Reinforcement Learning

Dr. P. Kuppusamy
Prof / CSE

Reinforcement learning

- Reinforcement Learning is the agent must sense the environment, learns to behave (act) in a environment by performing actions (reinforcement) and seeing the results.
- Task
  - Learn how to behave successfully to achieve a goal while interacting with an external environment.
  - The goal of the agent is to **learn** an **action policy** that maximizes the total reward it will receive from any starting state.
- Examples
  – **Game playing:** player knows whether it win or lose, but not know how to move at each step

Applications

- A robot cleaning the room and recharging its battery
- Robot-soccer
- invest in shares
- Modeling the economy through rational agents
- Learning how to fly a helicopter
- Scheduling planes to their destinations

Reinforcement Learning Process

- RL contains two primary components:
  1. Agent (A) – RL algorithm that learns from trial and error
  2. Environment – World Space in which the agent moves (interact and take action)
- State (S) – Current situation returned by the environment
- Reward (R) – An immediate return from the environment to appraise the last action
- Policy (π) –Agent uses this approach to decide the next action based on the current state
- Value (V) – Expected long-term return with discount. Oppose to the short-term reward (R)
- Action-Value (Q) – Similar to value except it contains an additional parameter, the current action (A)

Figure shows RL is learning from interaction

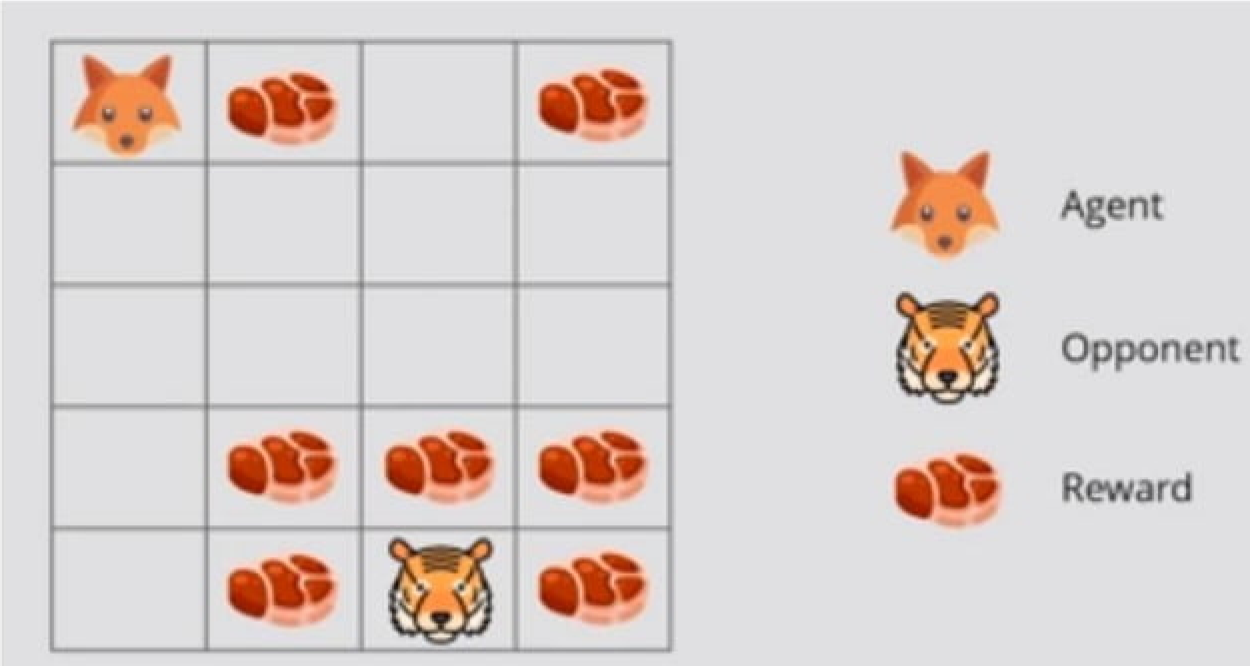
RL Approaches

- Two approaches
  - *Model based approach RL*:
    - learn the model, and use it to derive the optimal policy.

      e.g Adaptive dynamic learning(ADP) approach
  - *Model free approach RL*:
    - derive the optimal policy without learning the model.

      e.g LMS and Temporal difference approach
- Passive learning
  - The agent imply watches the world during transition and tries to learn the utilities in various states
- Active learning
  - The agent not simply watches, but also acts on the environment

Example



- Immediate reward is worth more than future reward.
- **Reward Maximization** - Agent is trained to take best (optimal) action to get maximum reward

Reward Maximization

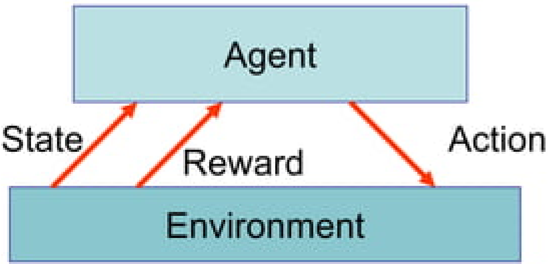
- **Exploration** – Search and capture more information about the environment
- **Exploitation** – Use the already known information to maximize the rewards

Reinforcement learning model

- Each percept(e) is enough to determine the State(the state is accessible)
- **Agent's task:** Find a optimal policy by mapping states of environment to actions of the agent, that maximize long-run measure of the reward (reinforcement)
- It can be modeled as Markov Decision Process (MDP) model.
  - Markov decision process (**MDP**) is a a mathematical framework for **modeling** decision making i.e mapping a solution in reinforcement learning.

MDP model

- MDP model <S,T,A,R>



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r1]{a1} s2 \xrightarrow[r2]{a2} s3$$

- S– set of states
- A– set of actions
- **Transition Function:** T(s,a,s') = P(s'|s,a) – the probability of transition from *s* to *s'* given action *a*

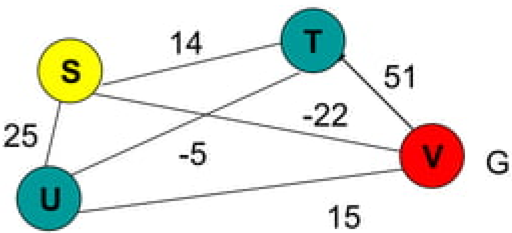
  T(s,a) → s'
- **Reward Function:** r(s,a) → r the expected reward for taking action *a* in state *s*

$$R(s,a) = \sum_{s'} P(s'|s,a) r(s,a,s')$$

$$R(s,a) = \sum_{s'} T(s,a,s') r(s,a,s')$$

MDP - Example I

- Consider the graph, and find the shortest path from a node S to a goal node G.
- Set of states {S, T, U, V}
- **Action** – Traversal from one state to another state
- **Reward** - Traversing an edge provides "length edge" in dollars.
- **Policy** – Path considered to reach the destination {S→T→V}



Q - Learning

- **Q-Learning** is a value-based **reinforcement learning** algorithm uses Q-values (action values) to iteratively improve the behavior of the learning agent.
- Goal is to maximize the Q value to find the optimal action-selection policy.
- The Q table helps to find the best action for each state and maximize the expected reward.
- **Q-Values / Action-Values:** Q-values are defined for states and actions.
- $Q(s, a)$ denotes an estimation of the action a at the state s.
- This estimation of $Q(s, a)$ will be iteratively computed using the **TD-Update rule.**
- **Reward:** At every transition, the agent observes a reward for every action from the environment, and then transits to another state.
- **Episode:** If at any point of time the agent ends up in one of the terminating states i.e. there are no further transition possible is called **completion of an episode.**

Q-Learning

- Initially agent explore the environment and update the Q-Table. When the Q-Table is ready, the agent will start to exploit the environment and taking better actions.
- It is an off-policy control algorithm i.e. the updated policy is different from the behavior policy. It estimates the reward for future actions and appends a value to the new state without any greedy policy

Temporal Difference or TD-Update:

- Estimate the value of Q is applied at every time step of the agents interaction with the environment $Q(S,A) \leftarrow Q(S,A) + \alpha\left(R + \gamma Q(S',A') - Q(S,A)\right)$

Advantage:

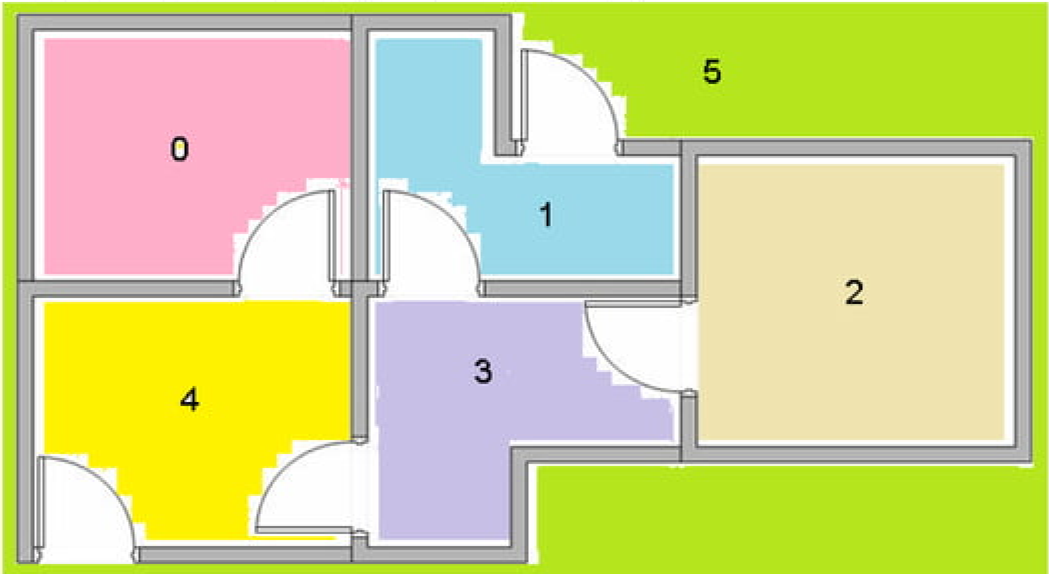
- Converges to an optimal policy in both deterministic and nondeterministic MDPs.

Disadvantage:

- Suitable for small problems.

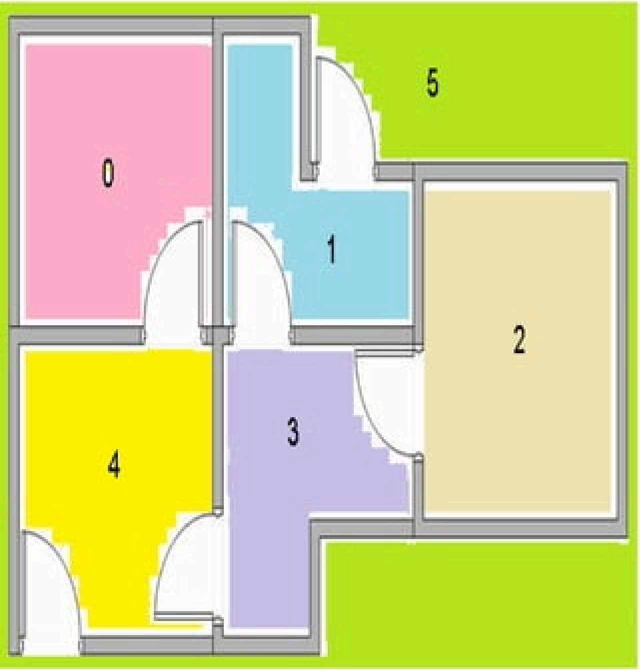
Understanding the Q – Learning

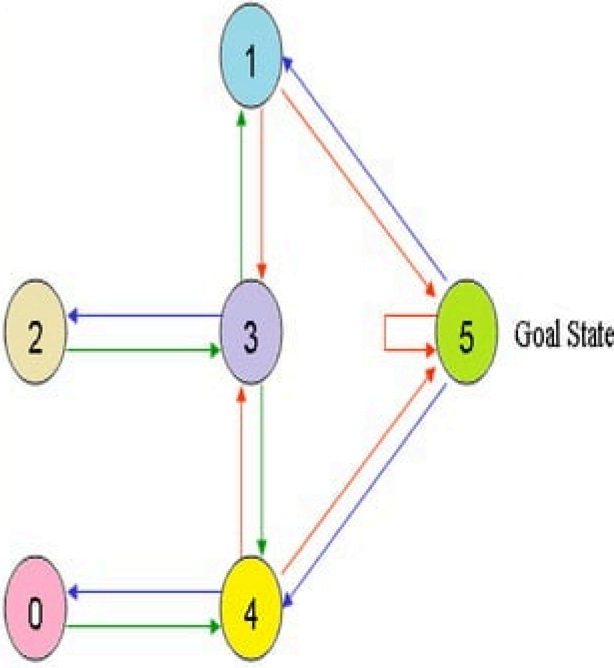
- Building Environment contains 5 rooms that are connected with doors.
- Each room is numbered from 0 to 4. The building outside is numbered as 5.
- Doors from room 1 and 4 leads to the building outside 5.
- **Problem:** Agent can place at any one of the rooms (0, 1, 2, 3, 4). Agent's goal is to reach the building outside (room 5).



Understanding the Q – Learning

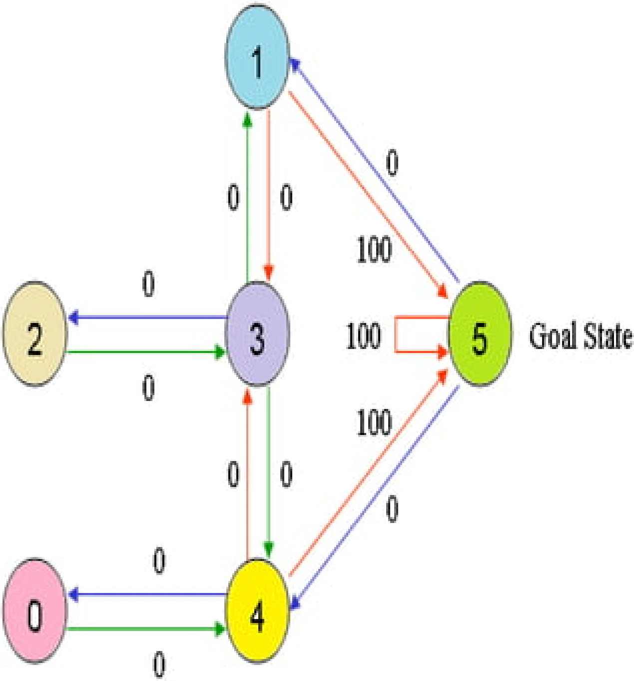
- Represent the room in the graph.
- Room number is the state and door is the edge.





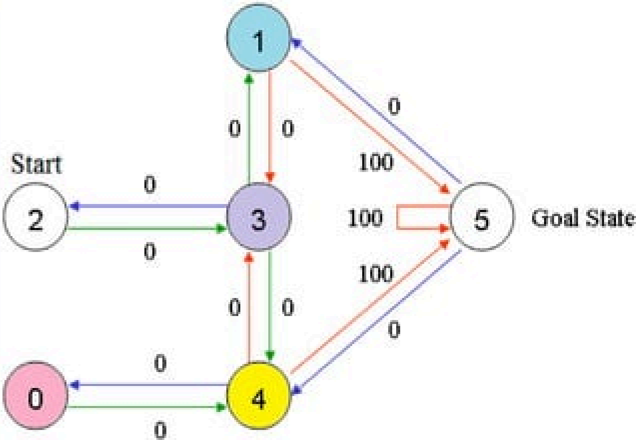
Understanding the Q – Learning

- Assign the Reward value to each door.
- The doors lead immediately to target is assigned an instant reward of 100.
- Other doors not directly connected to the target room have **zero** reward.
- For example, doors are two-way (0 leads to 4, and 4 leads back to 0), two edges are assigned to each room.
- Each edge contains an instant reward value



Understanding the Q – Learning

- Let consider agent starts from state *s* (Room) 2.
- Agent's movement from one state to another state is action *a*.
- Agent is traversing from state 2 to state 5 (Target).
  - Initial state = current state i.e. state 2
  - Transition State 2 → State 3
  - Transition State 3 → State (2, 1, 4)
  - Transition State 4 → State 5

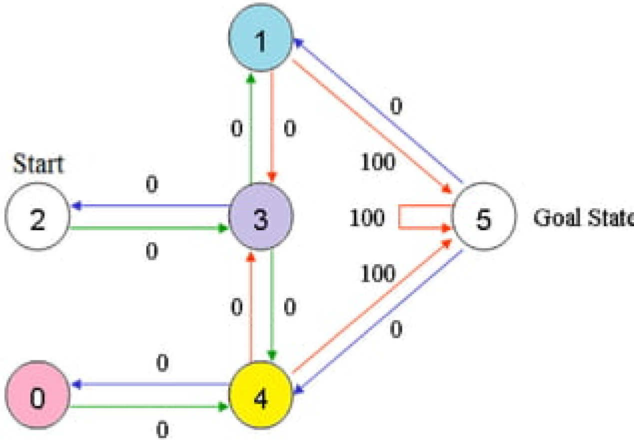


Understanding the Q – Learning

Prepare rewards table R (matrix)

- -1 denotes the no edge between the states
- 0 represents the indirect edge to the target

$$R = \begin{bmatrix} -1 & -1 & -1 & -1 & 0 & -1 \\ -1 & -1 & -1 & 0 & -1 & 100 \\ -1 & -1 & -1 & 0 & -1 & -1 \\ -1 & 0 & 0 & -1 & 0 & -1 \\ 0 & -1 & -1 & 0 & -1 & 100 \\ -1 & 0 & -1 & -1 & 0 & 100 \end{bmatrix}$$



Understanding the Q – Learning: **Prepare matrix Q**

- Matrix Q is the memory of the agent in which learned information from experience is stored.
- Row denotes the current state of the agent
- Column denotes the possible actions leading to the next state

Compute Q matrix:

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

- Gamma is discounting factor for future rewards. Its range is 0 to 1. i.e. $0 < Gamma < 1$.
- Future rewards are less valuable than current rewards so they must be discounted.
- If Gamma is closer to 0, the agent will tend to consider only the immediate rewards.
- If Gamma is closer to 1, the agent will tend to consider only future rewards with higher edge weights.

Q – Learning Algorithm

- Set the gamma parameter
- Set environment rewards in matrix R
- Initialize matrix Q as Zero
  - Select random initial (source) state
    - Set initial state *s* = current state
      - Select one action *a* among all possible actions using exploratory policy
    - Take this possible action *a*, going to the *next state s'*.
    - Observe reward *r*
  - Get maximum Q value to go to next state based on all possible actions
- Compute:
  - *Q(state, action) = R(state, action) + Gamma * max[Q(next state, all actions)]*
- Repeat the above steps until reach the goal state i.e current state = goal state

Example: Q – Learning

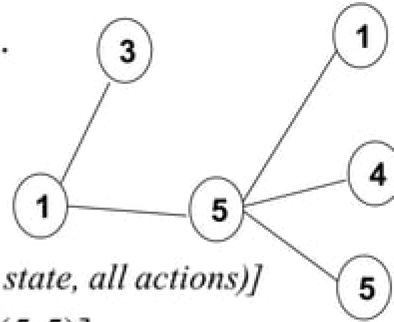
Matrix Q :

- Set the Gamma value = 0.8
- Initialize the matrix Q to zero matrix

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

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

Example: Q – Learning



- From state 1, agent can go to either state 3 or 5.
- Let's choose 5.
- From 5, Compute Max Q value to go to
- Next state based on all 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)]

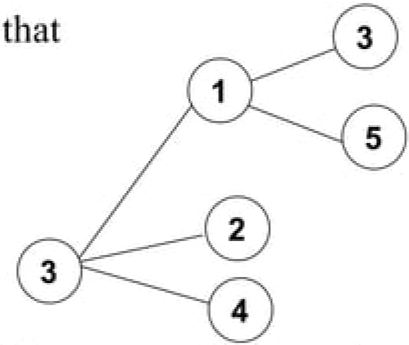
  = 100 + 0.8 * max[0, 0, 0] = 100 + 0 = **100**

- Update the Matrix Q.

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

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

- For next episode, choose next state 3 randomly that becomes current state.
- State 3 contains 3 choices i.e. state 1, 2 or 4.
- Let's choose state 1.
- Compute max Q value to go to next state based on all possible actions.



*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] = 0 + 80 = **80**

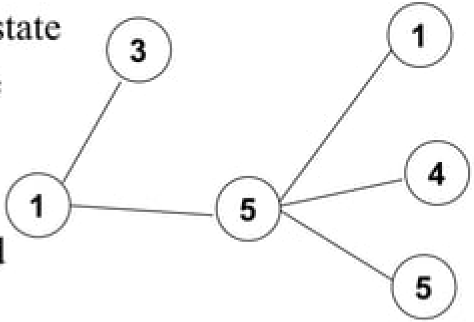
- Update the Matrix Q.

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

Action

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

- For next episode, next state 1 becomes current state
- Repeat the inner loop due to 1 is not target state
- From State 1, either can go to 3 or 5.
- Let's choose state 5.
- Compute max Q value to go to **next state** based on all 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 * max[0, 0, 0] = 100 + 0 = **100**
- Q remains the same due to Q(1,5) is already fed into the agent. Stop process

$$R = \begin{array}{c|cccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 0 & -1 & -1 & -1 & -1 & 0 & -1 \\ 1 & -1 & -1 & -1 & 0 & -1 & 100 \\ 2 & -1 & -1 & -1 & 0 & -1 & 100 \\ 3 & -1 & 0 & 0 & -1 & 0 & -1 \\ 4 & 0 & -1 & -1 & 0 & -1 & 100 \\ 5 & -1 & 0 & -1 & -1 & 0 & 100 \end{array}$$

Action

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

References

- Tom Markiewicz& Josh Zheng,Getting started with Artificial Intelligence, Published by O'Reilly Media,2017
- Stuart J. Russell and Peter Norvig,Artificial Intelligence A Modern Approach
- Richard Szeliski, Computer Vision: Algorithms and Applications, Springer 2010