Programing (DP) method uses the Bellman equation to compute the value of a state, allowing for bootstrapping. This means that the value of a state can be estimated based on the value of the next state, rather than waiting until the end of the episode.

However, the disadvantage of DP is that we can apply the DP method only when we know the model dynamics of the environment. That is, DP is a model-based method and we should know the transition probability in order to use it. When we don't know the model dynamics of the environment, we cannot apply the
DP method.

Recap

Monte Carlo method—The advantage of the MC method is that it is a model- free method, which means that it does not require the model dynamics of the environment to be known in order to estimate the value and Q functions.

However, the disadvantage of the MC method is that in order to estimate the state value or Q value we need to wait until the end of the episode, and if the episode is long then it will cost us a lot of time. Also, we cannot apply MC methods to continuous tasks (non-episodic tasks).

Could you provide a few examples of continuous tasks or non-episodic tasks that are a part of your daily routine?

Example: Robot Arm Control

Continuous State Space: Each joint can have an angle within a specific range (e.g., -180 to +180 degrees), and there are multiple joints, leading to a high-dimensional continuous state space.

Continuous Action Space: The action space consists of the torques or forces applied to the joints of the robot arm to control its movement. These torques or forces can also be continuous, allowing for a wide range of possible actions.

Infinite States and Actions: The combination of the continuous state space (e.g., infinite angles for each joint) and the continuous action space (e.g., infinite possible torques or forces for each joint) results in an infinite number of possible states and actions. This makes it impractical to store and process all state-action pairs explicitly.

The agent needs to learn a policy that can navigate this infinite space to achieve tasks such as reaching a target position or manipulating objects, requiring sophisticated algorithms that can efficiently explore and exploit the continuous space to find optimal actions.

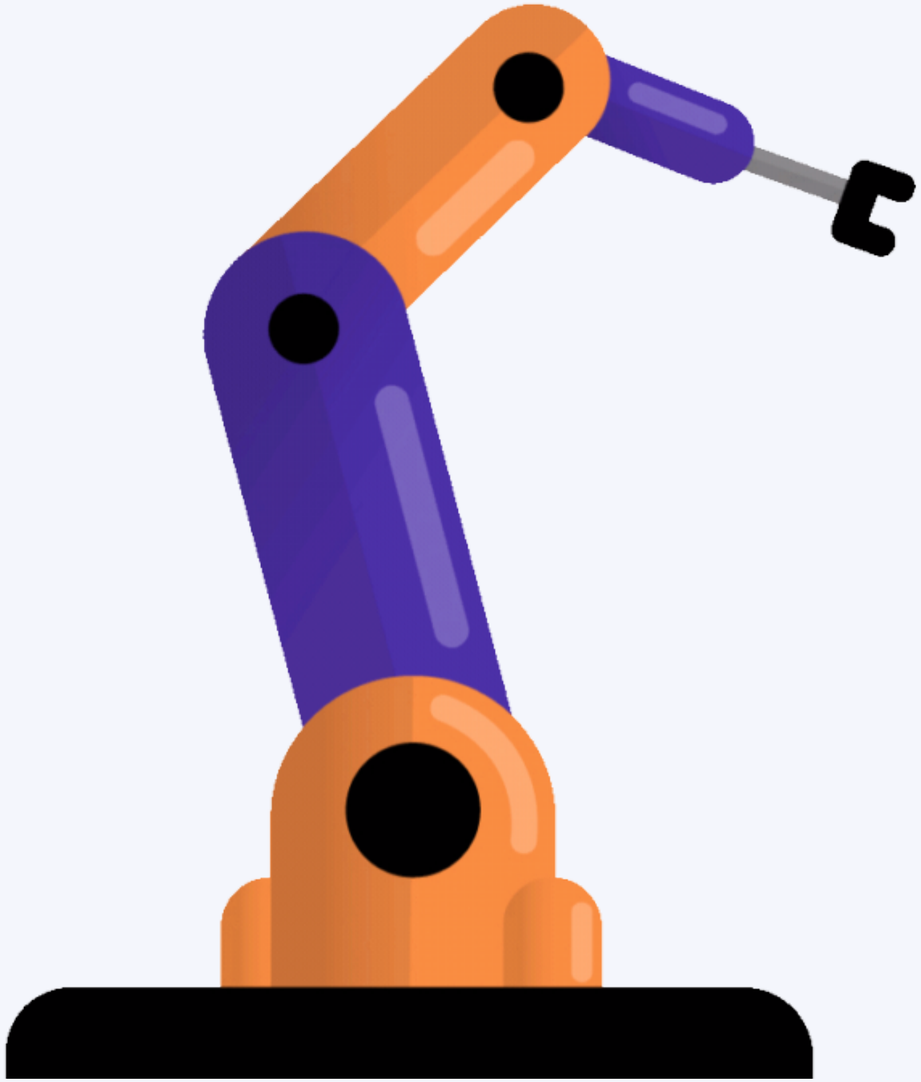
TD prediction

TD Prediction

- In the TD prediction method, we estimate the value function by inputting the policy.
- Unlike the Monte Carlo method, we don't need to know the model dynamics of the environment to calculate the value function or Q function.
- TD learning bootstraps like DP, so we do not have to wait for the episode to end.

So how is the update rule of TD learning created?

- In the MC method, we estimate the value of a state by taking its return.
- However, a single return value cannot approximate the value of a state perfectly. So, we generate N episodes and compute the value of a state as the average return of a state across N episodes:
- But with the MC method, we need to wait until the end of the episode to compute the value of a state and when the episode is long, it takes a lot of time.
- One more problem with the MC method is that we cannot apply it to non-episodic tasks (continuous tasks).



TD control

In the control method, our goal is to find the optimal policy, so we will start off with an initial random policy and then we will try to find the optimal policy iteratively. In the previous chapter, we learned that the control method can be classified into two categories:

- On-policy control
- Off-policy control

Understanding On-Policy and Off-Policy Methods

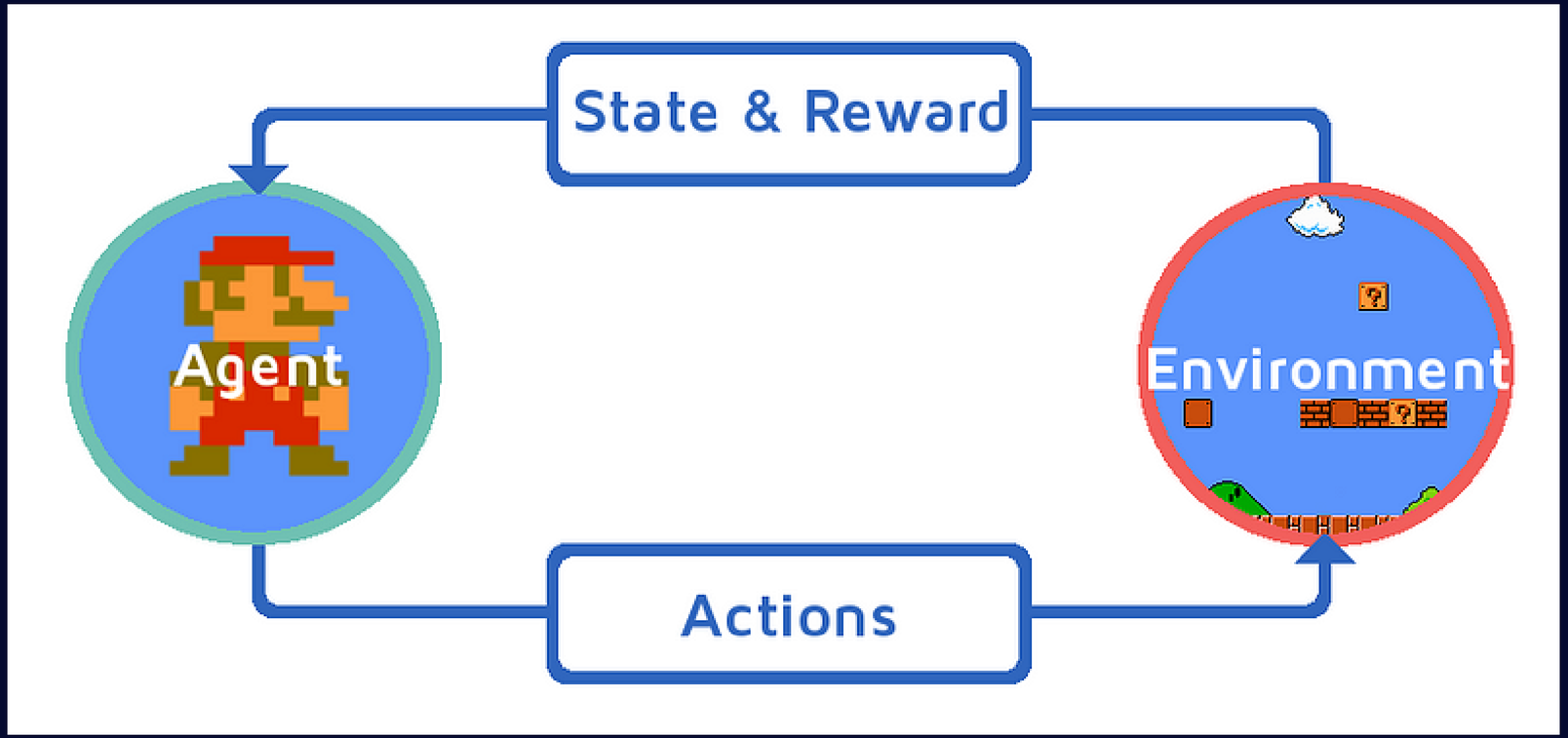
When it comes to reinforcement learning, there are two primary methods for controlling the agent's behavior.

On-Policy Control: The agent utilizes a single policy to generate episodes and subsequently enhance that same policy iteratively until the optimal one is found.

Off-Policy Control: The agent employs one policy to behave, but seeks to improve a different policy through the control method. we generate episodes using one policy and we try to improve a different policy iteratively to find the optimal policy.

On-policy TD control – SARSA

State-Action-Reward-State-Action



SARSA

- The SARSA method involves initializing the Q function with random values or zeros, extracting a policy from this function, and updating the Q function on every episode.
- The resulting optimal policy is not initially optimal, but it is gradually refined.
- The SARSA method uses an epsilon-greedy policy, which selects a random action with probability epsilon and the best action with probability 1-epsilon, instead of a greedy policy, which always selects the action with the maximum Q value.

SARSA Algorithm for Grid World

Let's consider a simple grid world with 3x3 cells, where the agent can move up, down, left, or right. The grid has a start state S and a goal state G. The agent receives a reward of -1 for each step and a reward of +10 for reaching the goal state. The discount factor γ is set to 0.9.

Initialization:

- Initialize the action-value function $Q(s, a)$ arbitrarily for all s and a.
- Initialize the state S to the start state.
- Choose an action A using an exploration policy (e.g., epsilon-greedy).

Algorithm:

1. **Repeat** for each time step:
    - Take action A and observe the reward R and the next state S'.
    - Choose the next action A' using the same exploration policy.
    - Calculate the TD target:

$$\text{TD target} = R + \gamma \cdot Q(S', A')$$

    - Update the action-value function:

$$Q(S, A) \leftarrow Q(S, A) + \alpha \cdot (\text{TD target} - Q(S, A))$$

      where α is the learning rate.
    - Set S to S' and A to A'.

Final Optimal Policy:

- The final optimal policy π can be derived from the learned action-value function $Q(s, a)$.
- For each state s, choose the action a that maximizes $Q(s, a)$:

$$\pi(s) = \arg\max_a Q(s, a)$$

- The optimal policy π specifies the best action to take in each state to maximize the expected cumulative reward over time.

Off-policy TD control – Q learning

Off-Policy Q Learning

Q-learning is a powerful algorithm in reinforcement learning that can help machines learn to make optimal decisions in complex environments. Here are some key features of Q-learning:

1. Off-policy: Q-learning is an off-policy algorithm, which means that it learns from actions that are not necessarily the optimal ones. Instead, it explores a range of possible actions and learns from the rewards that each action yields.
2. Value-based: Q-learning is a value-based algorithm, which means that it tries to learn the values of each state-action pair in the environment. By doing so, it can determine the optimal policy for selecting actions that maximize the expected cumulative reward.
3. Model-free: Q-learning is a model-free algorithm, which means that it does not require a model of the environment to learn. Instead, it learns directly from experience by interacting with the environment and observing the rewards that each action yields.
4. Convergence: Q-learning is guaranteed to converge to the optimal policy in a finite-state Markov decision process with a finite number of actions. However, in more complex environments, convergence may take longer or may not be guaranteed.

Overall, Q-learning is a powerful and flexible algorithm that can be applied to a wide range of reinforcement learning problems.

Comparing the DP, MC, and TD methods

Exploring Reinforcement Learning Algorithms

We've explored several key algorithms: Dynamic Programming (DP) with value iteration and policy iteration, Monte Carlo (MC) methods, and Temporal Difference (TD) learning. These algorithms are crucial for finding the optimal policy in classic reinforcement learning. Let's recap the differences between them:

1. Dynamic Programming (DP):
    - Model-based method: It uses the model dynamics of the environment to compute the optimal policy.
    - Limitation: Cannot be applied without the model dynamics of the environment.
2. Monte Carlo (MC):
    - Model-free method: Computes the optimal policy without relying on the model dynamics of the environment.
    - Limitation: Applicable only to episodic tasks, not continuous tasks.
3. Temporal Difference (TD) Learning:
    - Model-free method: Combines elements of DP (bootstrapping) and MC (model-free).
    - Advantage: Can be applied in a wide range of tasks and environments.

Lab: Computing the optimal policy using Q learning

- TD learning is a unique approach that utilizes both DP and the MC method, as we learned in the first section of this chapter. While TD learning is a model-free method, it also bootstraps like DP. Later on, we delved into how to perform prediction tasks with TD learning, and we explored the algorithm of the TD prediction method.
- Moving forward, we examined how TD learning can be used for control tasks. Initially, we learned about the on-policy TD control method known as SARSA. Subsequently, we explored the off-policy TD control method called Q learning. We also learned how to apply these methods in the Frozen Lake environment to discover the optimal policy.
- Furthermore, we made a distinction between the SARSA and Q learning methods. SARSA is an on-policy algorithm, implying that we use a single epsilon-greedy policy to select an action and to compute the Q value of the next state-action pair. However, Q learning is an off-policy algorithm that uses a greedy policy to compute the Q value of the next state-action pair, while selecting an action using an epsilon-greedy policy.


Thank You!