Giorgio Mendoza

RBE595-S24-S04

Dr. Dadkhah Terani

Reinforcement Learning

HW5

1- The first episode of an agent interacting with an environment under policy $\pi\pi$ is as follows:

Timestep	Reward	State	Action
0		X	U1
1	16	X	U2
2	12	X	U1
3	24	X	U1
4	16	T	

Assume discount factor, γ=0.5, step size $\alpha = 0.1$ and q_π is initially zero.
What are the estimates of $q_\pi\,(X, U1)$ and $q_\pi\,(X, U2)$ using 2-step SARSA?

```
# Given values
gamma = 0.5  # discount factor
alpha = 0.1  # step size
rewards = [0, 16, 12, 24, 16]  # rewards at each timestep
q_values = {'XU1': 0, 'XU2': 0}  # initial Q values for state-action pairs


# Updating Q(X, U1) using 2-step SARSA
# Q(X, U1) = Q(X, U1) + alpha * (R1 + gamma * R2 + gamma^2 * Q(X, U1) - Q(X, U1))
# Since Q(X, U1) is initially zero, the last term cancels out in the first update
q_values['XU1'] = q_values['XU1'] + alpha * (rewards[1] + gamma * rewards[2] + gamma**2 *
q_values['XU1'] - q_values['XU1'])


# Updating Q(X, U2) using 2-step SARSA
# Q(X, U2) = Q(X, U2) + alpha * (R2 + gamma * R3 + gamma^2 * Q(T, -) - Q(X, U2))
# Since Q(T, -) is zero (not provided but usually terminal states have zero value) and Q(X, U2)
is initially zero, the last term cancels out
```

```
q_values['XU2'] = q_values['XU2'] + alpha * (rewards[2] + gamma * rewards[3] + gamma**2 * 0 -
q_values['XU2'])
q_values
```

```
Result
{'XU1': 2.2, 'XU2': 2.4000000000000004}
```

After applying the 2-step SARSA update rule, the action-value estimates are as follows:

$q_{\pi}(X, U1)$ *is approximately* $2,2$

$q_{\pi}(X, U2)$ *is approximately* $2,4$

2- What is the purpose of introducing Control Variates in per-decision importance sampling?

Importance sampling is a method used to estimate the value of a target policy while following a different behavior policy. However, this method can lead to high variance in the estimates, especially when the behavior policy differs significantly from the target policy.

Control variates are a statistical technique used to reduce the variance of an estimator by introducing additional information about the system being modeled. The purpose of introducing control variates in per-decision importance sampling is to reduce the variance of the importance sampling estimator without introducing bias.

In per-decision importance sampling, each decision (or action) taken in an episode is weighted by the ratio of the probability under the target policy to the probability under the behavior policy. This can lead to high variance if the probabilities under the two policies are very different.

By introducing a control variate, one can use the knowledge of the expected value of a related quantity to reduce the variance of the estimate. In the case of reinforcement learning, a common choice for a control variate is the value function of the behavior policy, which is known or can be estimated from data.

The control variate is used to adjust the importance sampling weights so that the variability due to parts of the system that are well understood (i.e., the behavior policy's value function) does not contribute to the variance of the overall estimate. This results in a more stable and reliable estimate of the value of the target policy, even when the behavior policy differs significantly.

So in a few words, importance sampling estimator is useful for:

- Reduce the variance of the importance sampling estimator.
- Increase the stability and reliability of the value estimates.
- Improve the efficiency of the estimation process, making it more practical for applications in reinforcement learning.

3- In off-policy learning, what are the pros and cons of the Tree-Backup algorithm versus off-policy SARSA (comment on the complexity, exploration, variance, and bias, and others)?

Off-policy learning refers to the process of learning the value of the optimal policy independently of the agent's actions. The Tree-Backup algorithm and the off-policy SARSA (usually referred to as SARSA(λ) in an off-policy context) are two methods used in this approach, each with its own advantages and disadvantages:

Tree-Backup Algorithm

Pros:

Backup Depth: Tree-Backup allows for multi-step lookahead in a single update, which can lead to more informed and potentially faster learning.

Lower Variance: It can have lower variance in updates compared to methods using importance sampling because it does not rely on the potentially large importance sampling ratios that can occur when the behavior policy is very different from the target policy.

Bias Reduction: By considering multiple future action possibilities, it can reduce the bias that might be introduced if one were only to consider a single action in the future as in traditional SARSA.

Cons:

Complexity: Tree-Backup is computationally more complex due to the requirement to consider multiple future actions at each step.

Sample Efficiency: It may require more samples to learn effectively compared to methods that bootstrap from existing value estimates.

Off-Policy SARSA (SARSA(λ))

Pros:

Flexibility: Off-policy SARSA can learn about the optimal policy while following an exploratory or even a completely random policy, which provides great flexibility in learning.

Exploration: Using eligibility traces (the λ in SARSA(λ)), it can attribute credit to states and actions visited in the past, leading to more effective exploration.

Simplicity: It is conceptually simpler than Tree-Backup since it follows a straightforward update rule based on the current policy.

Cons:

Higher Variance: It can suffer from high variance when the behavior policy is very different from the target policy due to the reliance on importance sampling ratios.

Bias: It can be biased if the behavior policy does not sufficiently explore the state-action space.

Complexity with λ: Introducing eligibility traces (λ) adds complexity to the algorithm and requires careful tuning of the λ parameter.

The choice between these two algorithms can depend on the specific requirements of the task, the computational resources available, the need for fast learning, and the extent to which the behavior policy differs from the target policy. The best choice can depend on the empirical performance in the specific environment where the learning is taking place.

3- Exercise 7.4 of the textbook (page 148).

Exercise 7.4 Prove that the n-step return of Sarsa (7.4) can be written exactly in terms of a novel TD error, as

$$G_{t:t+n} = Q_{t-1}(S_t, A_t) + \sum_{k=t}^{\min(t+n,T)-1} \gamma^{k-t} \left[R_{k+1} + \gamma Q_k(S_{k+1}, A_{k+1}) - Q_{k-1}(S_k, A_k) \right].$$

```
from sympy import symbols, Eq, IndexedBase, Idx, Sum

# define the symbols
```

```
gamma = symbols('gamma', real=True, positive=True)  # Discount factor
R = IndexedBase('R')  # Rewards
Q = IndexedBase('Q')  # Q values
t, T, n, k = symbols('t T n k', integer=True)
i = Idx('i', (0, n-1))  # Index for summation
# define the traditional n-step return using symbolic summation
traditional_n_step_return = Sum(gamma**i * R[t+i+1], (i, 0, n-1)) + gamma**n * Q[t+n,
symbols('A')]
# define the novel TD error based n-step return using symbolic summation
td_error_summation = Sum(gamma**(k-t) * (R[k+1] + gamma*Q[k+1, symbols('A')] - Q[k,
symbols('A')]), (k, t, t+n-1))
novel_TD_error_n_step_return = Q[t, symbols('A')] + td_error_summation
# set the traditional n-step return equal to the novel TD error-based n-step return
proof_eq = Eq(traditional_n_step_return.doit(), novel_TD_error_n_step_return.doit())
# simplify the proof equation
simplified_proof = proof_eq.simplify()
simplified_proof
```

Result

```
Eq(Q[t, A] + gamma**(-t)*Sum(gamma**k*(gamma*Q[k + 1, A] - Q[k, A] + R[k + 1]),
(k, t, n + t - 1)), gamma**n*Q[n + t, A] + Sum(gamma**i*R[t + i + 1], (i, 0, n
- 1)))
```

The symbolic proof results in the following equation:

$$Q[t, A] + \gamma^{-t} \sum_{k=t}^{n+t-1} \gamma^k \left(\gamma Q[k+1, A] - Q[k, A] + R[k+1]\right) = \gamma^n Q[n+t, A] + \sum_{i=0}^{n-1} \gamma^i R[t+i+1]$$

This equation represents the equivalence between the traditional n-step return on the right-hand side and the novel TD error-based n-step return on the left-hand side. The equation asserts that both forms yield the same value, which is the sum of the rewards and the value of the state-action pair at time $t + n$, appropriately discounted by γ.

The left-hand side incorporates the TD errors across the n-step span, while the right-hand side directly sums the rewards and adds the discounted value of the future state-action pair.