Exercise Set 3

6.0 VU AKNUM Reinforcement Learning

June 11, 2020

Exercise (6.1) If V changes during the episode, the TD update equation (6.2) will change to:

$$V_{t+1}(S_t) \leftarrow V_t(S_t) + \alpha[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)] \tag{1}$$

In the same manner, the TD error (6.5) changes to:

$$\delta_t \doteq R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t) \tag{2}$$

Thus, the MC/TD identity can be adapted to changing V using Equation 1 and Equation (6.6):

$G_t - V_t(S_t) = R_{t+1} + \gamma G_{t+1} - V_t(S_t)$
$= R_{t+1} + \gamma G_{t+1} - V_t(S_t) + \gamma V_t(S_{t+1}) - \gamma V_t(S_{t+1})$
$= \delta_t + \gamma(G_{t+1} - V_t(S_{t+1}))$
$= \delta_t + \gamma(G_{t+1} - V_{t+1}(S_{t+1}) + e_{t+1})$
$= \delta_t + \gamma(\delta_{t+1} + e_{t+1}) + \gamma^2(G_{t+2} - V_{t+2}(S_{t+2}) + e_{t+2})$
$= \sum_{k=t}^{T-t} \gamma^{k-t}\delta_k + \gamma^{k-t+1}e_{k+1}$

Per Equation 1, the difference between V_t and V_{t+1} is represented by $e_{t+1} = \alpha[R_{t+1} + \gamma V_t(S_{t+1}) - V_t(S_t)]$.

Exercise (6.8) The Sarsa update equation is defined as action value version of the TD update equation (6.2):

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{3}$$

Thus, it is easy to adapt the MC/TD identity (6.6) to Sarsa:

$G - Q(S_t, A_t) = R_{t+1} + \gamma G_{t+1} - Q(S_t, A_t)$
$= R_{t+1} + \gamma G_{t+1} - Q(S_t, A_t) + \gamma Q(S_{t+1}, A_{t+1}) - \gamma Q(S_{t+1}, A_{t+1})$
$= \delta_t + \gamma(G_{t+1} - Q(S_{t+1}, A_{t+1}))$
$= \delta_t + \gamma\delta_{t+1} + \gamma^2(G_{t+2} - Q_{t+2}(S_{t+2}, A_{t+2}))$
$= \sum_{k=t}^{T-t} \gamma^{k-t}\delta_k$

The minor difference being that G_t now depends on states-action pairs instead of states alone.

Exercise (6.11) Whereas Sarsa uses A_{t+1} according to the policy under consideration π, Q-learning chooses $A_{t+1} = \arg\max_a Q(S_{t+1}, a))$, i.e. it chooses greedily the action that maximizes the existing action value estimates. Therefore, the policy being followed is different from the policy being optimized, which is the defining property of off-policy methods.

Exercise (6.12) The update equation for Sarsa is given in Equation 3. The Q-learning update equation is:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4}$$

In general, when actions are chosen greedily, Sarsa and Q-learning will be similar since choosing A_{t+1} according to a policy greedy to Q is the same as choosing $A_{t+1} = \arg\max_a Q(S_{t+1}, a))$.

But there is a minor difference, in that Sarsa chooses A_{t+1} prior to updating Q and execute exactly this action in the next step. In contrast, Q-learning only retrieves A_{t+1} in order to update $Q(S_t, A_t)$, but does not execute it. It will in S_{t+1} choose an action according to the *updated* Q. This means if $S_t = S_{t+1}$, A_{t+1} according to Q_t might be different than A_{t+1} according to Q_{t+1} and hence Sarsa and Q-learning might be different in this case.

Exercise (6.13) The update equation for Exected Sarsa is given in Equation 5 and Equation 6.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbb{E}_\pi[Q(S_{t+1}, A_{t+1}) \mid S_{t+1}] - Q(S_t, A_t)] \tag{5}$$

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{6}$$

Now, for Double Expected Sarsa, the action value function used to determine the best action needs to be different from the action value function used to estimate the action value. Denoting them by Q_2 and Q_1 resp leads to the following update equation:

$$Q_1(S_t, A_t) = Q_1(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q_2(S_{t+1}, a) - Q_1(S_t, A_t)] \tag{7}$$

As in Double Q-learning, the behavior policy π used to compute the expectation could be an ϵ-greedy policy based on the average of Q_1 and Q_2.

Exercise (7.1) The n-step TD update adapted from Equation 1 is given by:

$$V_{t+n}(S_t) = V_{t+n-1}(S_t) + \alpha[G_{t:t+n} - V_{t+n-1}(S_t)] \tag{8}$$

The n-step return is given by:

$$G_{t:t+n} = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{9}$$

When trying to derive the n-step TD/MC identity, one can make use of the assumption that V does not change, hence $V_{t+n-1} = V$.

$G_{t:t+n} - V_{t+n-1}(S_t) = G_{t:t+n} - V(S_t)$

$= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t)$

Now, again with the excessive addition trick one can introduce the TD error as defined by Equation 2:

$= R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - V(S_t) + \gamma V(S_{t+1}) - \gamma V(S_{t+1})$

$= \delta_t + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - \gamma V(S_{t+1})$

$= \delta_t + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - \gamma V(S_{t+1}) + \gamma^2 V(S_{t+2}) - \gamma^2 V(S_{t+2}) + \gamma V(S_{t+1}) - \gamma V(S_{t+1})$

$= \delta_t + \gamma \delta_{t+1} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - \gamma V(S_{t+1}) - \gamma^2 V(S_{t+2}) + \gamma V(S_{t+1})$

$= \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) - \gamma V(S_{t+1}) - \gamma^2 V(S_{t+2}) - \gamma^3 V(S_{t+3}) + \gamma V(S_{t+1}) + \gamma^2 V(S_{t+2})$

After replacing all $n-1$ reward terms, one ends up with a nice sum of these terms, while the last term (the value estimate $\gamma^n V(S_{t+n})$ remains. Also, the "excess" terms introduced by this also form a nice sum.

$= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k + \gamma^n V(S_{t+n}) - \sum_{k=t+1}^{t+n} \gamma^{k-t} V(S_k) + \sum_{k=t+1}^{t+n-1} \gamma^{k-t} V(S_k)$

The two excess sums which have to be eliminated are the same except the second sum terminates one step earlier. Conveniently, there is still the remaining value estimate term of $G_{t:t+n}$ which can simply be added to the shorter sum. One ends up with:

$= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k - \sum_{k=t+1}^{t+n} \gamma^{k-t} V(S_k) + \sum_{k=t+1}^{t+n} \gamma^{k-t} V(S_k)$

$= \sum_{k=t}^{t+n-1} \gamma^{k-t} \delta_k$

Exercise (8.1) The problem at hand is a grid with around 50 states but also some obstacles. The agent needs to reach the goal from a fixed starting position. The transitions are deterministic but initially unknown. The reward is 1 when the goal is reached. Important: the rewards are discounted since $\gamma = 0.95$.

The nonplanning method is implemented as a 1-step method. In order to adapt it to a multi-step method, it could be rewritten into n-step Sarsa. This is done in the implementation task in `dynamaze.ipynb`. The comparison there shows that the nonplanning multi-step method is better than the nonplanning 1-step method, but it is still outperformed by the planning 1-step methods.

As for why this might be the case: The main problem for the nonplanning 1-step method is that only a single state is updated (c. Figure 8.3). This is due to the sparsity of the reward (initially only one state has a non-zero reward). The multi-step method improves upon that because it takes the last n states into account and updates them accordingly. But the planning method maintains a model across episodes, which enables it in theory to update all already visited states, while n-step method can only update states visited in a given episode. This is why the planning method is able to propagate relevant changes faster to more states. Another advantage of the planning method is that the planning computation can be executed concurrently to the direct RL steps.

An important caveat is that this is obviously contingent on the specifics of this task, among them a fixed environment, deterministic transitions and a policy that breaks ties randomly.