Department of Information Technology

Course Name: 6IT4-02:Machine Learning

Unit-V: Semi Supervised Learning

Reinforcement Learning

Reinforcement Learning

1. Reinforcement learning is all about learning from the environment through interactions.
2. Finding the optimal policy / optimal value functions is the key for solving reinforcement learning problems.
3. Dynamic programming methods are used to find optimal policy/optimal value functions using the bellman optimality equations.
4. Dynamic programming methods are model based methods, require the complete knowledge of environment. such as transition probabilities and rewards.

Reinforcement Learning

There are two types of tasks in RL

1. **Prediction** : This type of task predicts the expected total reward from any given state assuming the function **π(a|s)** is given.

   (in other words) **Policy π** is given, it calculates the **Value function V(π)** with or without the model.

   ex: **Policy evaluation**

Reinforcement Learning

There are two types of tasks in RL

2. **Control** : This type of task finds the policy **π(a|s)** that maximizes the expected total reward from any given state.

(in other words) **Some Policy π** is given , it finds the **Optimal policy π***.

ex: **Policy improvement**

Policy iteration is the combination of both to find the optimal policy. Just like in supervised learning , we have regression and classification tasks, in reinforcement learning, we have prediction and control tasks.

Reinforcement Learning

There are two types of policy learning

1. **On policy learning** : It learns on the job. which means it evaluates or improves the policy that is used to make the decisions. It directly learns a policy which gives you decisions about which action to take in some state.

2.Off policy learning : It evaluates one policy (target policy) while following another policy (behaviour policy) just like we learn to do something while observing others doing the same thing.

target policy may be deterministic (ex: greedy) while **behaviour policy** is stochastic.

Reinforcement Learning

In RL problems we have two different tasks in nature.

1.Episodic task : A task which can last a finite amount of time is called **Episodic task (an episode)**

Ex : Playing a game of chess (win or lose or draw)

we only get the reward at the end of the task or another option is to distribute the reward evenly across all actions taken in that episode.

Ex: you lost the queen (-10 points), you lost one of the rooks (-5 points) etc..

2. **Continuous task** : A task which never ends is called **Continuous task**

Ex: Trading in the crypto currency markets or learning Machine learning on internet. in this , rewards may be given with discounting with a discount factor $\lambda \in [0,1]$

Reinforcement Learning

Model Free Methods

In MDP , we are given all the components to solve a problem, but what if we are not given some of the components ???

what if we are not given the transition probabilities and rewards for a RL problem?? That means we are not given the dynamics of the environment.

can we solve the problem?? , if so How???

This type of learning is called model free learning

Reinforcement Learning

In Model-free , we just focus on figuring out the value functions directly from the interactions with the environment

How to figure out V for unknown MDP (assume we get the policy)??

*T*here are few approaches for solving these kind of problems

1.Monte Carlo approach

2.Temporal-Difference approach

Reinforcement Learning

Monte Carlo Policy Evaluation

The goal here, again, is to learn the value function vpi(s) from episodes of experience under a policy pi. Recall that the return is the total discounted reward:
S1, A1, R2, ….Sk ~ pi

Also recall that the value function is the expected return:

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

We know that we can estimate any expected value simply by adding up samples and dividing by the total number of samples:

$$\bar{V}_\pi(s) = \frac{1}{N} \sum_{i=1}^{N} G_{i,s}$$

i – Episode index
s – Index of state

Reinforcement Learning

First Visit Monte Carlo: Average returns only for first time s is visited in an episode.

Here's a step-by-step view of how the algorithm works:

1. Initialize the policy, state-value function
2. Start by generating an episode according to the current policy
    - 2.1 Keep track of the states encountered through that episode
3. Select a state in 2.1
    - 3.1 Add to a list the return received after first occurrence of this state
    - 3.2 Average over all returns
    - 3.3 Set the value of the state as that computed average
4. Repeat step 3
5. Repeat 2-4 until satisfied

Reinforcement Learning

Every visit Monte Carlo: Average returns for every time s is visited in an episode.

For this algorithm, we just change step #3.1 to 'Add to a list the return received after every occurrence of this state'.

Let's consider a simple example to further understand this concept. Suppose there's an environment where we have 2 states – A and B. Let's say we observed 2 sample episodes:

$$A + 3 \rightarrow A + 2 \rightarrow B - 4 \rightarrow A + 4 \rightarrow B - 3 \rightarrow \text{terminate}$$

$$B - 2 \rightarrow A + 3 \rightarrow B - 3 \rightarrow \text{terminate}$$

A+3 => A indicates a transition from state A to state A, with a reward +3. Let's find out the value function using both methods:

First visit	***Every visit***
V(A) = 1/2(2 + 0) = 1 V(B) = 1/2(-3 + -2) = -5/2	V(A) = 1/4(2 + -1 + 1 + 0) = 1/2 V(B) = 1/4(-3 + -3 + -2 + -3) = -11/4

Reinforcement Learning

Solution:

First Visit

Episode 1: start with state A:-> 3+2-4+4-3=2

Episode 2: start with state A:->3-3=0

V(A)=(2+0)/2= 1

Episode 1: start with state B:->-4+4-3=-3

Episode 2: start with state B:-> -2+3-3=-2

V(B)=(-3-2)/2= -5/2

Reinforcement Learning

Every Visit

Episode 1: First Occurrence of A->3+2-4+4-3=2
Second Occurrence of A->2-4+4-3=-1
Third Occurrence of A->4-3=1
Episode 2: First Occurrence of A->3-3=0
So V(A)=(2-1+1+0)//4= 1/2

Episode 1: First Occurrence of B->-4+4-3=-3
Second Occurrence of B-> -3
Episode 2: First Occurrence of B->-2+3-3=-2
Second Occurrence of B-> -3
So V(B)=(-3-3-2-3)//4= -11/4

Q-Learning

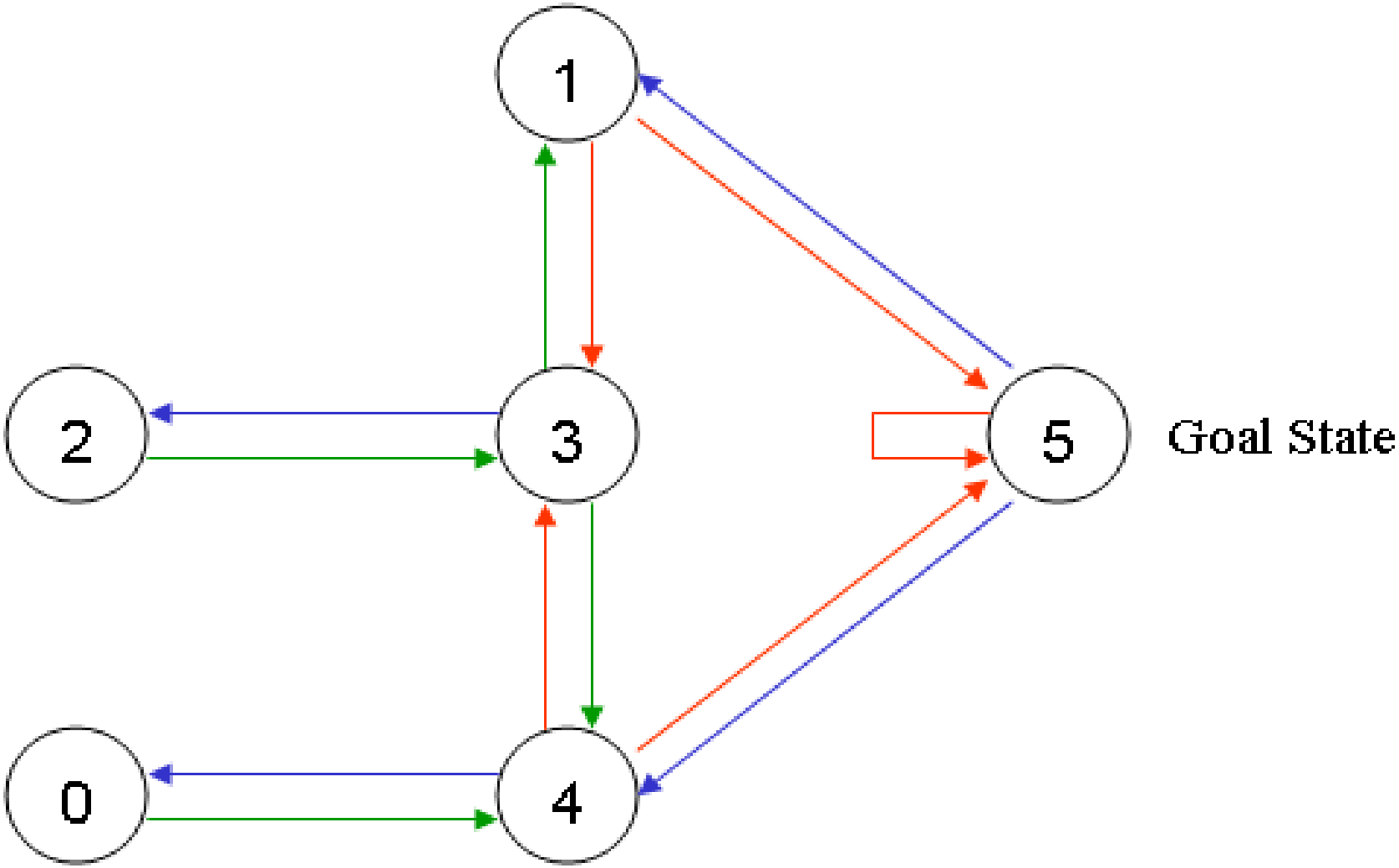
- Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state.
- It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed.
- More specifically, q-learning seeks to learn a policy that maximizes the total reward.

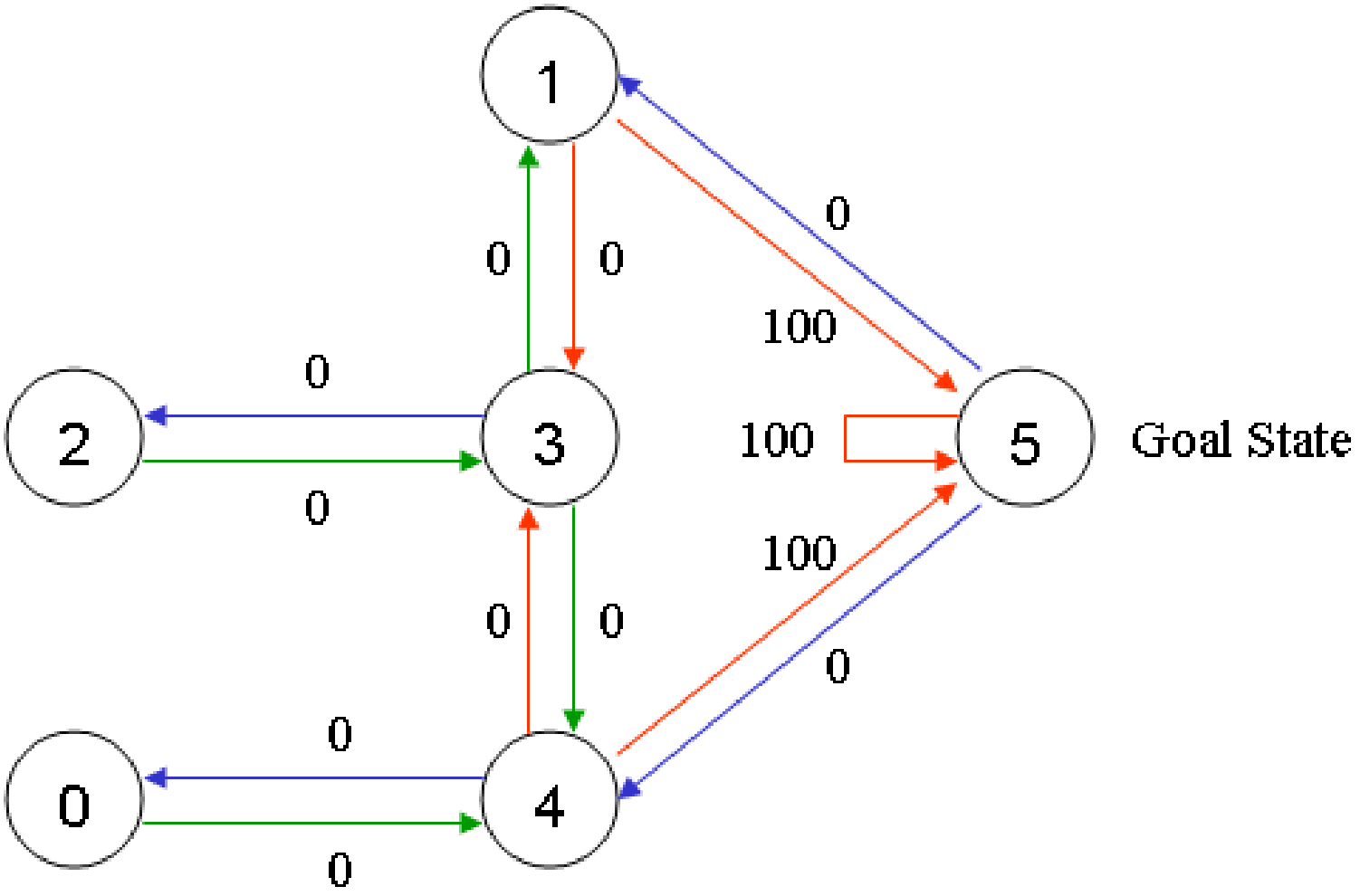
- **What's 'Q'?**
- The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

- ***Q*-learning** is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances.

- For any finite Markov decision process (FMDP), *Q*-learning finds an optimal policy in the sense maximizing the expected value of the total reward over any and all successive steps, starting from the current state.
- *Q*-learning can identify an optimal action-selection policy for any given FMDP, given infinite exploration time and a partly-random policy.
- "Q" names the function that returns the reward used to provide the reinforcement and can be said to stand for the "quality" of an action taken in a given state.

Example: Q-Learning

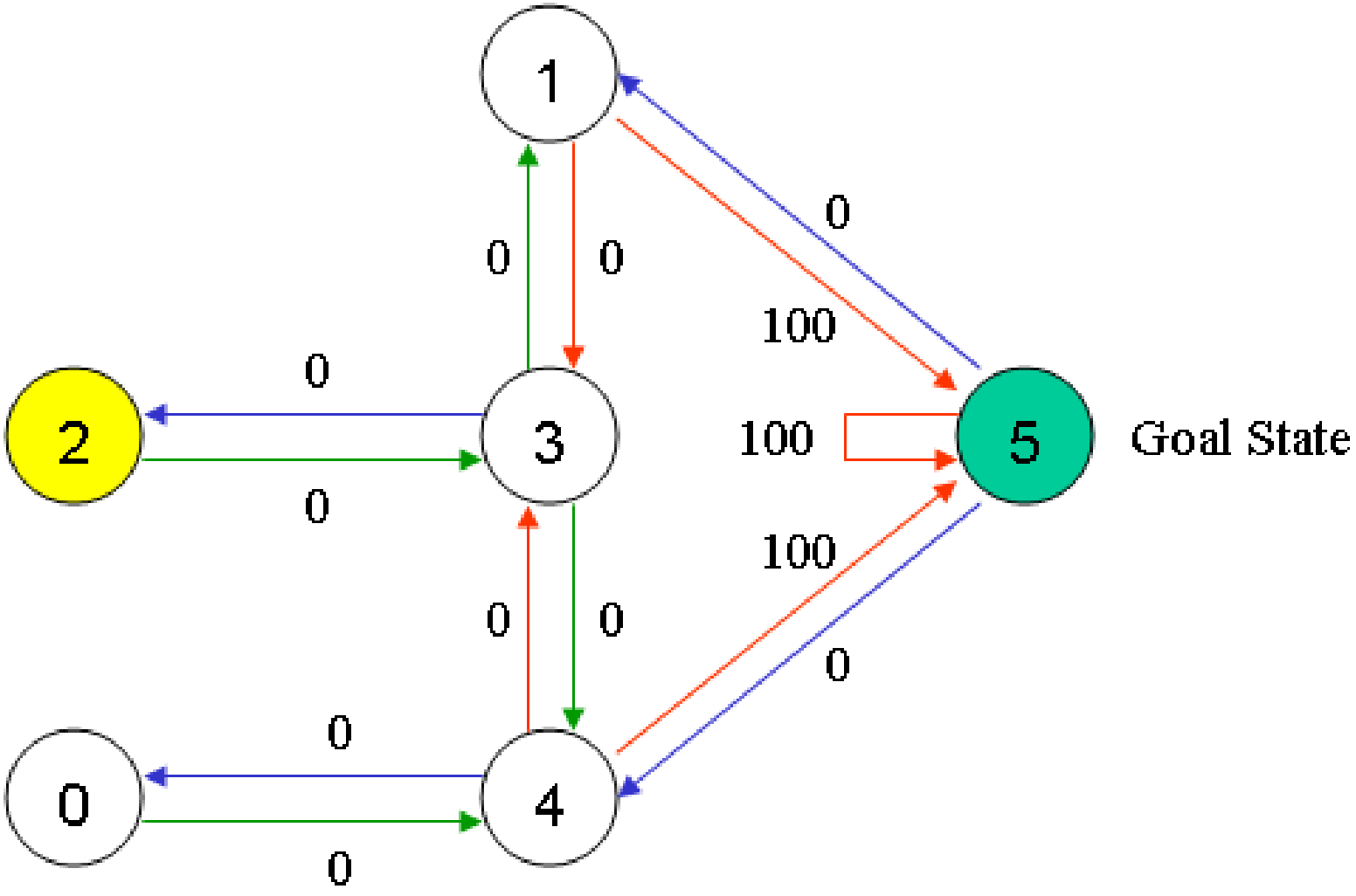



1
0
0
0
100
2
0
3
100
5
Goal State
0
100
0
0
0
4
0
0

- The terminology in Q-Learning includes the terms "state" and "action".
- We'll call each room, including outside,

----------→ "state",

- and the agent's movement from one room to another will be an---→ "action". In our diagram, a "state" is depicted as a node, while "action" is represented by the arrows.


1
2
3
4
5
0
Goal State
0
0
0
100
0
0
100
100
0
0
0
0

- Suppose the agent is in state 2. From state 2, it can go to state 3 because state 2 is connected to 3. From state 2, however, the agent cannot directly go to state 1 because there is no direct door connecting room 1 and 2 (thus, no arrows).
- From state 3, it can go either to state 1 or 4 or back to 2 (look at all the arrows about state 3). If the agent is in state 4, then the three possible actions are to go to state 0, 5 or 3. If the agent is in state 1, it can go either to state 5 or 3. From state 0, it can only go back to state 4.
- We can put the state diagram and the instant reward values into the following reward table, "matrix R".

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

The -1's in the table represent null values (i.e.; where there isn't a link between nodes). For example, State 0 cannot go to State 1.

- Now we'll add a similar matrix, "Q", to the brain of our agent, representing the memory of what the agent has learned through experience. The rows of matrix Q represent the current state of the agent, and the columns represent the possible actions leading to the next state (the links between the nodes).
- The agent starts out knowing nothing, the matrix Q is initialized to zero.

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

The transition rule of Q learning is a very simple formula:

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]

According to this formula, a value assigned to a specific element of matrix Q, is equal to the sum of the corresponding value in matrix R and the learning parameter Gamma, multiplied by the maximum value of Q for all possible actions in the next state.

- Episode:
- The agent will explore from state to state until it reaches the goal. We'll call each exploration an episode.
- Each episode consists of the agent moving from the initial state to the goal state. Each time the agent arrives at the goal state, the program goes to the next episode.

Q-Learning ALgorithm

The Q-Learning algorithm goes as follows:
1. Set the gamma parameter, and environment rewards in matrix R.
2. Initialize matrix Q to zero.
3. For each episode:
Select a random initial state.
Do While the goal state hasn't been reached.
•Select one among all possible actions for the current state.
•Using this possible action, consider going to the next state.
•Get maximum Q value for this next state based on all possible actions.
•Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
•Set the next state as the current state.
End Do
End For

- The algorithm above is used by the agent to learn from experience. Each episode is equivalent to one training session.
- In each training session, the agent explores the environment (represented by matrix R), receives the reward (if any) until it reaches the goal state.
- The purpose of the training is to enhance the 'brain' of our agent, represented by matrix Q. More training results in a more optimized matrix Q.

- The Gamma parameter has a range of 0 to 1 (0 <= Gamma > 1).
- If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.
- To use the matrix Q, the agent simply traces the sequence of states, from the initial state to goal state. The algorithm finds the actions with the highest reward values recorded in matrix Q for current state:

Algorithm to utilize the Q matrix:

1. Set current state = initial state.

2. From current state, find the action with the highest Q value.

3. Set current state = next state.

4. Repeat Steps 2 and 3 until current state = goal state.

The algorithm above will return the sequence of states from the initial state to the goal state.

- To understand how the Q-learning algorithm works, we'll go through a few episodes step by step. The rest of the steps are illustrated in the source code examples.
- We'll start by setting the value of the learning parameter Gamma = 0.8, and the initial state as Room 1.
- Initialize matrix Q as a zero matrix:

Look at the second row (state 1) of matrix R. There are two possible actions for the current state 1: go to state 3, or go to state 5. By random selection, we select to go to 5 as our action.

$$R = \begin{array}{c} \\ \text{State} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \text{Action} \\ \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix} \end{array}$$

Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
Q(1, 5) = R(1, 5) + 0.8 * Max[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0.8 * 0 = 100

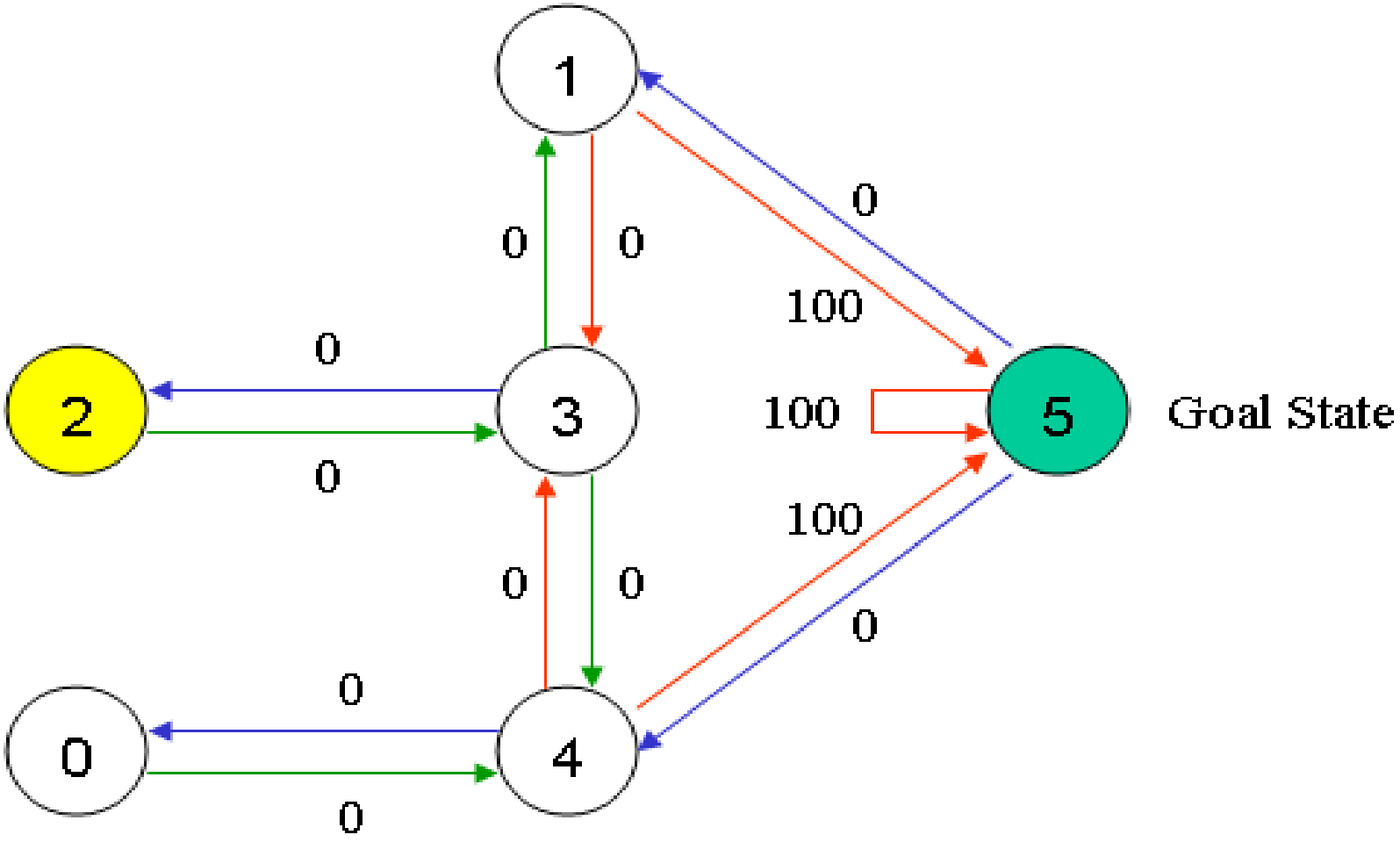
Since matrix Q is still initialized to zero, Q(5, 1), Q(5, 4), Q(5, 5), are all zero.
The result of this computation for Q(1, 5) is 100 because of the instant reward from R(5, 1).
The next state, 5, now becomes the current state. Because 5 is the goal state, we've finished one episode.

For the next episode, we start with a randomly chosen initial state. This time, we have state 3 as our initial state.
Look at the fourth row of matrix R; it has 3 possible actions: go to state 1, 2 or 4. By random selection, we select to go to state 1 as our action.
Then, we compute the Q value:
Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
Q(3,1) = R(3,1) + 0.8 * Max[Q(1, 3), Q(1, 5)] = 0 + 0.8 * Max(0, 100) = 80
We use the updated matrix Q from the last episode. Q(1, 3) = 0 and Q(1, 5) = 100. The result of the computation is Q(3, 1) = 80 because the reward is zero. The matrix Q becomes:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

- The next state, 1, now becomes the current state. We repeat the inner loop of the Q learning algorithm because state 1 is not the goal state.

- So, starting the new loop with the current state 1, there are two possible actions: go to state 3, or go to state 5. By lucky draw, our action selected is 5.


1
0
0
0
100
2
0
3
100
5
Goal State
0
0
100
0
0
0
0
4
0

Now, imaging we're in state 5, there are three possible actions: go to state 1, 4 or 5.
We compute the Q value using the maximum value of these possible actions.
Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
Q(1, 5) = R(1, 5) + 0.8 * Max[Q(5, 1), Q(5, 4), Q(5, 5)] = 100 + 0.8 * 0 = 100

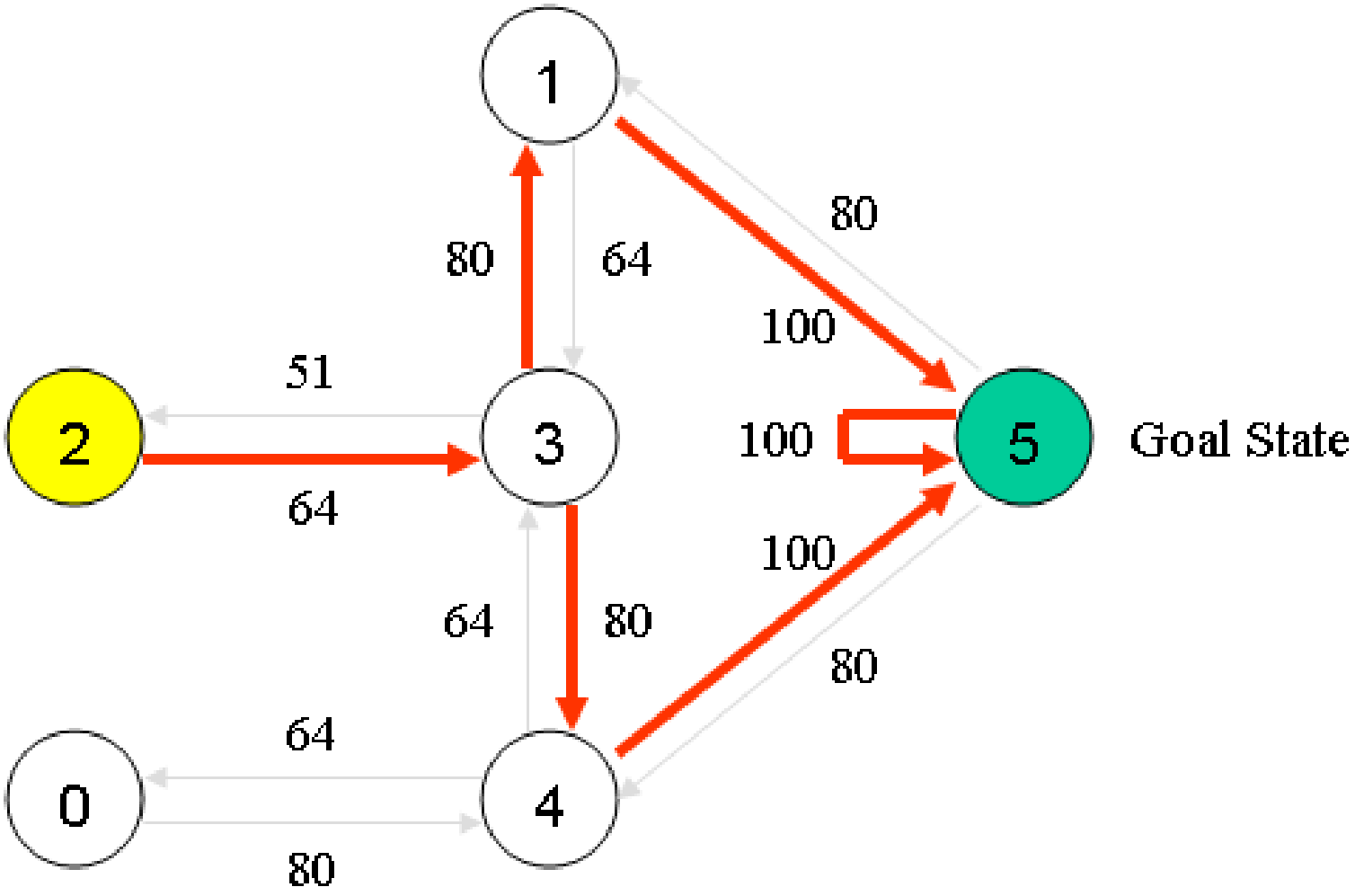
The updated entries of matrix Q, Q(5, 1), Q(5, 4), Q(5, 5), are all zero. The result of this computation for Q(1, 5) is 100 because of the instant reward from R(5, 1). This result does not change the Q matrix.
Because 5 is the goal state, we finish this episode. Our agent's brain now contain updated matrix Q as:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 80 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

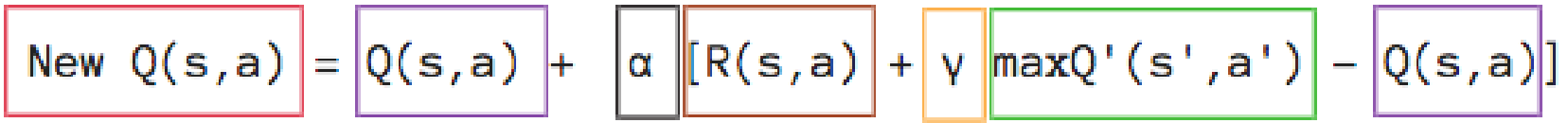
- If our agent learns more through further episodes, it will finally reach convergence values in matrix Q like:

$$Q = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 80 & 0 \\ 0 & 0 & 0 & 64 & 0 & 100 \\ 0 & 0 & 0 & 64 & 0 & 0 \\ 0 & 80 & 51 & 0 & 80 & 0 \\ 64 & 0 & 0 & 64 & 0 & 100 \\ 0 & 80 & 0 & 0 & 80 & 100 \end{bmatrix} \end{array}$$

- Once the matrix Q gets close enough to a state of convergence, we know our agent has learned the most optimal paths to the goal state. Tracing the best sequences of states is as simple as following the links with the highest values at each state.


1
80
64
80
100
51
2
3
100
5
Goal State
64
100
64
80
80
64
0
4
80

For example, from initial State 2, the agent can use the matrix Q as a guide:
From State 2 the maximum Q values suggests the action to go to state 3.
From State 3 the maximum Q values suggest two alternatives: go to state 1 or 4. Suppose we arbitrarily choose to go to 1.
From State 1 the maximum Q values suggests the action to go to state 5.
Thus the sequence is 2 - 3 - 1 - 5.



- New Q Value for that state and the action
- Learning Rate
- Reward for taking that action at that state
- Current Q Values
- Maximum expected future reward given the new state (s') and all possible actions at that new state.
- Discount Rate

TD-learning

JECRC Foundation

- **Temporal difference (TD) learning** is an approach to learning how to predict a quantity that depends on future values of a given signal.

TD-learning

JECRC Foundation

- The name TD derives from its use of changes, or differences, in predictions over successive time steps to drive the learning process.

TD-learning

- Temporal difference (TD) learning, is a model-free learning algorithm.

TD-learning

JECRC Foundation

It has two important properties:

1. It doesn't require the model dynamics to be known in advance.
2. It can be applied for non-episodic tasks as well.

TD-learning

JECRC Foundation

Bootstrapping:

- Temporal difference learning approximates the current estimate based on the previously learned estimate. This approach is called **bootstrapping.**

TD-learning

JECRC Foundation

- TD update rule for updating the value of a state:

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

value of a previous state = value of previous state + learning_rate * (reward + discount_factor(value of current state) — value of previous state)

TD Prediction Algorithm

- To update the values of all states using the TD update rule.

TD Prediction Algorithm

Step:

1. First, initialize V(S) to 0 or some arbitrary value.

TD Prediction Algorithm

Step:

2. Begin the episode. For every step in the episode, perform an action A in the state S and receive a reward R and move to the next state (s').

TD Prediction Algorithm

Step:

3. Update the value of the previous state using the TD update rule.

TD Prediction Algorithm

Step:

3. Update the value of the previous state using the TD update rule.

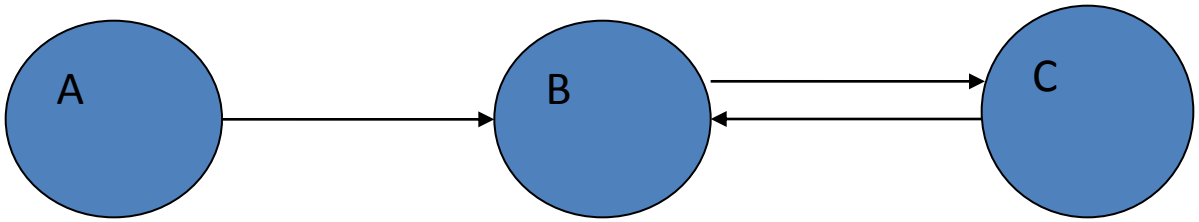
TD Prediction Algorithm

Step:

4. Repeat steps 2 and 3 until we reach the terminal state.

TD Prediction Algorithm: Example

State: A, B & C Action: Right, Left



Step-I

JECRC Foundation

Value Table

State	Value
A	0
B	0
C	0

Step-II

JECRC Foundation

- Say we are in a starting state (s)→ (A) and we take an action right and move to the next state (s')→ (B) and receive a reward (R) as 0.5.

- Update the value of state(A) using this information:

$$V(S_t) \leftarrow V(S_t) + \alpha\left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\right]$$

Step-II

JECRC Foundation

- Let us consider the learning rate (α) as 0.1
- and the discount factor (gamma) as 0.5;
- We know that the value of the state (A), as in V(s), is 0
- and the value of the next state (B), as in V(s’), is also 0.

Step-III

- Update the value of the previous state using the TD update rule.

Step-III

JECRC Foundation

- V(s) = V(A) = 0 + 0.1 [0.5 + 0.5 (0)-0]
- V(A) = 0.05

Step-III

JECRC Foundation

So, we update the value for the state (A) as 0.05 in the value table, as shown in the following Table:

State	Value
A	0.05
B	0
C	0

Step-II

JECRC Foundation

- Now that we are in the state (s) as (B), we take an action right and move to the next state (s') (C) and receive a reward (R) as -0.3
- Now find updated value of state (B)?

Step-III

- Update the value of the previous state using the TD update rule.

Step-lll

JECRC Foundation

We will substitute the values in the TD update equation:

So V(B) = 0 + 0.1 [-0.3 + 0.5(0)-0]

V(B) = -0.03

update the value of state (B) as -0.03 in the value table:

Step-III

JECRC Foundation

So, we update the value for the state (B) as -0.03 in the value table, as shown in the following Table:

State	Value
A	0.05
B	-0.03
C	0

Step-II

JECRC Foundation

- We are now in the state (s)→ (C).
- Let's take an action left.
- We again go back to that state (s')→ (B) and receive a reward (R) as -0.2.
- Here, the value of the state (C) is 0 and the value of the next state (B) is -0.03 in the value table.

Step-III

JECRC Foundation

Now find updated value of state (C) ?

So V(s) = V(C) = 0 +0.1 [-0.2 + 0.5 (-0.03)-0)]

V(C) = 0.1[-0.215]

V(C) = -0.0215

Update the value of state (C) as –0.0215 in the value table:

Step-III

JECRC Foundation

So, we update the value for the state (C) as -0.0215 in the value table, as shown in the following Table:

State	Value
A	0.05
B	-0.03
C	-0.0215

TD Prediction Algorithm

Step:

4. Repeat steps 2 and 3 until we reach the terminal state.

TD Prediction Algorithm

JECRC Foundation

THANK YOU