Giorgio Mendoza

RBE595-S24-S04

Dr. Dadkhah Terani

Reinforcement Learning

HW4

1- Between DP (Dynamic Programming), MC (Monte-Carlo) and TD (Temporal Difference), which one of these algorithms use bootstrapping? Explain.

Among Dynamic Programming, Monte Carlo, and Temporal Difference methods, it is the Temporal Difference method that uses bootstrapping.

Bootstrapping refers to the idea of updating estimates based on other estimated values. In TD methods, the value of the current state is updated not based on the final outcome or return (as in MC methods), but based on the estimated value of the following state. This makes TD methods a blend of DP and MC approaches: they update estimates based on experience (like MC) but also use existing value estimates (like DP).

In contrast, Dynamic Programming methods also use bootstrapping, as they update the value of a state based on the estimated values of successor states. However, DP assumes a complete and accurate model of the environment.

Monte Carlo methods do not use bootstrapping. They update the value of a state based on the actual returns received (i.e., the sum of rewards from that state until the end of the episode), not on other estimated values.

2- We mentioned that the target value for TD is $[R_t+1 + \gamma(S_t+1)]$. What is the target value for Monte-carlo, Q-learning, SARSA and Expected-SARSA.

The target values for Monte Carlo, Q-learning, SARSA, and Expected-SARSA vary due to the different ways these methods estimate and update their value functions, for example:

Monte Carlo:

Target value: G_t

In Monte Carlo methods, the target is the actual return Gt, which is the sum of rewards from time step t until the end of the episode. This method relies on complete episodes and does not bootstrap.

Q-learning:

Target value: $R_{t+1} + \gamma max_{a'}$ $\mathbf{Q}(S_{t+1}, a_{t+1}\ )$

Sarsa:

Target value: $R_{t+1} + \gamma\mathbf{Q}(S_{t+1}, a_{t+1}\ )$

SARSA is an on-policy TD control algorithm. Its target value includes the reward R_{t+1} and the value of the next state-action pair $\mathbf{Q}(S_{t+1}, a_{t+1}\ )$ as dictated by the current policy. This means the policy used to select actions is the same policy used to update the value estimates.

Expected-SARSA:

Target value: $R_{t+1} + \gamma\mathbf{E}[\mathbf{Q}(S_{t+1}, a_{t+1}\ )]$

Expected-SARSA, like SARSA, is an on-policy method. However, instead of using the Q-value of the actually taken action in the next state (as in SARSA), it takes the expected value over all possible actions in the next state, according to the current policy. This expected value is the average of the Q-values of all possible actions in the next state, weighted by the probability of taking each action under the current policy.

3- What are the similarities between TD and MC?

Policy Evaluation and Control: Both TD and MC methods can be used for policy evaluation (estimating the value function of a policy) and for control (finding an optimal policy).

Learning from Experience: They are experience-based methods, meaning they learn directly from episodes of experience. They do not require a model of the environment's dynamics (i.e., the transition probabilities and rewards), unlike Dynamic Programming methods.

Incremental Updates: Both methods update estimates incrementally over episodes. In MC, this update occurs at the end of each episode, while in TD, it can happen at every time step.

Sample-based Estimates: TD and MC use samples (experienced transitions and rewards) to estimate the value functions. This is in contrast to Dynamic Programming, which requires a complete model of the environment.

Flexibility with Partial Episodes: Both methods can be adapted to work with incomplete episodes, which is useful in continuing tasks where there is no natural terminal state.

Policy Dependency: Both methods can be used in conjunction with either on-policy or off-policy learning strategies, although the specific implementation details can vary.

These similarities underline their roles as key methods in reinforcement learning, particularly in scenarios where having a complete model of the environment is impractical or impossible. However, it's important to note that they also have significant differences, particularly in how they update their value estimates (TD uses bootstrapping, while MC relies on full episodes to reach a terminal state).

4- Assume that we have two states x and y with the current value of $V(x) = 10$, $V(y) = 1$. We run an episode of $\{x, 3, y, 0, y, 5, T\}$. What's the new estimate of $V(x)$, $V(y)$ using TD (assume step size $\alpha = 0.1$ and discount rate $\gamma = 0.9$)

```
#given values
V_x = 10  # initial value of state x
V_y = 1   # initial value of state y
alpha = 0.1  # step size
gamma = 0.9  # discount rate
# episode given
episode = [('x', 3), ('y', 0), ('y', 5), ('T', 0)]  # format: (state, reward)
# temporal difference (TD) update formula:
# V(s) = V(s) + alpha * (reward + gamma * V(s') - V(s))
# where V(s) is the current state value, V(s') is the next state value, reward is the reward for
moving to the next state.
# Applying TD update rule for the episode
```

```
#since it's an episode, go through each transition and update the values

#dictionary to hold current values of states

V = {'x': V_x, 'y': V_y, 'T': 0}  # terminal state value is always 0

# applying TD updates across episode

for i in range(len(episode) - 1):

    current_state, reward = episode[i]

    next_state = episode[i + 1][0]

    # TD update rule

    V[current_state] = V[current_state] + alpha * (reward + gamma * V[next_state] -
V[current_state])

# The updated values for V(x) and V(y)

V_x_updated = V['x']

V_y_updated = V['y']

V_x_updated, V_y_updated
```

Result

```
(9.39, 1.391)
```

5- Can we consider TD an online (real-time) method and MC an offline method? Why?

Temporal Difference (TD) methods can be considered online methods. In TD learning, the agent updates its value function after each step or few steps within an episode. This means that the agent doesn't have to wait until the end of the episode to learn; it can incrementally learn and update its policy after each step based on the latest experience. This property makes TD methods particularly suitable for learning in an ongoing, real-time environment where waiting until the end of an episode might be impractical or impossible.

Monte Carlo (MC) methods, on the other hand, are often considered offline methods because they require a complete episode to be finished before the value function can be updated. MC updates happen only after the final outcome is known, which means it waits until the end of an episode to make an update based on the entire sequence of observed rewards. This characteristic makes MC methods less suitable for real-time updates, especially in environments where episodes are long or continuous without terminal states.

6- Does Q-learning learn the outcome of exploratory actions? (Refer to the Cliff walking example)

Q-learning is an off-policy reinforcement learning algorithm, which means that it learns the value of the optimal policy independently of the agent's actions, including the exploratory actions.

In the context of the Cliff Walking example, where an agent must find a path from a start state to a goal state while avoiding a "cliff" that results in a large negative reward, Q-learning will learn the values of actions regardless of whether they are part of the current policy or not. It does this by updating its Q-values using the maximum future reward that can be obtained from the next state, regardless of the action that is actually taken. This property allows it to learn the optimal policy even if the agent is exploring less optimal actions.

During exploration, the agent may choose a suboptimal action to learn about the environment. In Q-learning, even if an exploratory action is taken and it leads to a suboptimal outcome (like falling off the cliff), the algorithm updates the Q-value for the state-action pair based on the maximum Q-value for the next state, not the Q-value of the exploratory action. This helps the agent to learn the optimal policy over time, which will avoid the cliff, even though it is exploring.

So, while Q-learning does experience the outcomes of exploratory actions, its learning process focuses on what would happen if the optimal action were always taken at the next step, not necessarily the exploratory action. This distinction is what makes Q-learning an off-policy method.

7- What is the advantage of Double Q-learning over Q-learning?

These are some of the advantages of Double Q-learning over Q-learning:

Reduced Overestimation: By decoupling the action selection from the value update, Double Q-learning reduces the positive bias that is introduced by the maximization step in standard Q-learning.

Improved Convergence Properties: Due to the reduction in overestimation, Double Q-learning can converge to the true Q-values more accurately, which can result in better policy performance, especially in environments with stochastic transitions and rewards.

Robustness to Noise: Double Q-learning is generally more robust to noise because it does not always use the overestimated Q-values for updates, which can be particularly beneficial in environments with high variance in rewards.

Stability in Learning: The algorithm can exhibit more stable learning behavior because it is less prone to the feedback loop of overestimation that can occur in standard Q-learning.

Double Q-learning offers these advantages, but it may require more memory to store two Q-tables and can be more complex to implement than standard Q-learning.