

Reinforcement Learning

Prof. G Panda, FNAE, FNASc, FIET(UK)
IIT Bhubaneswar

Outline

- Introduction
- What is Reinforcement Learning(RL)?
- Examples
- Markov Decision Process
- Q-Learning
- Solving RL problem
- Applications



Introduction

Machine Learning :

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed.

A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data



Types of Machine Learning

With respect to the feedback type to learner:

Supervised learning : Task Driven (Classification)

Unsupervised learning : Data Driven (Clustering)

Reinforcement learning : Agent learns to interact with environment to achieve a reward.



Supervised Learning vs Reinforcement Learning

Supervised Learning:

Step: 1

Teacher: Does picture 1 show a car or a flower?

Learner: A flower.

Teacher: No, it's a car.

Step: 2

Teacher: Does picture 2 show a car or a flower?

Learner: A car.

Teacher: Yes, it's a car.

Step: 3



Supervised Learning vs Reinforcement Learning

Reinforcement Learning:

Step: 1

World: You are in state 9. Choose action A or C.

Learner: Action A.

World: Your reward is 100.

Step: 2

World: You are in state 32. Choose action B or E.

Learner: Action B.

World: Your reward is 50.

Step: 3



What is Reinforcement Learning?

Reinforcement learning is an important type of Machine Learning where an agent learn how to behave in a environment by performing actions and seeing the results.

Reinforcement learning emphasizes learning feedback that **evaluates the learner's performance without providing standards of correctness** in the form of behavioral targets.

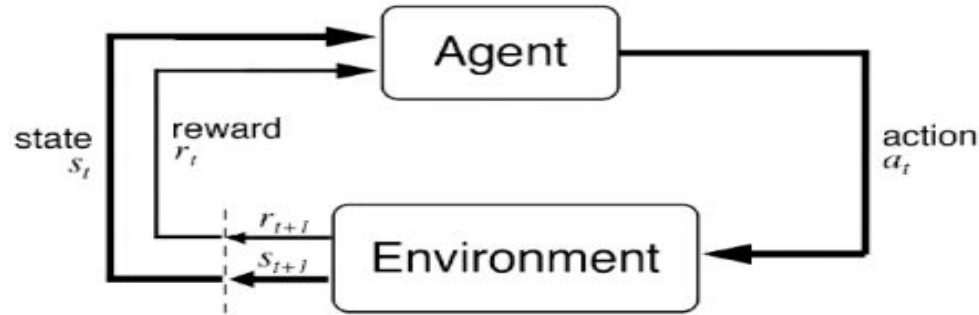


Examples

- Bicycle riding.
- Baby learning to walk.
- Rat in a maze



Agent - Environment interaction



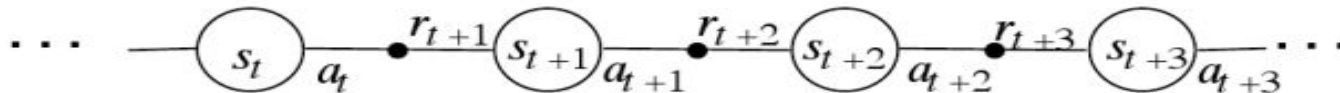
Agent and environment interact at discrete time steps : $t = 0, 1, 2, \dots$

Agent observes state at step t : $s_t \in S$

produces action at step t : $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \mathfrak{R}$

and resulting next state : s_{t+1}



Elements of Reinforcement Learning

Agent: Intelligent programs

Environment: External condition

Policy:

1. Defines the agent's behavior at a given time
2. A mapping from states to actions
3. Lookup tables or simple function



Elements of Reinforcement Learning

Reward function :

- Defines the goal in an RL problem

Value function:

- Reward function indicates what is good in an immediate sense while a value function specifies what is good in the long run.
- Value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

Steps for Reinforcement Learning

1. The agent observes an input state.
2. An action is determined by a decision making function (policy).
3. The action is performed.
4. The agent receives a scalar reward or reinforcement from the environment.
5. Information about the reward given for that state / action pair is recorded.



Associative and Non-Associative

Associative :

Situation Dependent and mapping from situation to the actions that are best in that situation.

Non-Associative:

Situation independent. No need for associating different action with different situations.



Exploration vs Exploitation

Exploration: Searching the space for possible solutions.

Exploitation: Taking advantage of existing best solutions.

- There is a trade-off between them. Both should be balanced.

ϵ - Greedy algorithm:

- With ϵ probability chooses random action.
- With $1-\epsilon$ probability chooses action which gives maximum cumulative reward.



1-armed Bandit problem

- Pull arms sequentially so as to maximize the Total reward.
- Non-associative.
- No exploration only exploitation is done.



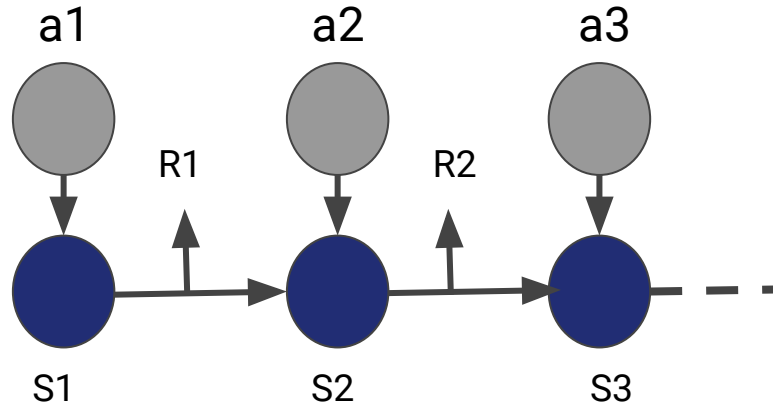
N-armed Bandit

- Sequence of Bandit's
- Associative
- Can Explore new Bandit's as well as
- Exploit particular Bandit



Markov Decision Process(MDP)

MDP: Formulating Reinforcement learning problem into Mathematical framework.



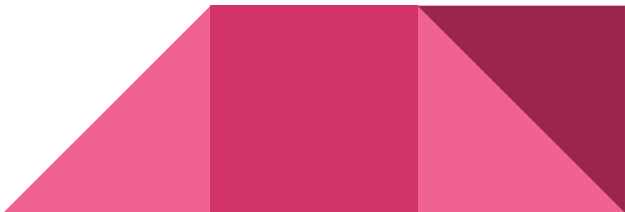
$a \rightarrow$ actions
 $S \rightarrow$ states
 $R \rightarrow$ rewards

Goal of Reinforcement Learning

Goal: To Maximize expected total reward

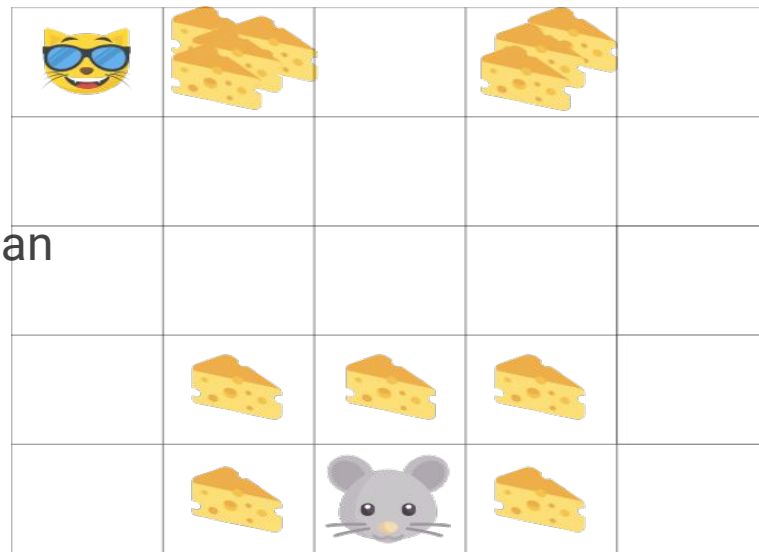
Total Reward: $G_t = R_{t+1} + R_{t+2} + R_{t+3} + \dots + R_T,$

Where, R_{t+1} indicates Reward in time stamp $t+1$.

- However, in reality we cannot just add the rewards like that, Since rewards in the next time stamp are more predictable than those in long run.
 - So, a discounting factor is introduced which accounts for uncertainty in obtaining future rewards
- 

Why to discount rewards?

- Considering agent as rat and opponent as cat. Goal is to maximize amount of cheese to be eaten by rat.
- Rewards near the rat are more probable than those near the cat.
- It is not sure that rat will be able to eat cheese near cat.
- So the reward near cat is to be discounted for the rat.



Discounting Rewards

- γ - is the discounting factor. Its value lies between 0 and 1.
- Larger the γ value, smaller the discount, indicates learning agent cares more about long term rewards.
- Smaller the γ value, larger the discount, indicates learning agent cares more about short term rewards.

Goal is to maximize expected cumulative discounted rewards.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$



Principle of Bellman Equation

The value of some state S_t at time t is sum of results to a terminal state, with reward of each successive state discounted.

The reward at subsequent state is discounted by γ

Discount factor is increased exponentially

$$V(S_t) = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots + \gamma^n R_{t+n}$$

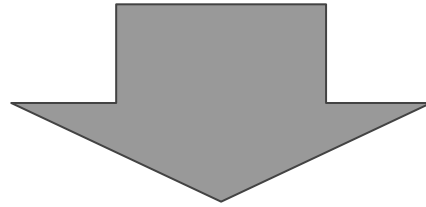
The Reward at the next step after taken some action a

The reward at next subsequent state is further discounted by γ^2



Bellman Equation

$$V(S_t) = R_t + \underbrace{\gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + \dots + \gamma^n R_{t+n}}_{\gamma V(S_{t+1})}$$



$$V(S_t) = R_t + \gamma(V(S_{t+1}))$$



Bellman optimality equation

- **State value based:** The value of a state is based on the best action(optimal) for that state and each subsequent state.

$$V(S_t) = \max_a (R(S_t, a) + \gamma(V(S_{t+1})))$$

The action a at state S_t
which maximizes the
reward



Illustration of Bellman optimality equation

$\gamma = 0.9$

		R=1
Wall		R=-1

Calculate 1 step away

	0.9 →	R=1
Wall		R=-1

Best action is move to the goal node.

Calculate adjacent steps

0.81 →	0.9 →	R=1
Wall	↑ 0.81	R=-1

Best action is move to the node with the highest value.

Calculate adjacent steps

0.81 →	0.9 →	R=1
Wall	↑ 0.81	R=-1
	↑ 0.73	

Calculate adjacent steps

0.81 →	0.9 →	R=1
Wall	↑ 0.81	R=-1
0.66 →	↑ 0.73	← 0.66



This produces a plan.

The Optimal Action (Move) for each state.

Bellman optimality equation

- **Action value based:** The value(Q-value or Quality value) of taking an action in that state is based on the best action(optimal) for next state and followed by subsequent states.

$$Q(S_t, a) = R(S_t, a) + \gamma [\max_{a'} Q(S_{t+1}, a')]$$

Q-value after choosing an action 'a' in the state S_t

Maximum of Q-value of next state S_{t+1} for all possible actions a'



Q-learning

- It is an algorithm that solves Reinforcement learning problems using Bellman equations.
- It learns optimal policy which says optimal action possible at any given state.
- It uses Q-table with rows indicating states and columns indicating actions.
- α is learning rate.

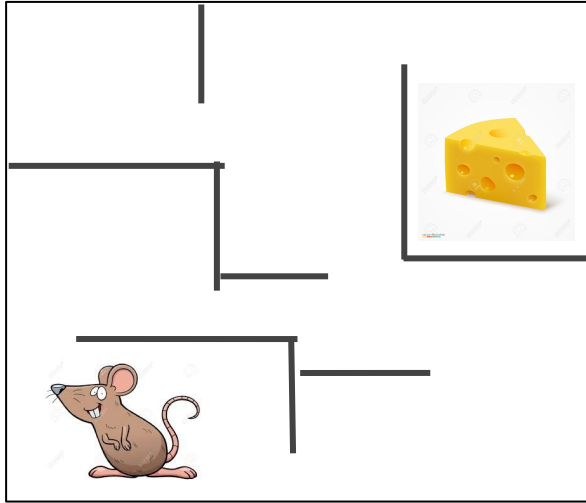
Equation:

$$Q(S_t, a) = (1 - \alpha)Q(S_t, a) + \alpha(R(S_t, a) + \gamma \max_{a'} Q(S_{t+1}, a'))$$


Algorithm Using Q-learning

1. Initialize $Q[][]$ (Q_table) size $n \times n \rightarrow$ zero
2. Define $R[][]$ (Reward table) size $n \times n$
3. α (learning rate) = 0.1 and γ (discount factor) = 0.8
4. For i in range(1, L): [L is a large number]
5. Choose random initial state ' S '
6. Find valid action ' a '
7. While not reached destination:
8. Compute $Q(S,a) = (1-\alpha)Q(S,a) + \alpha(R(S,a) + \gamma \max_{a'} Q(S',a'))$
9. Update $S = S'$ and $a = a'$
10. choosing an initial state ' s '
11. Test for optimal path by choosing $s' = \max_a Q(s,a)$
12. Repeat step-11 sequentially until destination is reached.

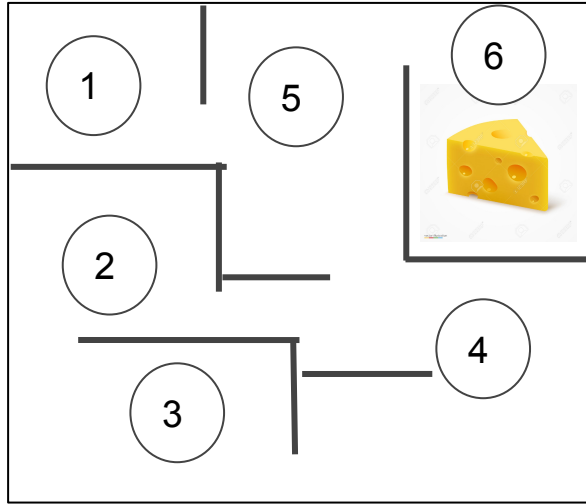
Example : Rat in a maze



Goal : To find optimal(shortest) path from rat to cheese

Example : Rat in a maze

Training Phase : We start training by assuming that initially rat can be in any state among possible 6 states.

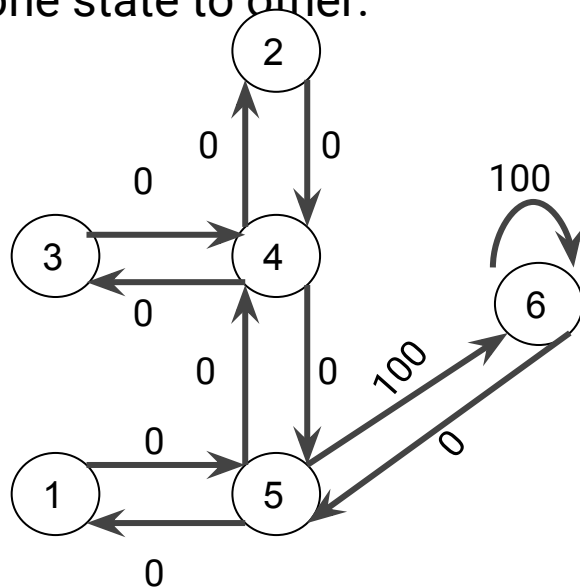


○ -> Indicates possible states of the rat(Agent)

6 -> Goal state

Graph representation of States

- Values on edges represents rewards gained when transitions(actions) are made from one state to other.



→ Indicates action
<0,100> are rewards

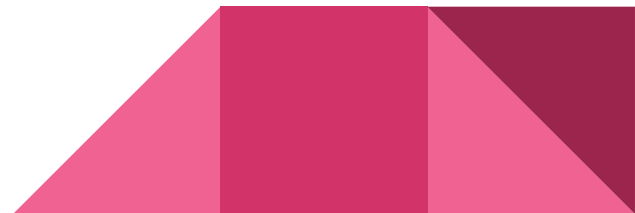
Solving the problem

Initialize Reward-table:

	1	2	3	4	5	6
1	-1	-1	-1	-1	0	-1
2	-1	-1	-1	0	-1	-1
3	-1	-1	-1	0	-1	-1
4	-1	0	0	-1	0	-1
5	0	-1	-1	0	-1	100
6	-1	-1	-1	-1	0	100

Reward :

- 1 -> no path exist
- 0 -> path exist but does not reach destination
- 100 -> reach destination



Solving the problem

Initialize Q-table:

		action					
		1	2	3	4	5	6
State	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	0	0	0	0	0	0



Solving the problem

Applying the Q-learning algorithm:

Using Bellman equation:

$$Q(\text{State}, \text{action}) = R(\text{State}, \text{action}) + \gamma[\max(Q(\text{next state}, \text{all actions}))]$$

We start building the optimal path, assuming agent starts in goal state and from there going backwards.

At each state we consider the optimal possible (state, action) pair.



Training Phase

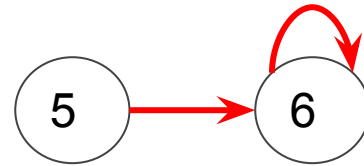
Step - 1: First optimal action is **moving to final state 6**, which is reachable from state 5 and 6.

$$Q(6,6) = R(6,6) + 0.8 * \max(Q(6,6), Q(6,5))$$

$$= 100 + 0.8 * \max(0, 0) = 100$$

$$Q(5,6) = R(5,6) + 0.8 * \max(Q(6,6), Q(6,5))$$

$$= 100 + 0.8 * \max(0, 0) = 100$$



Note: $Q(a,b)$ correspond to row a and column b in initial Q-table

Training Phase

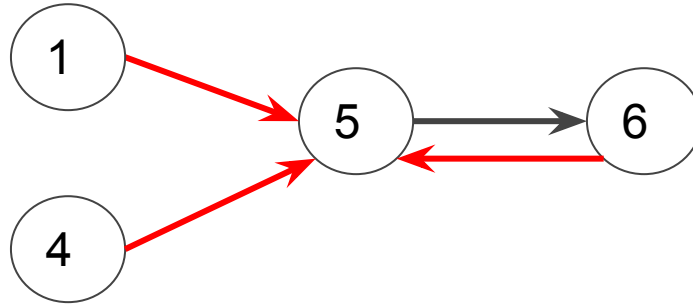
update Q-table after step -1:

	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	100
6	0	0	0	0	0	100



Training Phase

Step - 2: Next optimal action can be moving to state 5 which will lead to previously found optimal path(5->6). State 5 is reachable from 1, 4 and 6.

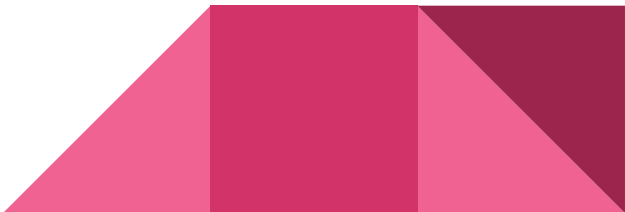


Training Phase

Computing Q-table values:

$$\begin{aligned} Q(1,5) &= R(1,5) + 0.8 * \max(Q(5,1), Q(5,4), Q(5,6)) \\ &= 0 + 0.8 * \max(0, 0, 100) = 80 \end{aligned}$$

$$\begin{aligned} Q(4,5) &= R(4,5) + 0.8 * \max(Q(5,1), Q(5,4), Q(5,6)) \\ &= 0 + 0.8 * \max(0, 0, 100) = 80 \end{aligned}$$

$$\begin{aligned} Q(6,5) &= R(6,5) + 0.8 * \max(Q(5,1), Q(5,4), Q(5,6)) \\ &= 0 + 0.8 * \max(0, 0, 100) = 80 \end{aligned}$$


Training Phase

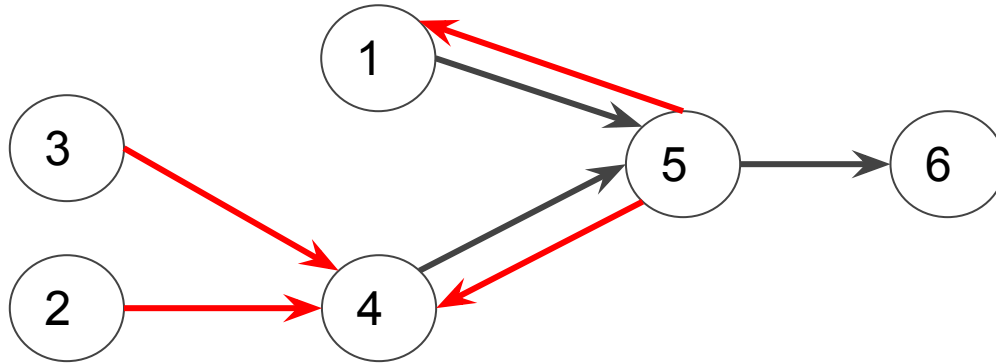
update Q-table after step -2:

	1	2	3	4	5	6
1	0	0	0	0	80	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	80	0
5	0	0	0	0	0	100
6	0	0	0	0	80	100



Training Phase

Step - 3: Next optimal action is moving to state 1 or 4 which will lead to previously found optimal paths(1->5->6 or 4->5->6). State 1 is reachable from 5 and state 4 is reachable from 2, 3 and 5.



Training Phase

Computing Q-table values:

$$\begin{aligned} Q(2,4) &= R(2,4) + 0.8 * \max(Q(4,2), Q(4,3), Q(4,5)) \\ &= 0 + 0.8 * \max(0, 0, 80) = 64 \end{aligned}$$

$$\begin{aligned} Q(5,4) &= R(5,4) + 0.8 * \max(Q(4,2), Q(4,3), Q(4,5)) \\ &= 0 + 0.8 * \max(0, 0, 80) = 64 \end{aligned}$$

$$\begin{aligned} Q(3,4) &= R(3,4) + 0.8 * \max(Q(4,2), Q(4,3), Q(4,5)) \\ &= 0 + 0.8 * \max(0, 0, 80) = 64 \end{aligned}$$

$$Q(5,1) = R(5,1) + 0.8 * \max(Q(1,5)) = 0 + 0.8 * \max(80) = 64$$

Training Phase

Computing Q-table values:

$$\begin{aligned} Q(5,1) &= R(5,1) + 0.8 * \max(Q(1,5)) \\ &= 0 + 0.8 * \max(80) = 64 \end{aligned}$$



Training Phase

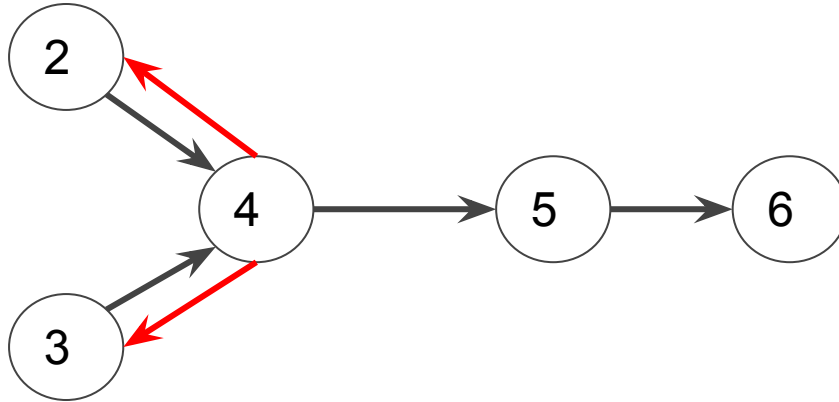
update Q-table after step -3:

	1	2	3	4	5	6
1	0	0	0	0	80	0
2	0	0	0	64	0	0
3	0	0	0	64	0	0
4	0	0	0	0	80	0
5	64	0	0	64	0	100
6	0	0	0	0	80	100



Training Phase

Step - 4: Next optimal action is moving to state 2 or 3 which will lead to previously found optimal paths(2->4->5->6 or 3->4->5->6). The only reachable state to 2 or 3 is state 4.



Training Phase

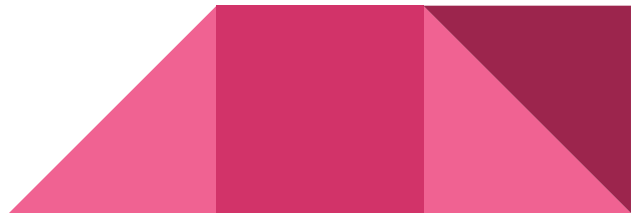
Computing Q-table values:

$$Q(4,2) = R(4,2) + 0.8 * \max(Q(2,4))$$

$$= 0 + 0.8 * \max(64) = 51.2$$

$$Q(4,3) = R(4,3) + 0.8 * \max(Q(3,4))$$

$$= 0 + 0.8 * \max(64) = 51.2$$



Training Phase

update Q-table after step -4:

	1	2	3	4	5	6
1	0	0	0	0	80	0
2	0	0	0	64	0	0
3	0	0	0	64	0	0
4	0	51.2	51.2	0	80	0
5	64	0	0	64	0	100
6	0	0	0	0	80	100



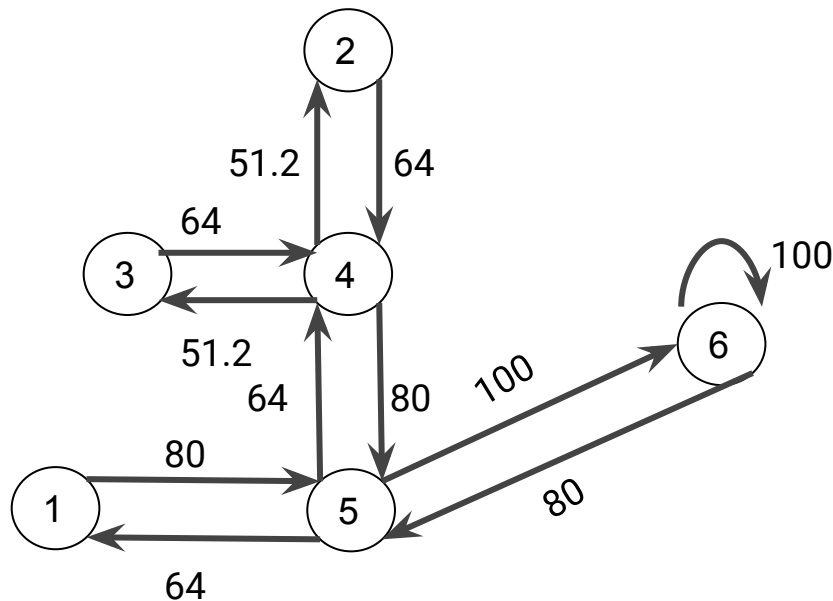
Training Phase

- This procedure of finding Q-values is repeated large number of times until convergence (when the normalized values not differ in two successive steps).
- When initial states are chosen at random, Q values obtained will be different and will change for each iteration. However the normalized Q values will finally converge to the above computed values.
- **Normalization** : $Q/\max(Q)*100$ [Divided by 100 to convert into percentage]



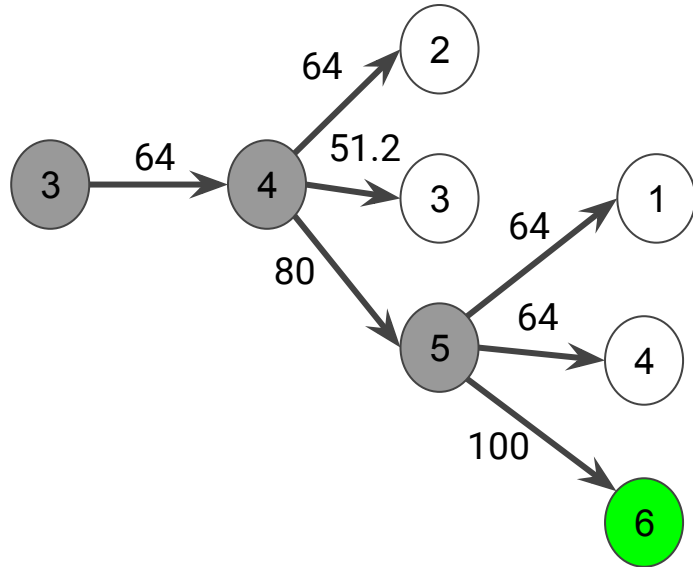
Graph representation of final Q-Table

- Edges represents learned Q-values



Test Phase

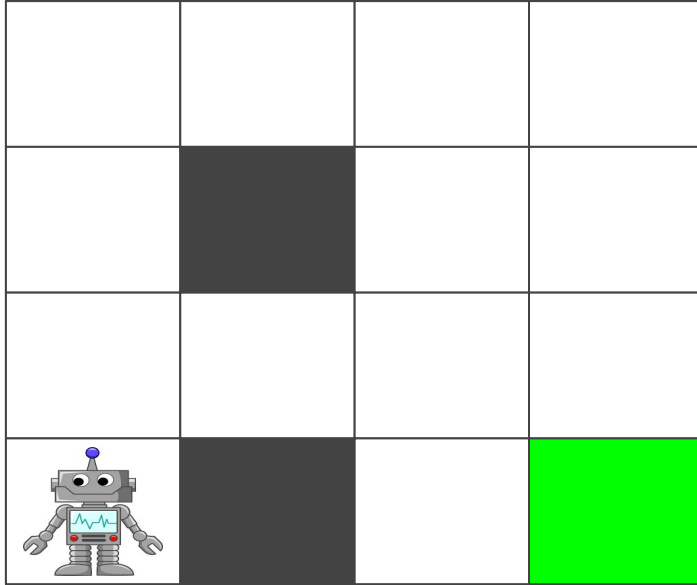
- Since rat is in state 2: Proceed by choosing action with maximum Q-value



Chosen path by rat is:

3 -> 4 -> 5 -> 6

Example 2 : Robot in a room



Goal : To reach the destination(box in green color) in shortest moves. Black boxes indicate walls.



Some applications

- Games.
- Robotics.
- Self-Driving cars.
- Resource Management in computer clusters.



References

- Reinforcement Learning: An Introduction Second edition, Richard S. Sutton and Andrew G. Barto c 2014, 2015.
- Hands-On Reinforcement Learning with Python: Master Reinforcement and Deep Reinforcement Learning Using OpenAI Gym and TensorFlow Book by Sudharsan Ravichandiran
- Algorithms for Reinforcement Learning Book by Csaba Szepesvari



Thank You

