Bellman equations (interrelationship optimally)

$$v_*(s) = \max_a Q_*(s,a)$$

$$Q_*(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_*(s'))$$

Optimal policy from Bellman equations

$$\pi_*(s) = \operatorname{argmax}_a Q_*(s,a)$$

Bellman equations (exercises)

Suppose there are four possible actions a_1 a_2, a_3 and a_4 from a given state s, and the following information is given:

- $Q^*(s,a_1) = 12$
- $Q^*(s,a_2) = 5$
- $Q^*(s,a_3) = -1$
- $Q^*(s,a_4) = 0$

a) Which will be the value of $V^*(s)$?

b) Suppose that ***s'*** is the state reached form s by taking the action a_2, and given that the transition function and reward are as follows: $T(s,a_2,s') = 1$; $R(s,a_2,s') = 4$; and that the discount factor is 0.5. Which will be the value of $V^*(s')$?

Value State iteration rule (*Dynamic Programming*)

How to calculate V*? By using the value iteration rule.

This rule sets that, if discounted factor is less than 1, or the MDP is finite, the process of calculating V* can be obtained by an iterative process

$$(V^*_0, V^*_1, \ldots, V^*_k, V^*_{k+1}, \ldots, V^*_n)$$

that is **guaranteed to converge**.

The core updating rule for this calculation is:

$$V^*_{k+1}(s) = \max_a [\sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V^*_k(s')]$$

Value State iteration rule (Algorithm)

Algorithm to calculate V*

Set V_0 arbitrarily (e.g. as 0 for all states)

k = 0

Do

  Difference = 0;

  For each s in S

    v = $V_k(s)$

    $V_{k+1}(s) = \max_a [\sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_k(s')]$

    Difference = max(Difference, |v - $V_{k+1}(s)$|

  k+=1

while Difference > small positive number

Value State iteration rule (example and exercises)

Given the following scenario:

Start,0	0	0	0	**Final, +1**
0	1	2	3	4

An agent want to go to the final cell from the start cell. **The only action available is to move right and it is a deterministic one**. The reward function is just related to the cell that the agent occupies, R(s) = 0 for s between 0 and 3, and R(s) = 1 in s = 4. **When the agent arrives at the state 4 it stays there for one time step more, receives a reward of +1 and stops moving anymore**. Use the discount factor 0.5.

Value State iteration rule (example by exercises)

Let the functions V^*_k for all states be written as: $[V_k^*(0) V_k^*(1) V_k^*(2) V_k^*(3) V_k^*(4)]$

Let starts with $V^*_0(i) = 0$ for all i: $V_0^* = [0\ 0\ 0\ 0\ 0]$

Then, using the value iteration let updates $V^*_1(i)$ for all i: (***to solve***)

And, using the value iteration update $V^*_2(i)$ for all i: (***to solve***)

Which will be V^*_3?: (***to solve***)

How many steps have to be run to reach convergence? (***to solve***)

Value State iteration rule (modified example with no deterministic behaviour)

At any grid location the agent can choose to stay where it is,or to move right or left. The action "***stay***" at its current location is successful with probability ½ and if the agent is at the leftmost of rightmost grid location, it ends in the available neighbour location with probability ½, but if the agent is at any inner grid location, it has a probability ¼ of ending on either neighboring locations.

Whatever other action ("**left**" or "**right**") in a inner grid location is successful with probability ⅓. and with probability ⅔ it remains in the same place. If the agent is in the leftmost grid location and the action is "**left**" or the agent is in the rightmost grid location and the action is "**right**", then it behaves as in the stay case.

Value State iteration rule (modified example with no deterministic behaviour)

```
T =  np.array([
```

e.g., ¼ is the probability to go from S_2 to S_3 when performing action 'Stay'

Left	Stay	Right	
[[1/2,1/2,0,0,0],	[1/2,1/2,0,0,0],	[2/3,1/3,0,0,0]],	S_0
[[1/3,2/3,0,0,0],	[1/4,1/2,1/4,0,0],	[0,2/3,1/3,0,0]],	S_1
[[0,1/3,2/3,0,0],	[0,1/4,1/2,1/4,0],	[0,0,2/3,1/3,0]],	S_2
[[0,0,1/3,2/3,0],	[0,0,1/4,1/2,1/4],	[0,0,0,2/3,1/3]],	S_3
[[0,0,0,1/3,2/3],	[0,0,0,1/2,1/2],	[0,0,0,1/2,1/2]]	S_4

```
])
```

Remember that, in this example, **R(s,a,s') = R(s)** Why? (*hint: revisit the scenario definition shown three slides before*)

Q-Value iteration rule

Let's recall from the Bellman equations:

$$v_*(s) = \max_a Q_*(s,a)$$

$$Q_*(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s\backslash) + \gamma V_*(s'))$$

Therefore:

$$Q_*(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s\backslash) + \gamma \max_{a'} Q_*(s',a'))$$

So, recall that Q^*_k (s,a) be the ***expected rewards*** from state *s* and action *a*, and then ***acting optimally*** after k steps. Which will be the updated rule for Q-value?

$$Q^*_{k+1}(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s\backslash) + \gamma \max_{a'} Q^*_k(s',a'))$$

Q-Value iteration rule (Algorithm)

Algorithm to calculate **Q*(s,a)**

Set $Q_0(s,a)$ arbitrarily (e.g. 0 for all pairs state,action)

k = 0

Do

  Difference = 0;

  For each s in S

    For each a in A

      q(s,a) = $Q_k(s,a)$

      $Q_{k+1}(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma \max_{a'} Q_k(s',a'))$

      Difference = max(Difference, $|q(s,a) - Q_{k+1}(s,a)|$

  k += 1

while Difference > small positive number