RBE595 - Week 7 Assignment

Keith Chester

Due date: February 22, 2023

Problem 1

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

Both dynamic programming (DP) and temporal difference (TD) utilize bootstrapping, while Monte-Carlo (MC) does not. Bootstrapping is the process of updating value estimates for a given state/action using estimates from other states/actions *without* waiting for the final outcome. Both DP and TD do this, while MC waits until the episode terminates to update visited state/action pairs to update its value estimates.

Problem 2

We mentioned that the target value for TD is $[R_{t+1} + \gamma V(S_{t+1})]$. *What is the target value for Monte-Carlo, Q-learning, SARSA and Expected-SARSA.*

Monte-Carlo

The target value update for Monte-Carlo is simply:

$$G_t \tag{1}$$

Q-Learning

The target value for a state/action pair is the max of the expected future reward obtained when taking the best possible action available in the next state. It utilizes the Bellman equation:

$$\Big(R_{t+1} + \gamma * \max\big(Q(s', a')\big)\Big) \tag{2}$$

SARSA

In SARSA (state, action, reward, state, action...) the target value of a given state/action pair is the estimated future rewards obtained by following the current policy. SARSA's target value update is:

$$\Big(R_{t+1} + \gamma Q(s', a')\Big) \tag{3}$$

Expected SARSA

In expected SARSA, teh target value of a given action/pair is the expected rewards obtained by following the current policy, but also updates the Q-value for a given state/pair pair with the target value update:

$$\Big(R_{t+1} + \gamma \sum_a \big(\pi * Q(s', a')\big)\Big) \tag{4}$$

...with notable newcomer π being our policy (probability of choosing a specific action). Thus we are summing the probability of all possible choices times the expected Q value.

Problem 3

What are the similarities of TD and MC?

Both TD and MC are model-free approaches - they do not need an exact representation of the environment or to know all of its rules to create an agent capable of navigating a given environment. They both apply sample-based updates; ie it does not need to compute every possible outcome (as is done in dynamic programming) and thus converges to an optimal policy over its experience (often the true-value optimal policy if certain conditions/requirements are met).

Problem 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 $\gamma = 0.9$)

First we take the TD update equation for the state value function for $V(x)$:

$$V(x) \leftarrow V(x) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(x)\Big] \tag{5}$$

...we can plug in the values provided within our problem and begin calculating. Note that $V(S_{t+1}) = V(y) = 1$:

$$V(x) \leftarrow 10 + 0.1\Big[3 + 0.9V(y) - 10]10 + 0.1[3 + 0.9 \cdot 1 - 10\Big] \tag{6}$$

...and we just calculate through to update our $V(x)$:

$$V(x) \leftarrow 10 + 0.1(-6.1) \tag{7}$$

$$V(x) \leftarrow 9.39 \tag{8}$$

We can then perform a similar path for $V(y)$.

$$V(y) \leftarrow V(y) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(y)\Big] \tag{9}$$

... here $V(S_{t+1}) = V(y) = 5$:

$$V(y) \leftarrow 1 + 0.1\Big[0 + 0.9V(y) - 1\Big] = 1 + 0.1\Big[0 + 0.9 \cdot 5 - 1\Big] \tag{10}$$

...calculating through:

$$V(y) \leftarrow 1 + 0.1[3.6] = 1.36 \tag{11}$$

Problem 5

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

First, we must define online and offline methods of learning. An online method learns while actively performing its actions - ie it updates its esimates and thus policy while actively performing its given task. By contrast, an offline policy waits until after the agent is working through its environment to update its policy/value estimates (after the episodes concludes).

By these definitions, we can agree with the statement proposed. TD updates its value estimates as it moves through the given environment at each step through an episode, while MC updates its value estimation only *after* the episode concludes.

Problem 6

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

In our cliff-walking example, our agent is moving through a two dimensional grid environment trying to reach a given goal without falling "off a cliff" via empty squares. The environment is entirely unknown to the agent, forcing it to take risky exploratory moves. If we use Q-learning, it estimates the value of a given action by iteratively updating it's Q-value estimation for each state/action pair based on observed rewards. Thus through trial and error of these exploratory actions Q-learning is converging on an expected total return for each state/action pair it could face, finding us eventually an optimal policy.

Exploratory actions may move us to bad states with terrible outcomes and poor rewards, but Q-learning properly updates the expected future rewards appropriately. In our cliff-example, our agent in simulation may very well take exploratory actions that cause it to "fall" down our cliff, but over enough explorations will find the correct path to safety.

...assuming that the cliff is a simulation, and not a real life robot. That would be significantly more short-lived as the first tumble off a cliff would be fairly final.

Problem 7

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

Double Q-Learning is a modification of the original Q-learning approach aiming to fix an overestimation bias. The overestimation bias is when the algorithm overestimates the true value of an action as it doesn't have enough estimation to assess the action's true value. It does this by utilizing two sets of Q-values (hence its name), updating the values as follows:

$$Q_1(S_t, A_t) = Q_1(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_2\big(S_{t+1}, \max Q_1(S_{t+1}, a)\big) - Q_1(S_t, A_t)\Big] \tag{12}$$

$$Q_2(S_t, A_t) = Q_2(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q_1\big(S_{t+1}, \max Q_2(S_{t+1}, a)\big) - Q_2(S_t, A_t)\Big] \tag{13}$$

When we update these values, one set of Q-values is used to select the next action and the other is used to estimate its value. Since this reduces the overestimation bias in the estimate of the action, performance of the function improves overall.