Exercise Set 4

6.0 VU AKNUM Reinforcement Learning

June 29, 2020

Exercise (Autonomous Orchard) The task is to approximate the optimal action value function for an orchard task. The training data is given below:

S	G
$(4, 7, 1)$	3
$(10, 6, 0)$	-15
$(20, 1, 15)$	5
$(4, 19, 3)$	21

The function can be approximated $\hat{q}(s, \mathbf{w}) \approx Q(s, \text{harvest})$ using an MC goal and a learning rate $\alpha = 0.01$. First, initialize $\mathbf{w} = (1, 1, 1)^T$.

t=0, example 1: Calculate $\hat{q}(S_0, \text{harvest}, \mathbf{w}) = 4 * 1 + 7 * 1 + 1 * 1 = S_0 * \mathbf{w} = 12$. The real value is $Q(S_0, \text{harvest}) = G_1 = 3$. The gradient is $\nabla\hat{q}(S_0, \text{harvest}, \mathbf{w}) = \left(\frac{\partial\hat{q}}{\partial w_1}, \frac{\partial\hat{q}}{\partial w_1}, \frac{\partial\hat{q}}{\partial w_1}\right)^T = (4, 7, 1)^T$. The update to $\mathbf{w}$ is $\alpha[Q(S_0, \text{harvest}) - \hat{q}(S_0, \text{harvest}, \mathbf{w})]\nabla\hat{q}(S_0, \text{harvest}, \mathbf{w}) = 0.01 * [3 - 12] * (4, 1, 7)^T = -0.09 * (4, 1, 7)^T = (-0.36, -0.09, -0.63)^T$. Thus, $\mathbf{w} = \mathbf{w}_1 = \mathbf{w}_0 + (-0.36, -0.09, -0.63)^T = (0.64, 0.91, 0.37)$.

t=1, example 2: $\hat{q}(S_1, \text{harvest}, \mathbf{w}) = 11.86$ $Q(S_1, \text{harvest}) = G_2 = -15$.
$\alpha[Q(S_1, \text{harvest}) - \hat{q}(S_1, \text{harvest}, \mathbf{w})]\nabla\hat{q}(S_1, \text{harvest}, \mathbf{w}) = (-2.686, -1.611, 0)^T$.
$\mathbf{w} = \mathbf{w}_2 = \mathbf{w}_1 + (-2.686, -1.611, 0)^T = (-2.046, -0.701, 0.37)$

t=2, example 3: $\hat{q}(S_2, \text{harvest}, \mathbf{w}) = -36.071$ $Q(S_2, \text{harvest}) = G_3 = 5$.
$\alpha[Q(S_2, \text{harvest}) - \hat{q}(S_2, \text{harvest}, \mathbf{w})]\nabla\hat{q}(S_2, \text{harvest}, \mathbf{w}) = (8.214, 0.411, 6.161)^T$.
$\mathbf{w} = \mathbf{w}_3 = \mathbf{w}_2 + (8.214, 0.411, 6.161)^T = (6.168, -0.29, 6.531)$

t=3, example 4: $\hat{q}(S_3, \text{harvest}, \mathbf{w}) = 38.755$ $Q(S_3, \text{harvest}) = G_4 = 21$.
$\alpha[Q(S_3, \text{harvest}) - \hat{q}(S_3, \text{harvest}, \mathbf{w})]\nabla\hat{q}(S_3, \text{harvest}, \mathbf{w}) = (-0.71, -3.373, -0.533)^T$.
$\mathbf{w} = \mathbf{w}_4 = \mathbf{w}_3 + (-0.71, -3.373, -0.533)^T = (5.458, -3.663, 5.998)$

t=4, example 1: $\hat{q}(S_0, \text{harvest}, \mathbf{w}) = 2.189$ $Q(S_4, \text{harvest}) = G_5 = 3$.
$\alpha[Q(S_0, \text{harvest}) - \hat{q}(S_0, \text{harvest}, \mathbf{w})]\nabla\hat{q}(S_0, \text{harvest}, \mathbf{w}) = (0.032, 0.057, 0.008)^T$.
$\mathbf{w} = \mathbf{w}_5 = \mathbf{w}_4 + (0.032, 0.057, 0.008)^T = (5.49, -3.606, 6.006)$

$\vdots$

Interestingly, after just one epoch the estimation was very near to the optimal value for S_0, but this is likely just intermediary luck. It will likely take multiple epochs until $\hat{q}$ converges.

Exercise (9.1) Tabular methods can be represented as linear methods. First, the state s needs to be converted into a numeric representation $\mathbf{x}(s)$, i.e. embedded in a vector space. For this, a range of possible encodings have been developed in machine learning research. Since for normal tabular methods the state is discrete, ehe easiest and most fitting for the task would be one-hot encoding. Each feature is converted into a vector of feature length where only the index of the specific feature value is 1 and the others are zero. For example, given $\mathcal{S} = \{left, right, up, down\}$, the specific state $S_t = up$ would be encoded as $\mathbf{x}(S_t)(0, 0, 1, 0)$. This works also for states consisting of multiple discrete values by simply conatenating the encodings.

In linear methods, the estimate is calculated using below equation.

$$\hat{v}(s, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s) \tag{1}$$

Now, using the one-hot encoding, something interesting happens. Let's consider the above example for $S_t = up$. Then $\hat{v}(up, \mathbf{w}) = \mathbf{w}^T \mathbf{x}(up) = w_1 * 0 + w_2 * 0 + w_3 * 1 + w_4 * 0 = w_3$. Meaning that the inner product of $\mathbf{w}$ with the one-hot encoding in effect *selects* the element w_i of the index i which correspons to the current state. Hence, w_i corresponds to the state element of the value table $V(i)$.

The general SGD update is given below.

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha[U_t - \hat{v}(S_t, \mathbf{w}_t)]\mathbf{x}(S_t) \tag{2}$$

U_t is the target, i.e. G_t or $R_{t+1} + \hat{v}(\mathbf{x}, \mathbf{w})$ depending on the tabular algorithm. As established above, $\hat{v}(\mathbf{x}, \mathbf{w})$ can be replaced with $V(S_t)$. Thus, the inner operation is just like the errors observed in tabular methods (e.g. TD error). Next, this error is multiplied with $\mathbf{x}(S_t)$, which will, in effect, convert the error into a vector which is zero everywhere except at the index of the current state. Now, this vector is added to $\mathbf{w}$, which is V as established above (meaning it holds all values as one-dimensional array). Note that the only element which will change is the element corresponding to the state value, as all other values are 0 in $\mathbf{x}$. Hence, the above equation can be replaced with:

$$V_{t+1} = V_t + \alpha[G_{t:T} - V(S_t)]\mathbf{x}(S_t) = V_{t+1}(S_t) = V_t(S_t) + \alpha[G_{t:T} - V_t(S_t)] \tag{3}$$

The above equation is just the good old tabular update.

Exercise (10.4) A differential version of semi-gradient Q-learning can be constructed by adapting the Sarsa equivalent. 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. The adaptation can be seen in Algorithm 1.

Algorithm 1 Differential version of semi-gradient Q-learning

Input: $\hat{q} : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, $\alpha, \beta > 0$

Initialize $\mathbf{w} \in \mathbb{R}^d$ arbitrarily.
Initialize $\bar{R}$ arbitrarily
Initialize S
while not converged **do**
  choose A as a function of $\hat{q}(S, ., \mathbf{w})$ (e.g. ϵ-greedy)
  Take action A, observe R, S'
  $\delta \leftarrow R - \bar{R} + \max_a \hat{q}(S', a, \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
  $\bar{R} \leftarrow \bar{R} + \beta\gamma$
  $\mathbf{w} \leftarrow \mathbf{w} + \alpha\delta\nabla\hat{q}(S, A, \mathbf{w})$
  $S \leftarrow S'$
end while

Exercise (11.1) The update equation n-step off-policy TD is given below.

$$V_{t+n}(S_t) \doteq V_{t+n-1}(S_t) + \alpha\rho_{t:t+n-1}[G_{t:t+n} - V_{t+n-1}(S_t)] \tag{4}$$

where $\rho_{t:t+n-1}$ is the importance sampling ratio $\rho_{t:h} \doteq \prod_{k=t}^{h} \frac{\pi(A_k|S_k)}{b(A_k|S_k)}$, which stays the same in the semi-gradient case. $G_{t:t+n}$ is the n-step return defined by below equation.

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n V_{t+n-1}(S_{t+n}) \tag{5}$$

This can be adapted to the approximative method by simply replacing the table V with the approximation $\hat{v}$. It uses simply the current weights.

$$G_{t:t+n} \doteq R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1}R_{t+n} + \gamma^n \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}) \tag{6}$$

Of course, the above case only holds for episodic tasks. For continuing tasks, the return needs to be undiscounted and differential. Thus, using the average reward $\bar{R}$, the equation changes to:

$$G_{t:t+n} \doteq R_{t+1} - \bar{R}_t + R_{t+2} - \bar{R}_{t+1} + \ldots R_{t+n} - \bar{R}_{t+n-1} + \hat{v}(S_{t+n}, \mathbf{w}_{t+n-1}) \tag{7}$$

Now, Equation 4 can be converted into a semi-gradient method by updating the parameters $\mathbf{w}$ (the free part of $\hat{v}$) instead of the table V.

$$\mathbf{w}_{t+n} \doteq \mathbf{w}_{t+n-1} + \alpha\rho_{t:t+n-1}[G_{t:t+n} - \hat{v}(S_t, \mathbf{w}_{t+n-1})]\nabla\hat{v}(S_t, \mathbf{w}_{t+n-1}) \tag{8}$$

Exercise (12.1) The return G_t can be defined recursively by $G_t = R_{t+1} + \gamma G_{t+1}$. Accordingly, the n-step return given by Equation 6 can be defined recursively by

$$G_{t:t+k} \doteq R_{t+1} + \gamma G_{t+1:t+k+1} \tag{9}$$

The λ-return is defined by the below equation.

$$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n} \tag{10}$$

We start now with converting this turn into a recurrence equation. In order to bring about the next step in a recurrence relation like in (3.9), the n-step returns G need to start with $t+1$. We can apply Equation 9 for that:

$G_t^\lambda \doteq (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_{t:t+n}$
$= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}(R_{t+1}+\gamma G_{t+1:t+n})$
$= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}R_{t+1} + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma G_{t+1:t+n}$
since R_{t+1} is constant, $\lim_{n\to\infty}\lambda^n R_{t+1} \to R_{t+1}/(1-\lambda)$.
$= (1-\lambda)R_{t+1}/(1-\lambda) + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma G_{t+1:t+n}$
$= R_{t+1} + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma G_{t+1:t+n}$
$= R_{t+1} + (1-\lambda)\gamma\sum_{n=1}^{\infty}\lambda^{n-1}G_{t+1:t+n}$

Now, we would only need to set $n=2$ in the sum to be able to replace the whole right hand term with Equation 10. Let's look closer at the term we need to extract $G_{t+1:t+1}$. This special case can be derived by setting $n=0$ in Equation 6. In general, it is simply reduced to $G_{t:t+0} = \sum_{k=1}^{0} R_{t+k} + \gamma^0\hat{v}(S_{t+0}, \mathbf{w}_{t+0-1}) = \hat{v}(S_t, \mathbf{w}_{t-1})$. We have:

$= R_{t+1} + (1-\lambda)\gamma\hat{v}(S_{t+1}, \mathbf{w}_t) + \gamma(1-\lambda)\sum_{n=2}^{\infty}\lambda^{n-1}G_{t+1:t+n}$
$= R_{t+1} + (1-\lambda)\gamma\hat{v}(S_{t+1}, \mathbf{w}_t) + \gamma G_{t+1}^\lambda$

The above result is the recurrence definition of the lambda return G_t^λ.