Reinforcement Learning	May 1, 2019
Recitation 8	
Lecturer:Yishay Mansour	*TA: Lee Cohen*

Value Function Approximation

Definition 1. We would like to reduce our reinforcement learning problem to a supervised learning problem. To do so we need:

1. Loss function: we will use **Mean Squared Error(MSE)** by defining
$$J(w) = \sum_s \mu(s)(V^\pi(s) - \hat{V}^\pi(s;w))^2$$
, where $\mu(s)$ is the steady state distribution of π.
2. Update algorithm - in this case we will **Stochastic Gradient Descent(SGD)**.
3. Build samples for training by replacing $V^\pi(s_t)$ with an estimated value (Monte-Carlo, TD(0), TD(λ)).

Once we implement the conditions above, we can approximate the value function. Depending on the approximation used, the gradients updates accordingly:

1. **Monte-Carlo**
$$\Delta w_t = \alpha(R_t(s) - \hat{V}^\pi(s;w_t))\nabla_w \hat{V}^\pi(s;w)$$
$$R_T(s_j) = \sum_{t=j}^{T} r_t$$
2. **TD(0)**
$$\Delta w(s) = \alpha[r_t + \gamma \hat{V}(s_{t+1};w) - \hat{V}(s_t)]\nabla_w \hat{V}(s_t;w)$$
Note: This is a **semi-gradient**, since w influences the target $R_t(s_t) = r_t + \gamma \hat{V}(s_{t+1};w)$.
3. **TD(λ)** - forward view
$$\Delta w = \alpha[R_t^\lambda - \hat{V}(s_t;w)]\nabla_w \hat{V}(s_t;w)$$

4. **TD(λ)** - backward view

$$\begin{aligned} e_t &= \gamma\lambda e_{t-1} + \nabla_w \hat{V}(s_t; w) \\ \Delta_t &= r_t + \gamma\hat{V}(s_{t+1}; w) - \hat{V}(s_t; w) \\ \Delta w &= \alpha\Delta_t e_t \end{aligned}$$

In the case of linear function approximation the formulas are reduced to:

1. **Monte-Carlo**

$$\Delta w = \alpha[R_t - \hat{V}(s_t; w_t)]x(s_t)$$

2. **TD(0)**

$$\Delta w = \alpha[r_t + \gamma\hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t)]x(s_t)$$

3. **TD(λ)** - forward view

$$\Delta w = \alpha[R_t^\lambda - \hat{V}(s_t; w)]x(s_t)$$

4. **TD(λ)** - backward view

$$\begin{aligned} e_t &= \gamma\lambda e_{t-1} + x(s_t) \\ \Delta_t &= r_t + \gamma\hat{V}(s_{t+1}; w_t) - \hat{V}(s_t; w_t) \\ \Delta w &= \alpha\Delta_t e_t \end{aligned}$$

Autonomous Orchard

Exercise 1. An AI controlled orchard needs to decide when to harvest its trees. To do this it measures the concentration of three chemicals in the air. Each day the orchard can choose to wait or harvest. Waiting costs one credit in operating costs while a harvest ends the process. Once a crop is harvested, packaged and sold, the orchard is told the profit or loss of that harvest. Most experts agree that the function mapping the chemical concentrations to the profit is linear with some error.

The orchard has several samples of the profits from other harvests. Begin to approximate

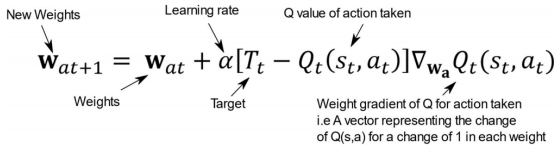
A (ppm)	B (ppm)	C (ppm)	Profit/Reward
4	7	1	3
10	6	0	-15
20	1	15	5
4	19	3	21

the function that maps the state feature vector to Q(state, harvest) using an MC goal. Do a gradient descent step on each sample. A sensible learning rate would be around 0.01, but feel free to try any value.

Solution 1. The real equation used for the harvest reward was:

$$r_{ht} = -2c_{at} + c_{bt} + 3c_{ct} \pm 1 \tag{1}$$

First set up arbitrary weights for harvesting $w_{h0} = (3 \quad 2 \quad 1)$.
We also choose a learning rate $\alpha = 0.01$. Let the feature vector representing state s_t be c_t which is a vector of concentrations of chemicals A, B, and C.



To calculate Q from the feature vector of state s_t: $Q(s_t, harvest_t) = c_t w_{ht}$
For MC: $T_t = R_t$

We calculate the $Q(s_t, harvest_t)$ values for the first sample:

$$Q(s_t, harvest_t) = \begin{pmatrix} c_{at} & c_{bt} & c_{ct} \end{pmatrix} \begin{pmatrix} w_{ha} \\ w_{hb} \\ w_{hc} \end{pmatrix}$$

$$Q(s_0, harvest_0) = c_{at}w_{ha} + c_{bt}w_{hb} + c_{ct}w_{hc} = 4 * 3 + 7 * 2 + 1 * 1 = 27$$

We now calculate the gradient

$$\nabla Q_t = \begin{pmatrix} \frac{dQ_t}{dw_{hat}} & \frac{dQ_t}{dw_{hbt}} & \frac{dQ_t}{dw_{hct}} \end{pmatrix}$$
$$\nabla Q_0 = \begin{pmatrix} c_{a0} & c_{b0} & c_{c0} \end{pmatrix} = \begin{pmatrix} 4 & 7 & 1 \end{pmatrix}$$

Now we can calculate the new weights:

$$w_{h1} = \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} + 0.01[3 - 27] \begin{pmatrix} 4 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 2.04 & 0.32 & 0.76 \end{pmatrix}$$

Repeating for sample 2:

$$Q(s_1, harvest_1) = 10 * 2.04 + 6 * 0.32 + 0 * 0.76 = 22.32$$
$$w_{h2} = \begin{pmatrix} 2.04 & 0.32 & 0.76 \end{pmatrix} + 0.01[-15 - 22.32] \begin{pmatrix} 10 & 6 & 0 \end{pmatrix} = \begin{pmatrix} -1.69 & -0.72 & 0.76 \end{pmatrix}$$

Repeating for sample 3:

$$Q(s_2, harvest_2) = 20 * -1.69 + 1 * -0.72 + 15 * 0.76 = -23.12$$
$$w_{h3} = \begin{pmatrix} -1.69 & -0.72 & 0.76 \end{pmatrix} + 0.01[5 + 23.12] \begin{pmatrix} 20 & 1 & 15 \end{pmatrix} = \begin{pmatrix} 3.93 & 0.44 & 4.98 \end{pmatrix}$$

Repeating for sample 4:

$$Q(s_3, harvest_3) = 4 * 3.93 + 19 * -0.44 + 3 * 4.98 = 22.3$$
$$w_{h4} = \begin{pmatrix} 3.93 & -0.44 & 4.98 \end{pmatrix} + 0.01[21 - 22.3] \begin{pmatrix} 4 & 19 & 3 \end{pmatrix} = \begin{pmatrix} 3.88 & -0.69 & 4.94 \end{pmatrix}$$

Exercise 2. The orchard also has a record from its own last harvest. Continue to approxi-

A (ppm)	B (ppm)	C (ppm)	Action	Profit/Reward
6	7	2	Wait	-1
3	8	4	Harvest	19

mate Q(state, harvest) and start approximating Q(state, wait). Run through this episode, performing gradient descent and using the TD(0) goal. Instead of choosing your actions in an ϵ-greedy way, assume the ϵ-greedy algorithm chose the actions in the table above.

Solution 2. For TD(0):

$$T_t = R_t + \gamma Q(s_{t+1}, a_{t+1})$$

Where a_{t+1} is chosen ϵ-greedily. For this question we assume it always picks the actions in the episode provided. We can set $\gamma = 1$ as there is an absorbing state but we could also select a lower value.

Need initial **wait** weight values: $w_{w4} = \begin{pmatrix} 3 & 2 & 1 \end{pmatrix}$

Day 1: Wait

$$Q(s_4, wait_4) = 6 * 3 + 7 * 2 + 2 * 1 = 34$$

We also need to calculate:

$$Q(s_5, harvest_5) = 3 * 3.88 + 8 * -0.69 + 4 * 4.94 = 25.88$$
$$w_{w5} = \begin{pmatrix} 3 & 2 & 1 \end{pmatrix} + 0.01[-1 + 25.88 - 34] \begin{pmatrix} 6 & 7 & 2 \end{pmatrix} = \begin{pmatrix} 2.45 & 1.36 & 0.82 \end{pmatrix}$$

Day 2: Harvest

$$Q(s_5, harvest_5) = 25.88$$

The state after harvest is an absorbing state so $Q(s_6, *) = 0$.
We also need to calculate:

$$w_{h6} = \begin{pmatrix} 3.88 & -0.69 & 4.94 \end{pmatrix} + 0.01[19 + 0 - 25.88] \begin{pmatrix} 3 & 8 & 4 \end{pmatrix} = \begin{pmatrix} 3.67 & 1.24 & 4.6648 \end{pmatrix}$$

Exercise 3. The orchard enters a new harvest season. Over the next three days the concentrations will be:

Day	A (ppm)	B (ppm)	C (ppm)
1	20	6	1
2	10	7	2
3	5	8	4

Run through an episode using these values. Choose your actions greedily and don't update Q. If you wait on the third day then the episode is over. If your algorithm harvested, did it harvest at its maximum possible Q(s, harvest) or would it have had a higher value on one of the other two days? If your algorithm waited on the third day, which day was it closest to harvesting?

Solution 3. We need to find Q values for each day and choose our actions greedily. Using the final weight vectors so far:

$$w_{w5} = \begin{pmatrix} 2.45 & 1.36 & 0.82 \end{pmatrix}$$
$$w_{h6} = \begin{pmatrix} 3.67 & 1.24 & 4.66 \end{pmatrix}$$

Day 1:

$$Q(s_6, wait_6) = 20 * 2.45 + 6 * 1.36 + 1 * 0.82 = 57.98$$
$$Q(s_6, harvest_6) = 20 * 3.67 + 6 * -1.24 + 1 * 4.66 = 70.62$$

So the episode immediately ends in a harvest. For the other days:
Day 2:

$$Q(s_7, harvest_7) = 10 * 3.67 + 7 * -1.24 + 2 * 4.66 = 37.34$$

Day 3:

$$Q(s_8, harvest_8) = 5 * 3.67 + 8 * -1.24 + 4 * 4.66 = 27.07$$

So the orchard harvested on the highest value of Q(s, harvest) that it would have had in those 3 days. If the orchard had waited on the third day then the closest day to harvesting would have been the one with the smallest value of

$$Q(s_t, wait_t) - Q(s_t, harvest_t)$$

Exercise 4.

1. What is the overall effect of increasing the learning rate? What happens when you set it too high? What happens when you set it too low?
2. Write a short RL problem where it would be better to use feature vectors to represent a state. Write one where it would be better to use discrete states.

Solution 4.

1. Increasing the learning rate overall increases the effect each sample has on determining the weights. If you set it too high, then the weights will never converge as each small error will create too large a change in the weights. If you set the learning rate too low, then the weights will take a long time to converge and will be vulnerable to getting stuck in local maxima.
2. For the inverted pendulum problem, a robot is trying to balance a rod on a hinge on top of it. As the rod can be at any angle the problem would be better modelled continuously. For a chess AI every possible state of the board can be a state in the model and so a discrete approach would be better.