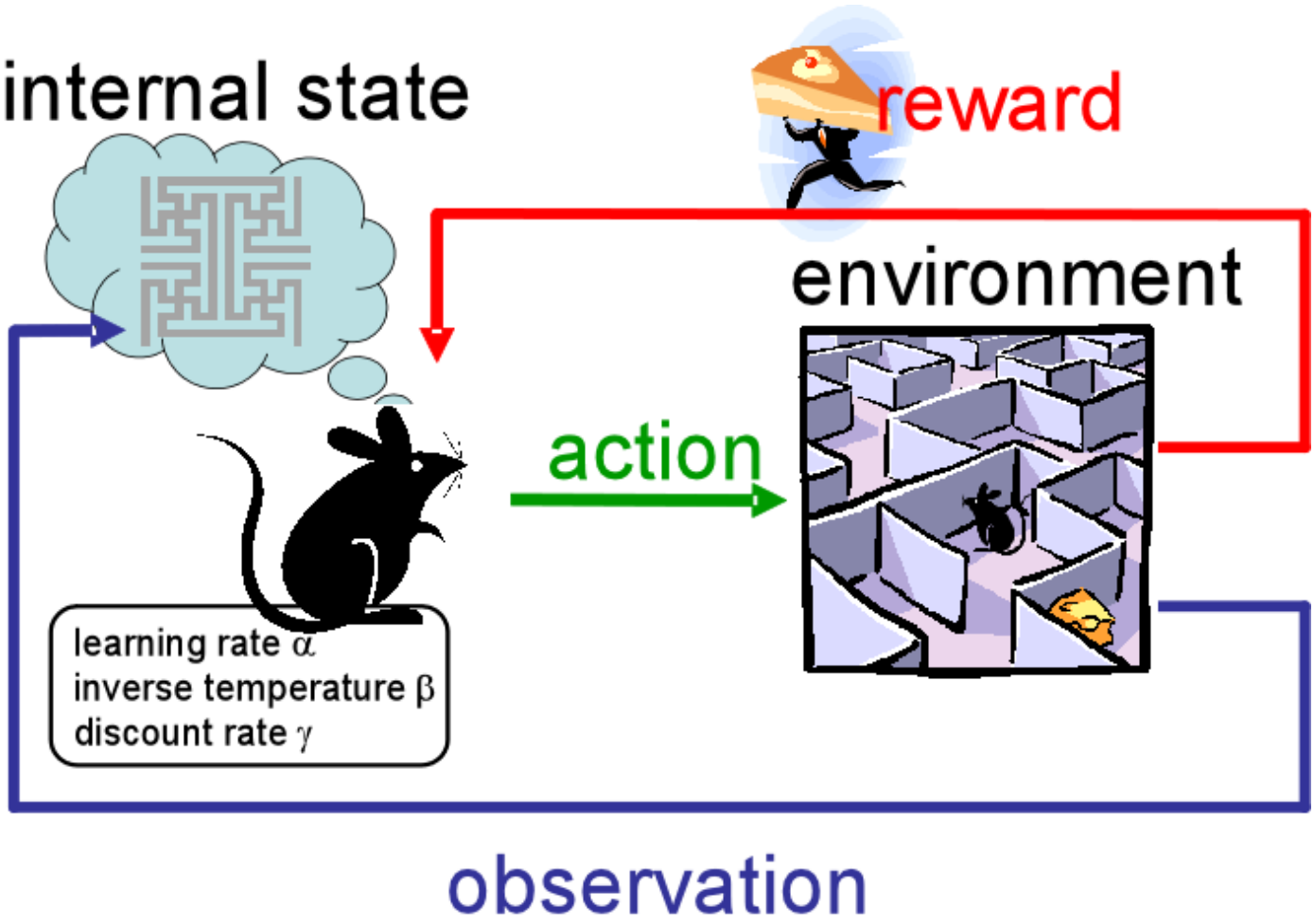
Published in Becoming Human: Artificial Intelligence Magazine

Aneek Das Follow
Mar 26, 2017 · 7 min read · Listen

The very basics of Reinforcement Learning



Reinforcement Learning

This article will be a brief diversion from my first post on Q Learning(link given at the

next one.

Reinforcement Learning is the science of making optimal decisions. It helps us formulate reward-motivated behaviour exhibited by living species . Lets say, you want to make a kid sit down to study for an exam. It is very difficult to do so, but if you give him a bar of chocolate every time he finishes a chapter/topic he will understand that if he keeps on studying he will get more chocolate bars. So he will have some motivation to study for the exam. Now initially the kid has no sense of time or how to prepare(he might go through every line and ponder upon it). He might take up hours studying a topic and never finish the syllabus in time. So, lets say if he finishes a topic within an hour we give him a huge bar, if he takes 1.5 hours a regular bar and a toffee if he takes longer than that. So, now not only does he study but his brain devises ways in which he can finish topics faster.

The kid represents the **Agent** . The reward system and the exam represent the **Environment**. The topics are analogous to **States** in reinforcement learning. So, the kid has to decide which topics to give more importance to(i.e., to calculate the value of each topic). This will be the work of our **Value-Function .** So, every time he travels from one state to another he gets a **Reward** and the methods he uses to complete the topics within time is our **Policy**.

This is carried out by our Dopamine system in our brain which takes care of reward-motivated behaviour. Most types of rewards increase the level of dopamine in the brain. For example, when you take addictive drugs it increases dopamine neuronal activity. So you feel like you are rewarded and you feel satisfied, hence the effect.

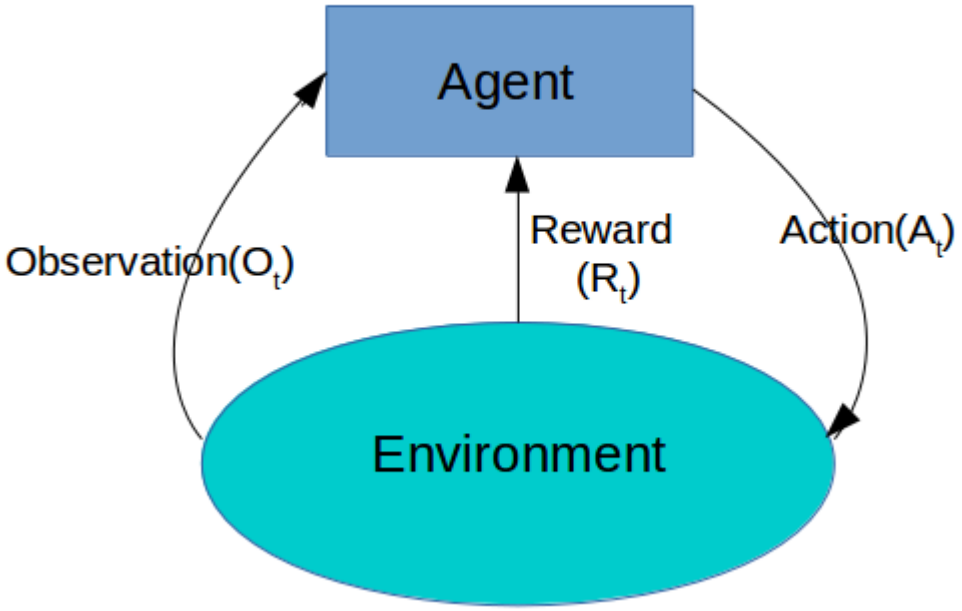
How is it different from other machine learning paradigms :

1. **There is no supervisor present.** So, there is no one to tell you the best possible action. You just get a reward for each action.
2. **Importance of time.** Reinforcement Learning pays much attention to sequential data unlike other paradigms where you receive random inputs. Here the input at




step only to find out that you made a huge blunder in a future step.

4. **The agents action effects its next input.** Say, you have the choice of going either left or right. After you take the action, the input at the next time step will be different if you chose right rather than left.



Q Learning

So, at any time-step (t), our agent sees an observation(instance) of the environment. Then out of the actions possible it chooses an action for which the environment gives it a reward and the next observation. So we will provide algorithms to our agent to take decisions with the *goal to maximize our overall reward at the end.*

History v/s State

History is the collection of all the variables that describe all the events that have taken place between the environment and the agent.

lasting and to store so much information would take up lots of space and processing time.

So, we want to create an abstract representation of our history, which stores sufficient information so that we can choose our next action. That is called State. So basically our outcome depends on how well we define our state.

$$S_t = f(H_t)$$

Where 'S' represents state and 'f' represents a function that can summarise the history at time-step 't'.

$$S_t^a \text{ and } S_t^e$$

The former represents the internal representation of the agent that summarises it's history and allows it to take future actions while the latter represents the internal representation of the environment that enables it to emit the next observation.

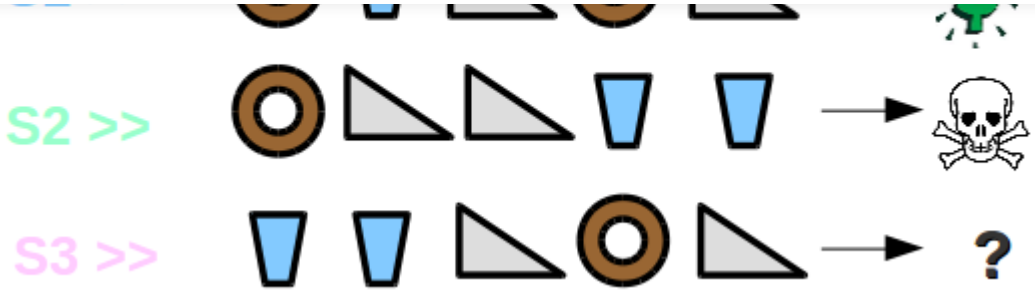
Markov State

A Markov state stores all the information from the past in an abstract form. So, say if we want to predict the future, rather than using the whole history, we can use the Markov State. The Markov State essentially contains no less information than the history.

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1,.....S_t]$$

So, here the probability of getting to a future state St+1 given state St is the same as getting to St+1 given all the previous states. This is because the state St already contains the information about the previous states embedded in it.



Say, we have a game in which there is a waiter at a restaurant. The restaurant offers 3 items, ***Donuts, Drinks and Sandwiches***. The customer tells the waiter to bring 5 items , one at a time. If the customer likes the sequence of the items, he rewards the waiter, else, the waiter gets shouted at. Now, the first time, he brings a donut, a drink, a sandwich, another donut and another sandwich sequentially. Luckily, the customer likes the order and gives the waiter a tip.

However, the next time the customer tells the waiter to bring 5 more items . He brings a donut, 2 sandwiches and 2 drinks sequentially. However, this time he gets shouted at.

So, what should be the outcome of the third sequence ????

Here the solution lies in the definition of the previous state. If say the state was defined solely as the sequence of bringing the items then he is most likely to get rewarded as the last 3 items of S1 match with S3. However, if the state is based on the number of each item then the waiter is most likely to get shouted at as S2 and S3 have the same number of each item. So, essentially the outcome depends on the definition of state.

The Environment

Say, you are playing a game of chess on a computer and you have coded the game yourself. So, you have a fair idea of how the computer analyses your move and how it predicts the next move given your move. Basically, you know exactly what the next move the computer will play given your move. This is a **Fully Observable Environment.**

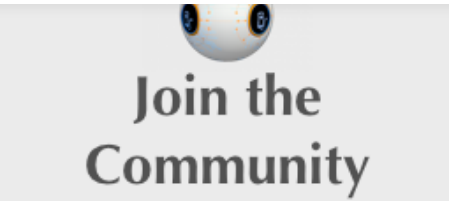
Now, consider you are playing a game of chess against a computer which was coded by

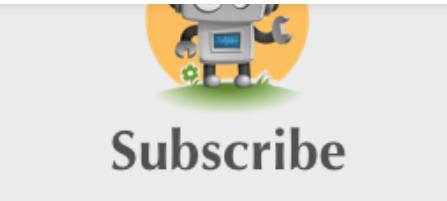
this case the most you can do is try and predict the internal dynamics of the game hoping that you made a close enough approximation of it's internal state.

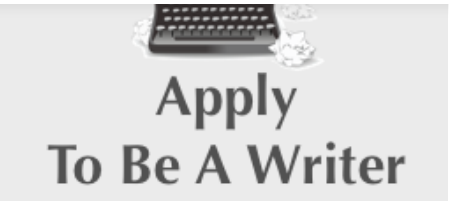
So, that will be all for this article. In the next one I will cover the **components of an RL Agent** and on the one after I will resume Q-Learning and how to train agents to play games using it. Until then, bye.

Link to **Introduction to Q-Learning :**

Introduction to Q-Learning
Imagine yourself in a treasure hunt in a maze . The game is as follows :
medium.com







Sign up for Latest from Becoming Human AI

By Becoming Human: Artificial Intelligence Magazine

Watch AI & Bot Conference for Free Take a look.

Emails will be sent to jimjywang@gmail.com. Not you?

Get this newsletter

Published in Towards Data Science

Aneek Das Follow

Mar 13, 2017 · 5 min read · Listen

Introduction to Q-Learning

Imagine yourself in a treasure hunt in a maze . The game is as follows :

You start at a given position, the starting state . From any state you can go left, right, up or down or stay in the same place provided you don't cross the premises of the maze. Each action will take you to a cell of the grid (a different state). Now, there is a treasure chest at one of the states (the goal state). Also, the maze has a pit of snakes in certain positions/states. Your goal is to travel from the starting state to the goal state by following a path that doesn't have snakes in it.

Start		Snakes	
-	Snakes	Snakes	
-	Snakes		
-	-	-	Treasure

Grid outline of the maze

When you place an agent in the grid(we will refer to it as our environment) it will first explore. It doesn't know what snakes are , neither does it know what or where the treasure is. So, to give it the idea of snakes and the treasure chest we will give some rewards to it after it takes each action. For every snake pit it steps onto we will give it a reward of -10. For the treasure we will give it a reward of +10. Now we want our agent

has to get the treasure as fast as possible. The '-' path in the figure shows the shortest path with maximum reward.

Q-Learning attempts to learn the value of being in a given state, and taking a specific action there.

What we will do is develop a table. Where the rows will be the states and the columns are the actions it can take. So, we have a 16x5 (80 possible state-action) pairs where each state is one cell of the maze-grid.

We start by initializing the table to be uniform (all zeros), and then as we observe the rewards we obtain for various actions, we update the table accordingly. We will update the table using the ***Bellman Eq***

$$Q(s,a) = r + \gamma(max(Q(s',a'))$$

eqn.1

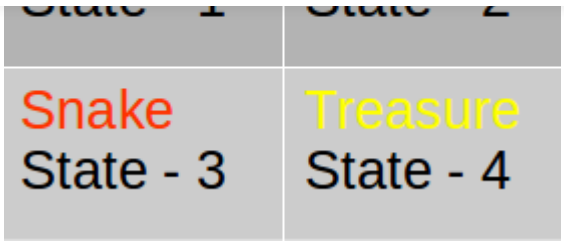
'S' represents the current state . 'a' represents the action the agent takes from the current state. ' S' ' represents the state resulting from the action. 'r' is the reward you get for taking the action and 'γ' is the discount factor. So, the Q value for the state 'S' taking the action 'a' is the sum of the instant reward and the discounted future reward (value of the resulting state). The discount factor 'γ' determines how much importance you want to give to future rewards. Say, you go to a state which is further away from the goal state, but from that state the chances of encountering a state with snakes is less, so, here the future reward is more even though the instant reward is less.

We will refer to each iteration(attempt made by the agent) as an episode. For each episode, the agent will try to achieve the goal state and for every transition it will keep on updating the values of the Q table.

Lets see how to calculate the Q table :

For this purpose we will take a smaller maze grid for ease



The initial Q-table would look like (states along the rows and actions along the columns) :

```
   U D L R N
1[ 0 0 0 0 0 ]
2[ 0 0 0 0 0 ]
3[ 0 0 0 0 0 ]
4[ 0 0 0 0 0 ]
```

Q Matrix

U — up, D — down, L — left, R — right

The reward table would look like :

```
    U   D   L   R   N
1[  E -10   E  -1  -1 ]
2[  E +10  -1   E  -1 ]
3[ -1   E   E +10  -1 ]
4[ -1   E -10   E +10]
```

R Matrix

Here, E represents NULL, i.e., there can be no such transitions.

Algorithm:

3. Select one among all possible actions for the current state (S).

4. Travel to the next state (S') as a result of that action (a).

5. For all possible actions from the state (S') select the one with the highest Q value.

6. Update Q-table using eqn.1 .

7. Set the next state as the current state.

8. If goal state reached then end.

Example : Lets say we start with state 1 . We can go either D or R. Say, we chose D . Then we will reach 3 (the snake pit). So, then we can go either U or R . So, taking $\gamma = 0.8$, we have :

$Q(1,D) = R(1,D) + \gamma*[max(Q(3,U) \& Q(3,R))]$

$Q(1,D) = -10 + 0.8*0 = -10$

Here, $max(Q(3,U) \& Q(3,R)) = 0$ as Q matrix not yet updated. -10 is for stepping on the snakes. So, new Q-table looks like :

	U	D	L	R	N
1[	0	-10	0	0	0]
2[	0	0	0	0	0]
3[	0	0	0	0	0]
4[	0	0	0	0	0]

Now, 3 is the starting state. From 3, lets say we go R. So, we go on 4 . From 4, we can go U or L .

$Q(3,R) = R(3,R) + 0.8*[max(Q(4,U) \& Q(4,L))]$

2	0	0	0	0	0
3	0	0	0	10	0
4	0	0	0	0	0

So, now we have reached the goal state 4. So, we terminate and more passes to let our agent understand the value of each state and action. Keep making passes until the values remain constant. This means that your agent has tried out all possible state-action pairs.

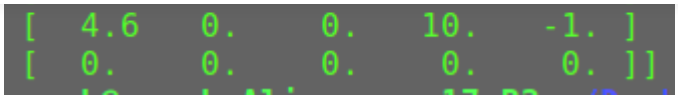
Implementation in python :




```
gamma = 0.8

#for matrices reward and q_matrix columns are in order (U, D, L, R, N)

# here the states are 0, 1, 2, 3 for convenience
# 0 is the starting state
# 1 is state to the right of 0
# 2 is the snake-pit state
# 3 is the treasure(goal state)

reward = np.array([[0, -10, 0, -1, -1],
                   [0, 10, -1, 0, -1],
                   [-1, 0, 0, 10, -1],
                   [-1, 0, -10, 0, 10]])

q_matrix = np.zeros((4,5))

# -1 represent invalid transitions
transition_matrix = np.array([[-1, 2, -1, 1, 1],
                              [-1, 3, 0, -1, 2],
                              [0, -1, -1, 3, 3],
                              [1, -1, 2, -1, 4]])

# for valid actions
# encoded up as 0, down as 1, left as 2, right as 3, no action as 4
# the rows are the states
valid_actions = np.array([[1, 3, 4],
                          [1, 2, 4],
                          [0, 3, 4],
                          [0, 2, 4]])

for i in range(1000): # 1000 episodes
    start_state = 0
    current_state = start_state
    while current_state != 3:
        action = random.choice(valid_actions[current_state])
        next_state = transition_matrix[current_state][action]
        future_rewards = []
        for action_nxt in valid_actions[next_state]:
            future_rewards.append(q_matrix[next_state][action_nxt])
        q_state = reward[current_state][action] + gamma*max(future_rewards)
        q_matrix[current_state][action] = q_state
        print(q_matrix)
        current_state = next_state
        if current_state == 3:
            print('goal state reached')

print('final q-matrix : ')
print(q_matrix)
```

Output for the last q_matrix :



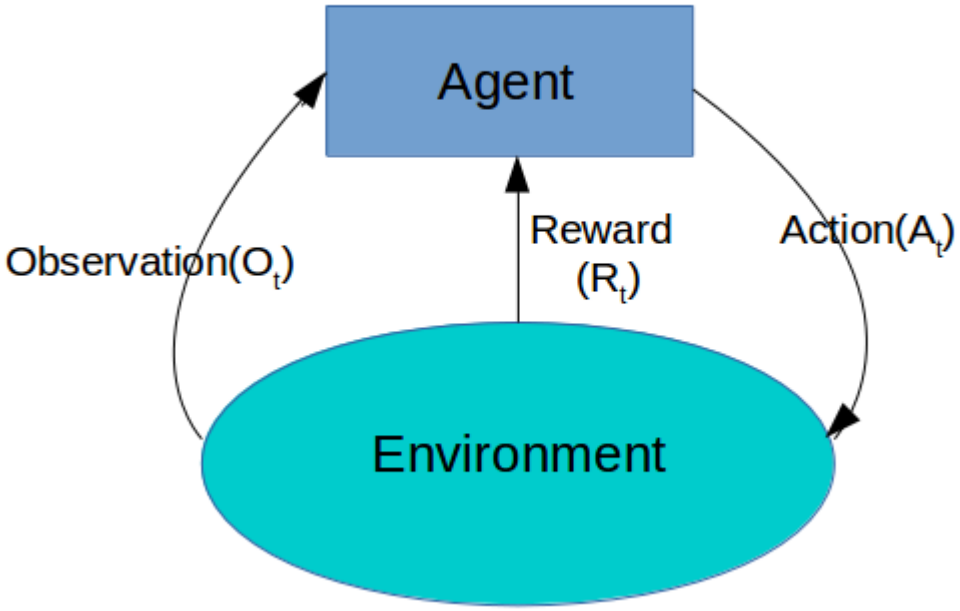



In the next article I will cover the usage of Neural Networks for Q-Learning and the short-comings of using the tabular approach. Also, we will be working on games from Open-AI gym for testing. Until then, bye.

Published in Becoming Human: Artificial Intelligence Magazine

Aneek Das Follow
Apr 2, 2017 · 8 min read · Listen

Components of a Reinforcement Learning Agent and it's application on SNAKE



This article is the third in my series about Q-Learning. If you have not read the previous two you might be interested to give them a read. The links will be provided at the end of this article. This article is divided into two parts, the first part covers the basic components of an RL agent while the second part applies those concepts on the classic game of SNAKE.

1. **Policy**
2. **Value Function**
3. **Model**

The RL agent may have one or more of these components. We shall go through each of them in detail.

POLICY

Policy defines the behaviour of the agent.

Say you have to reach a destination within a span of time. If you are lazy, you would look for shortcuts rather than taking the obvious long route. While if you are a methodical person you would rather take the long route which will guarantee that you reach your goal in time even though you have to do more work to get there. If you are taking a shortcut, you may reach the destination faster or you may not even reach the destination if your calculations are wrong or if there are obstacles in the way. Here, the *lazy* or *methodical* refers to as policy because it defines the way in which you approach a problem.So, in case of our agent, it is *how the agent picks it's actions.*

Say, you are looking for shortcuts and you have 2 choices at a particular time, left or right. Since you don't know which way to go, you assign a random probability to each direction.This is a **Stochastic Policy** as for every decision you have a probability given the current state. Over experience you want to optimise this probability distribution so that you get a higher chance of success while finding shortcuts.

$$\prod(a|s) = \mathbf{P}\left[A=a \mid S=s\right]$$

Stochastic Policy equation

maps states to actions. This is called **Deterministic Policy.**

$$a = \prod(s)$$

Deterministic Policy equation

Here, '∏' is just the policy function that simply gives an output state given your current state.

VALUE FUNCTION

Value Function defines how good it is to be in a particular state.

Consider the previous situation of having a decision to go either left or right. This time, you have a rough idea about the two paths, i.e., you abstractly know about the obstacles in each way and the estimated time it would take if you chose that path. So, you try to calculate which path will be more feasible for you by weighing both the reward and obstacles. This is called the value function.

$$V_{\prod}(s) = E_{\prod}\left[R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \ldots \mid S_t = s\right]$$

Value Function equation

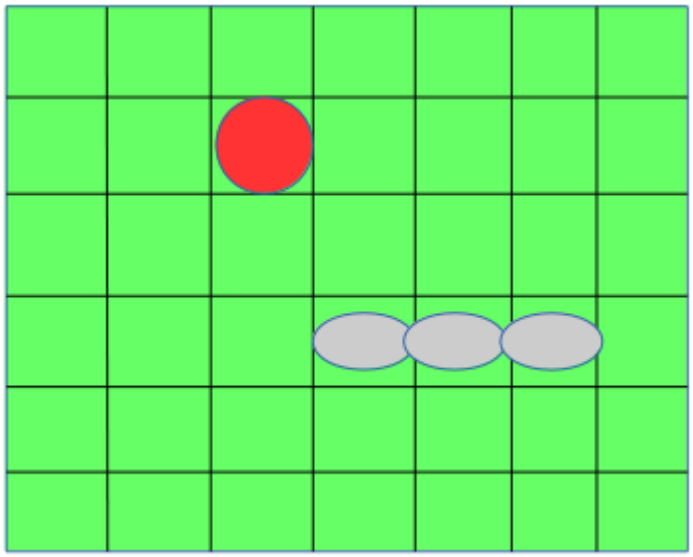
Here, 's' is our current state. '∏' is the policy that we use to define our behaviour. In the situation above '∏' is the behaviour of being *lazy, i.e.,* we got to the current state by being lazy .'R' represents the reward that we get for travelling to a state. 'γ' represents the discount factor which helps us weight the impact of future rewards. So, basically the value of being in a state is the sum of the present reward we are getting for being in that state and the future rewards we can get if we take different actions from that state. We discount the future rewards as the future is quite uncertain. Since, it is a stochastic




We again consider the situation of the 2 roads (left and right). You are standing at the intersection. You can see a little bit of the path ahead in each road. For example, the road to the left does not have proper lighting. Also, it is a muddy road. While the one to the right has proper lighting and is a concrete road. From that you will try to predict the road ahead and accordingly chose a path. So, for our agent, it will try to predict the dynamics of the environment with the data it has seen till that point. This is called the **Transition Model**.

Now, you take the road to the right and you found out that you have come closer to the goal. For the agent this will be it's reward. Now, at the end of the road you come across a similar situation. Again there are two similar roads (left and right) with the same attributes as before. Using your experience and intuition you will predict a similar journey you had before by taking the road to the right. So, according to your intuition you predict that you will come closer to the destination (and there may be free food as well!!!!) . For our agent this is analogous to predicting the reward provided by environment. This is called the **Reward Model**.

SNAKE



Grid for SNAKE

2. Eat cherries(pink circle to get bonus points).

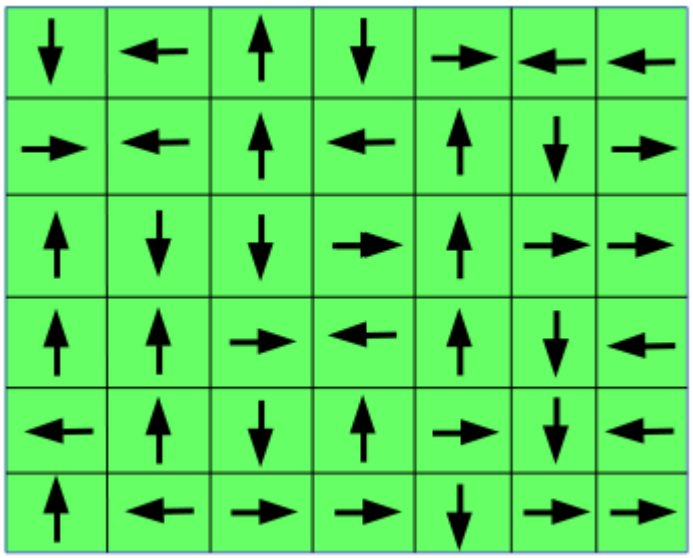
3. If you hit wall or eat yourself you die.

4. Controls are UP, DOWN, LEFT, RIGHT

**For the sake of convenience let us consider the environment to be a grid. Each box of the grid is the same size as each block of the snake. We will take the first block of snake as our reference for the snake.

Applying POLICY

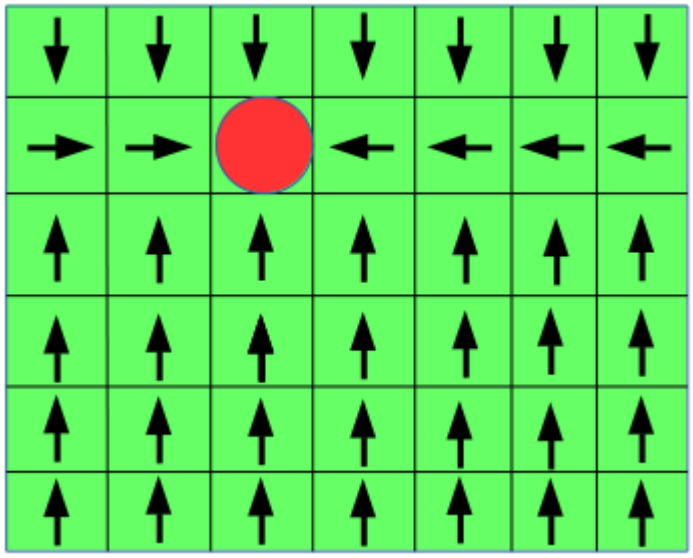
In case of Policy, we need to give the snake some behaviour. For that let us mark each block of the grid with a random direction. So, when the snake is in that block it will turn to the direction stated in the block.



Policy map for SNAKE

Now, it will follow the directions in the grid. Say, an apple appears. Then it may get to the apple (highly improbable) or it may keep moving randomly. We stop our snake when the first apple disappears(either the snake eats it or timeout condition). We check if the snake has eaten the apple or not, i.e., the goal state has been reached or not . If not then we change the directions in our model(change the policy) and retry

generalise and figures out how the policy should be changed to get maximum reward and reach goal state. So, after learning :



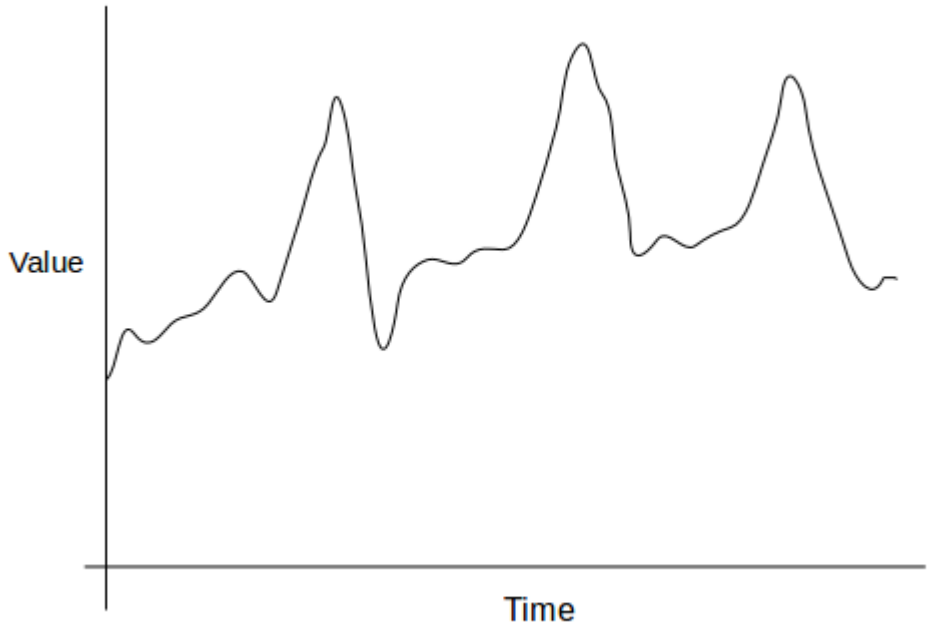
Policy map after training

Applying VALUE FUNCTION

In case of value function we will assign a value for every cell of the grid. Since, we want our snake to converge on the goal as fast as possible we assign a value of -1 for every block (if you don't understand why, please read the first article). Gradually the snake understands the longer it takes or the further it is from the goal the more negative reward it gets. So, it calculates the value of each cell accordingly. So, after learning :

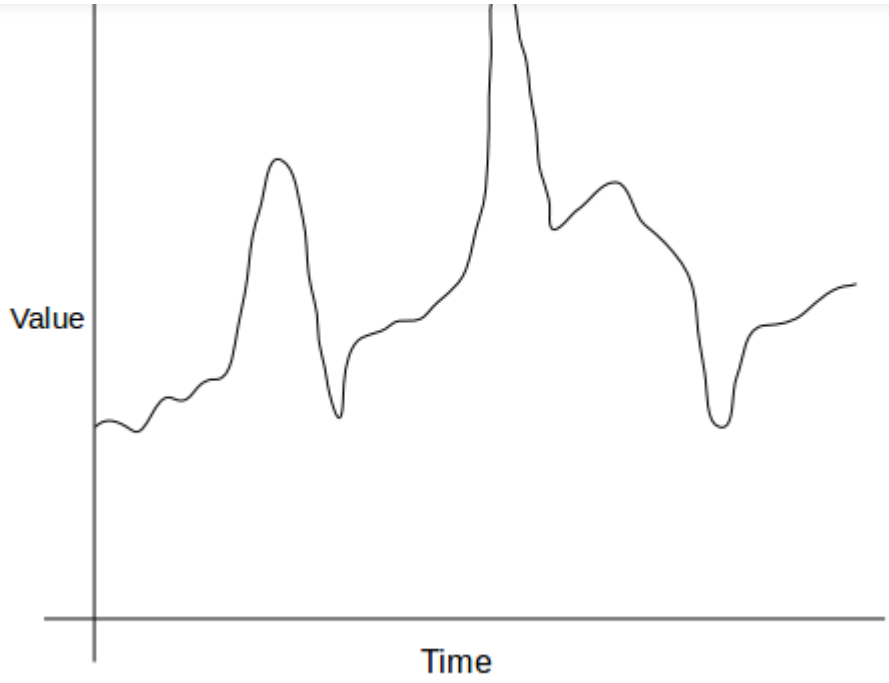
Now, let us visualise the value function :

say, we start from a square far away. As we get closer to the reward, the value function increases. Then when we finally get to the reward the value function is maximum(represented by peaks in the graph below). Then there is a time when the apple disappears (the trough/depression that follows the peak). So, we have a very low value function. The apple again appears and the cycle goes on like this. If we plot it against time we get :



Plot of Value Function with time, fig. 1

For the next visualisation, say a cherry appears. So, the value function rises to a huge peak (the peak in the middle). The snake captures it but still the value function doesn't fall as much . This is because of the presence of the apple. But as soon as the snake captures the apple, the value function drops drastically. Plot against time :



Plot of Value Function with time, fig. 2

Applying MODEL

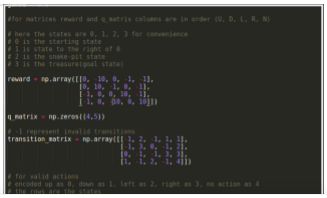
As I said earlier, model is just a prediction of the environment variables according to the agent. So, initially the snake doesn't have an idea about the surrounding. It takes a step and gets a reward of -1. It takes another step and still gets a reward of -1. So, it predicts that for the next step it takes it will also get a reward of -1. This would be the *transition model.* Same applies for the reward of the apple.

So, that would be all for this article. Hope you enjoyed reading it. For the next one we will start with small games on OpenAI Gym .If you liked it, please do recommend it. Until then, bye.

Link to Article 1:

Introduction to Q-Learning

Imagine yourself in a treasure hunt in a maze . The game is as follows :

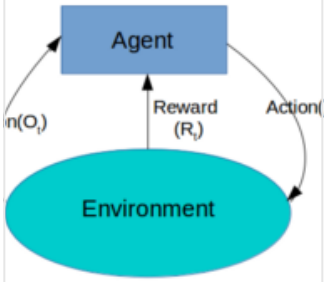


Link to Article 2:

The very basics of Reinforcement Learning

This article will be a brief diversion from my first post on Q Learning(link given at the end). I thought it would be...

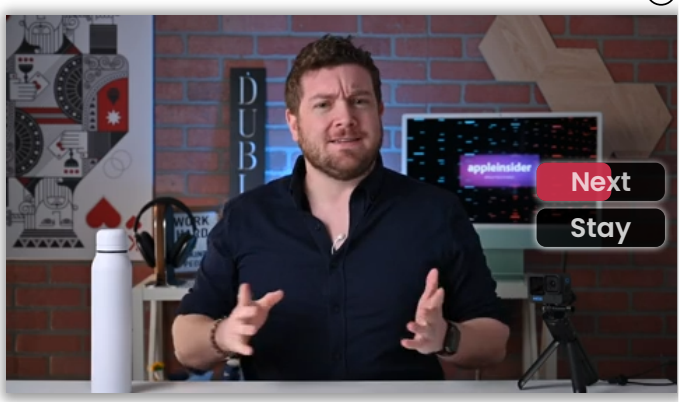
medium.com



Somethin

We've been



⇧ SCROLL TO TOP

Reinforcement Learning Tutorial

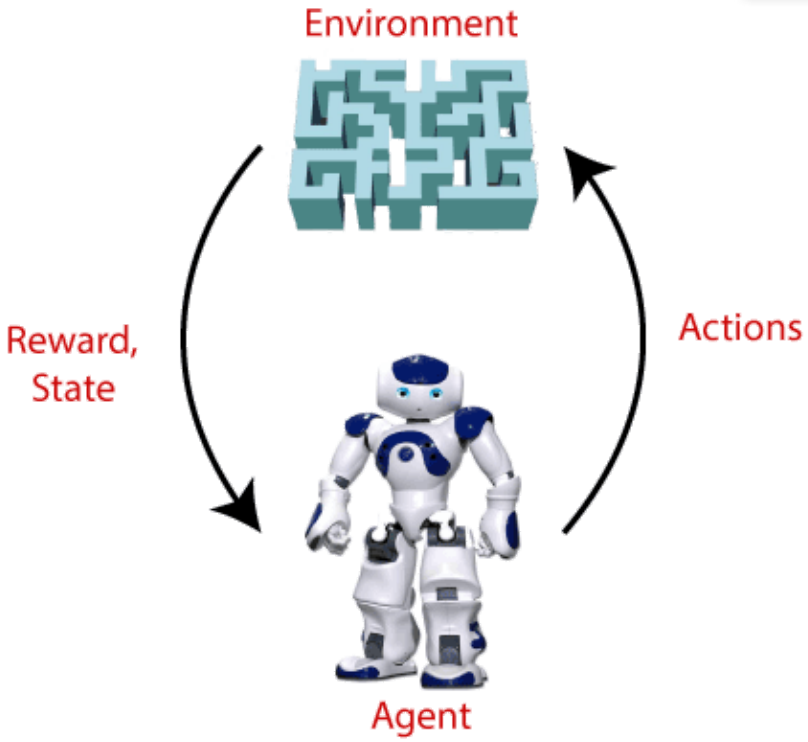
Our Reinforcement learning tutorial will give you a complete overview of reinforcement learning, including MDP and Q-learning. In RL tutorial, you will learn the below topics:

What is Reinforcement Learning?

- Reinforcement Learning is a feedback-based Ma
e in an environment by performing

For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, so the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that ***"Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that."*** How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.
- It is a core part of Artificial intelligence, and all AI agent works on the concept of reinforcement learning. Here we do not need to pre-program the agent, as it learns from its own experience without any human intervention.
- **Example:** Suppose there is an AI agent present within a maze environment, and his goal is to find the diamond. The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback.
- The agent continues doing these three things (**take action, change state/remain in the same state, and get feedback**), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.

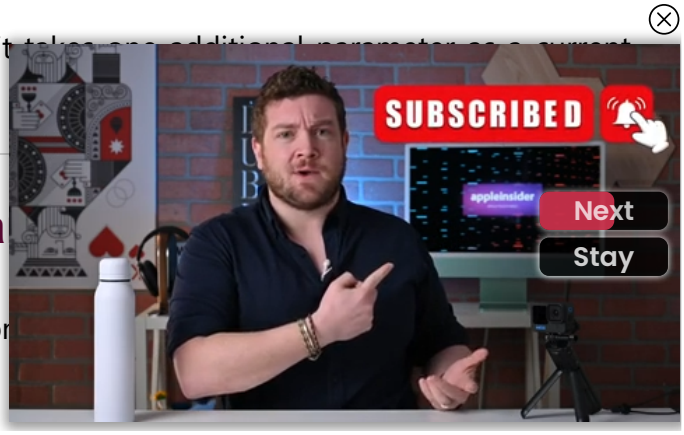


Terms used in Reinforcement Learning

- **Agent():** An entity that can perceive/explore the environment and act upon it.
- **Environment():** A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.
- **Action():** Actions are the moves taken by an agent within the environment.
- **State():** State is a situation returned by the environment after each action taken by the agent.
- **Reward():** A feedback returned to the agent from the environment to evaluate the action of the agent.
- **Policy():** Policy is a strategy applied by the agent for the next action based on the current state.
- **Value():** It is expected long-term retuned with the discount factor and opposite to the short-term reward.
- **Q-value():** It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Key Features of Reinforcement Lea

- In RL, the agent is not instructed about the enviro
- ne hit and trial process.



- The agent takes the next action and changes states according to the feedback of the previous action.
- The agent may get a delayed reward.
- The environment is stochastic, and the agent needs to explore it to reach to get the maximum positive rewards.

Approaches to implement Reinforcement Learning

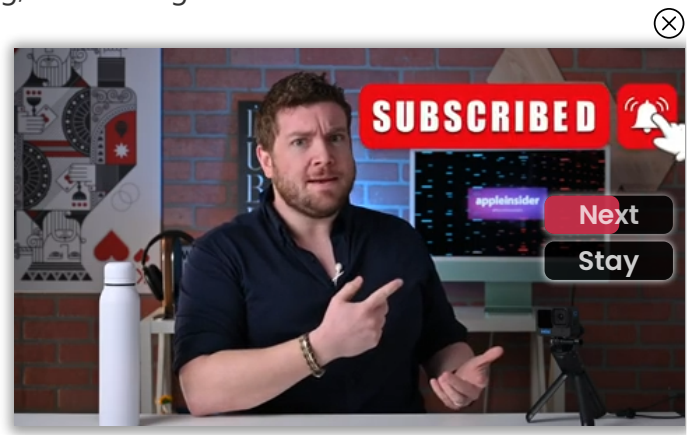
There are mainly three ways to implement reinforcement-learning in ML, which are:

1. **Value-based:**
   The value-based approach is about to find the optimal value function, which is the maximum value at a state under any policy. Therefore, the agent expects the long-term return at any state(s) under policy π.
2. **Policy-based:**
   Policy-based approach is to find the optimal policy for the maximum future rewards without using the value function. In this approach, the agent tries to apply such a policy that the action performed in each step helps to maximize the future reward.
   The policy-based approach has mainly two types of policy:
   - **Deterministic:** The same action is produced by the policy (π) at any state.
   - **Stochastic:** In this policy, probability determines the produced action.
3. **Model-based:** In the model-based approach, a virtual model is created for the environment, and the agent explores that environment to learn it. There is no particular solution or algorithm for this approach because the model representation is different for each environment.

Elements of Reinforcement Learning

There are four main elements of Reinforcement Learning, which are given below:

1. Policy
2. Reward Signal
3. Value Function
4. Model of the environment



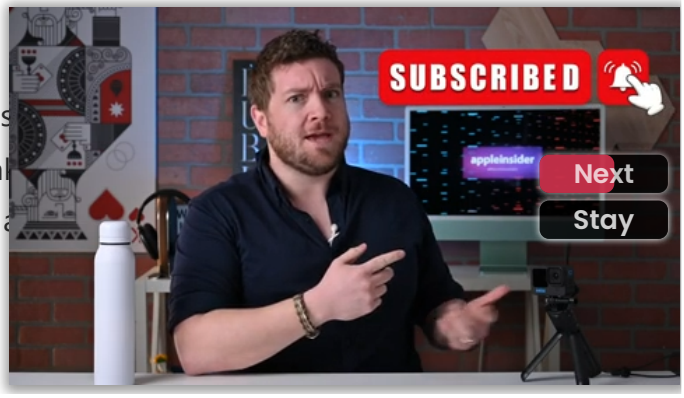
1) Policy: A policy can be defined as a way how an agent behaves at a given time. It maps the perceived states of the environment to the actions taken on those states. A policy is the core element of the RL as it alone can define the behavior of the agent. In some cases, it may be a simple function or a lookup table, whereas, for other cases, it may involve general computation as a search process. It could be deterministic or a stochastic policy:

For deterministic policy: a = π(s)
For stochastic policy: π(a | s) = P[At =a | St = s]

2) Reward Signal: The goal of reinforcement learning is defined by the reward signal. At each state, the environment sends an immediate signal to the learning agent, and this signal is known as a **reward signal**. These rewards are given according to the good and bad actions taken by the agent. The agent's main objective is to maximize the total number of rewards for good actions. The reward signal can change the policy, such as if an action selected by the agent leads to low reward, then the policy may change to select other actions in the future.

3) Value Function: The value function gives information about how good the situation and action are and how much reward an agent can expect. A reward indicates the **immediate signal for each good and bad action**, whereas a value function specifies **the good state and action for the future**. The value function depends on the reward as, without reward, there could be no value. The goal of estimating values is to achieve more rewards.

4) Model: The last element of reinforcement learning is
environment. With the help of the model, one can mak
behave. Such as, if a state and an action are given, then a



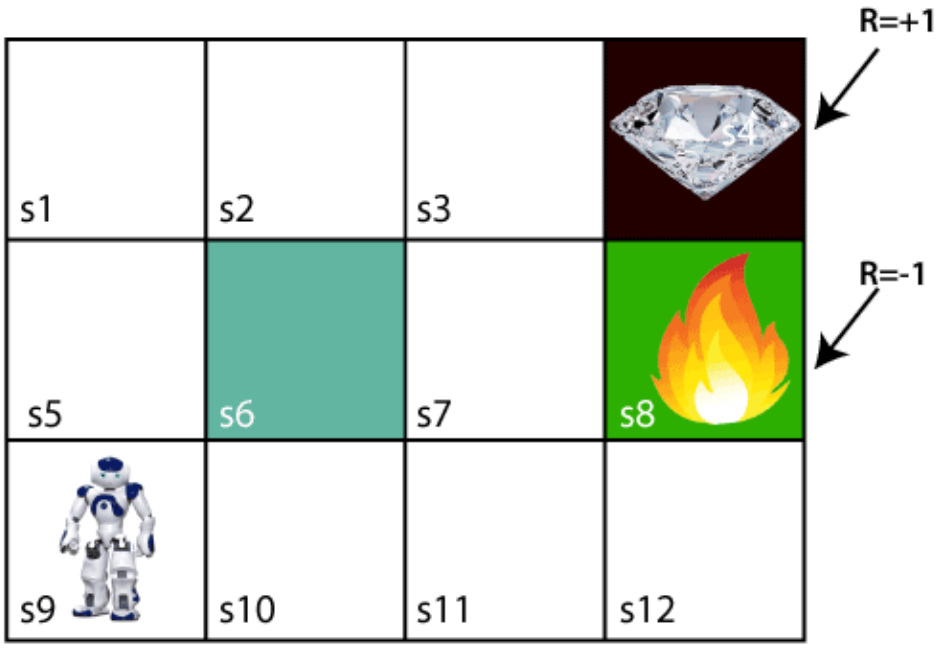
The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the RL problems **with the help of the model** are termed as the **model-based approach**. Comparatively, an approach **without using a model** is called a **model-free approach**.

How does Reinforcement Learning Work?

To understand the working process of the RL, we need to consider two main things:

- **Environment:** It can be anything such as a room, maze, football ground, etc.
- **Agent:** An intelligent agent such as AI robot.

Let's take an example of a maze environment that the agent needs to explore. Consider the below image:



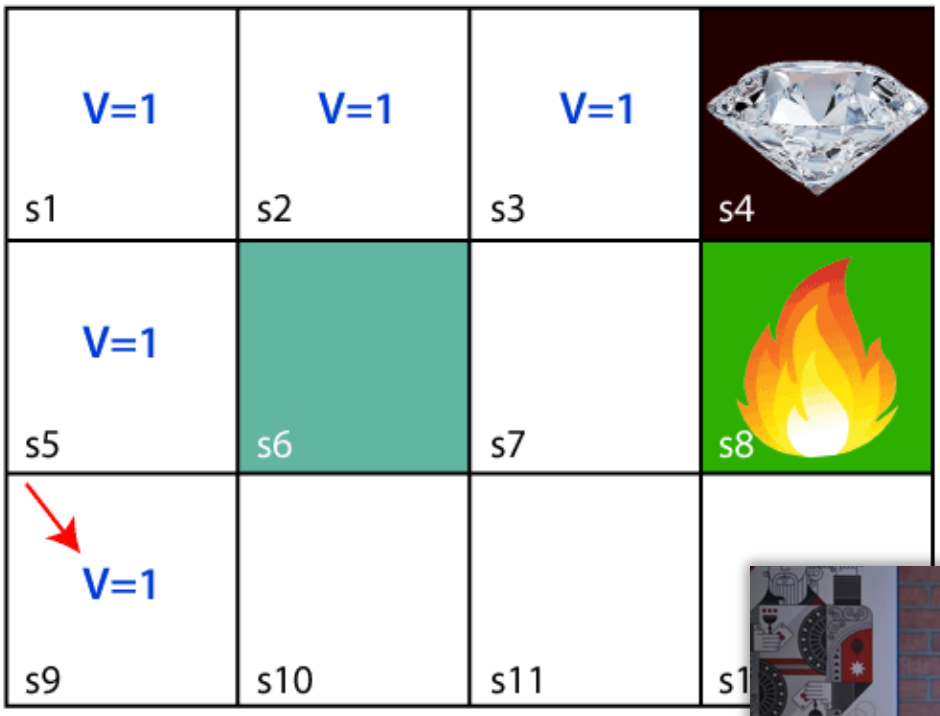
Lorem ipsum

In the above image, the agent is at the very first block block, which is a **wall**, S_8 a **fire pit**, and S_4 a **diamond bl**

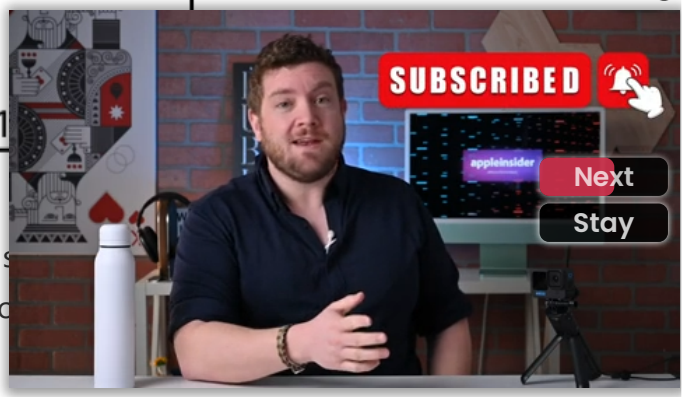
The agent cannot cross the S_6 block, as it is a solid wall. If the agent reaches the S_4 block, then get the **+1 reward;** if it reaches the fire pit, then gets **-1 reward point**. It can take four actions**: move up, move down, move left, and move right.**

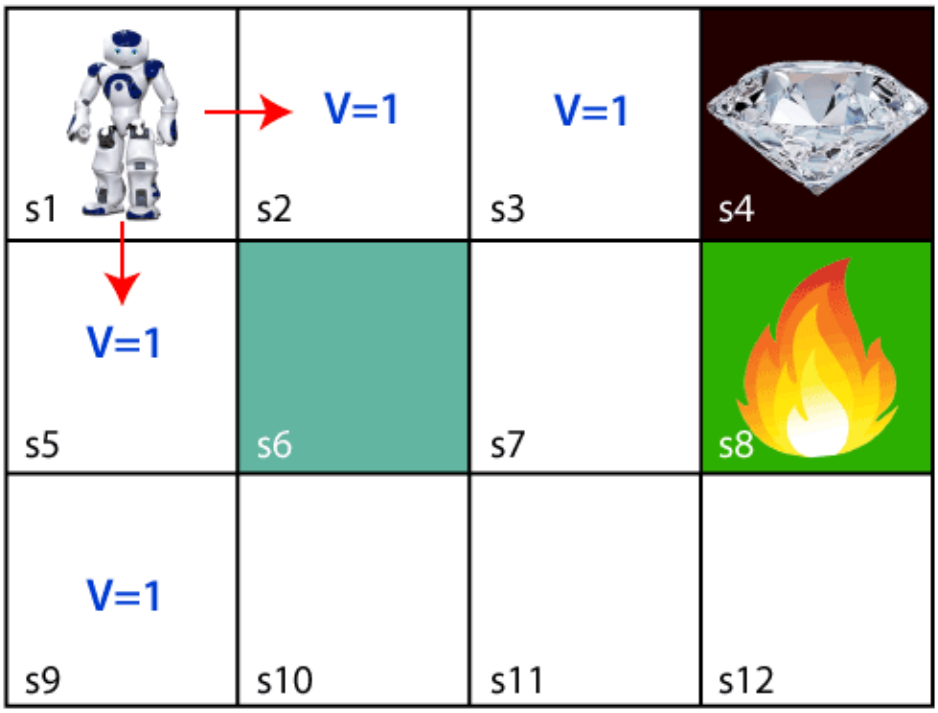
The agent can take any path to reach to the final point, but he needs to make it in possible fewer steps. Suppose the agent considers the path **S9-S5-S1-S2-S3**, so he will get the +1-reward point.

The agent will try to remember the preceding steps that it has taken to reach the final step. To memorize the steps, it assigns 1 value to each previous step. Consider the below step:



Now, the agent has successfully stored the previous s
he agent do if he starts moving fr
elow diagram:

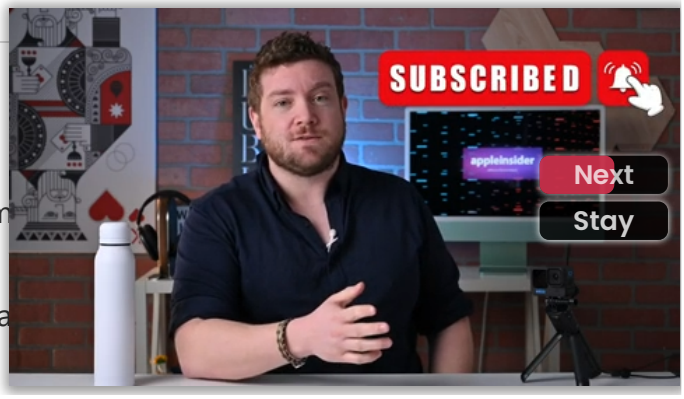




It will be a difficult condition for the agent whether he should go up or down as each block has the same value. So, the above approach is not suitable for the agent to reach the destination. Hence to solve the problem, we will use the **Bellman equation**, which is the main concept behind reinforcement learning.

The Bellman Equation

The Bellman equation was introduced by the Mathem
1953, and hence it is called as a Bellman equation. It
e values of a decision problem a



It is a way of calculating the value functions in dynamic programming or environment that leads to modern reinforcement learning.

The key-elements used in Bellman equations are:

- Action performed by the agent is referred to as "a"
- State occurred by performing the action is "s."
- The reward/feedback obtained for each good and bad action is "R."
- A discount factor is Gamma "γ."

The Bellman equation can be written as:

$$V(s) = \max [R(s,a) + \gamma V(s`)]$$

Where,

V(s)= value calculated at a particular point.

R(s,a) = Reward at a particular state s by performing an action.

γ = Discount factor

V(s`) = The value at the previous state.

In the above equation, we are taking the max of the complete values because the agent tries to find the optimal solution always.

So now, using the Bellman equation, we will find value at each state of the given environment. We will start from the block, which is next to the target block.

For 1st block:

γV(s`)], here V(s')= 0 because there

V(s3)= max[R(s,a)]=> V(s3)= max[1]=> **V(s3)= 1.**

For 2nd block:

V(s2) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 1, and R(s, a)= 0, because there is no reward at this state.

V(s2)= max[0.9(1)]=> V(s)= max[0.9]=> **V(s2) =0.9**

For 3rd block:

V(s1) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.9, and R(s, a)= 0, because there is no reward at this state also.

V(s1)= max[0.9(0.9)]=> V(s3)= max[0.81]=> **V(s1) =0.81**

For 4th block:

V(s5) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.81, and R(s, a)= 0, because there is no reward at this state also.

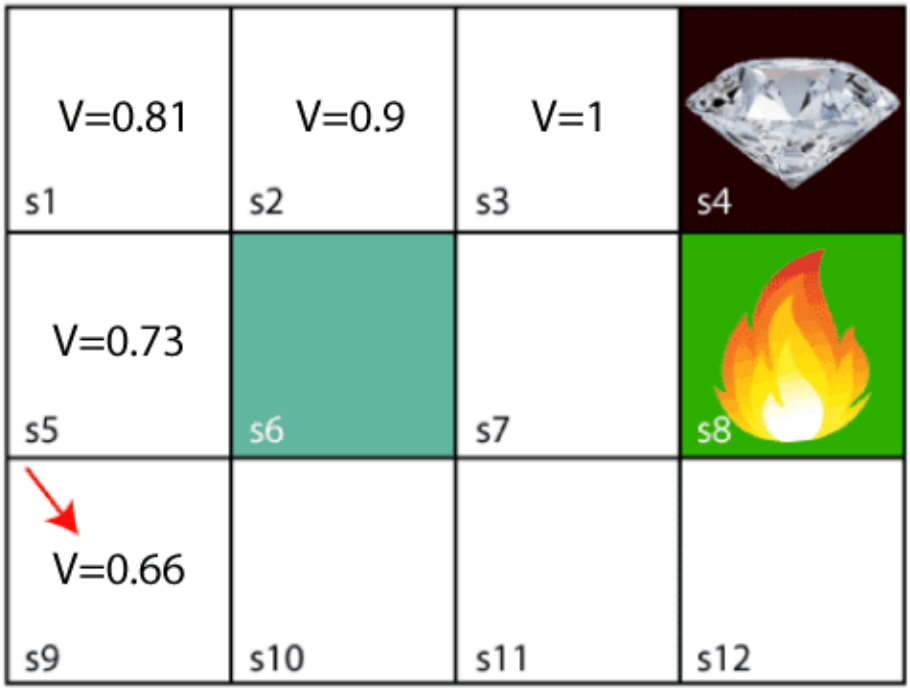
V(s5)= max[0.9(0.81)]=> V(s5)= max[0.81]=> **V(s5) =0.73**

For 5th block:

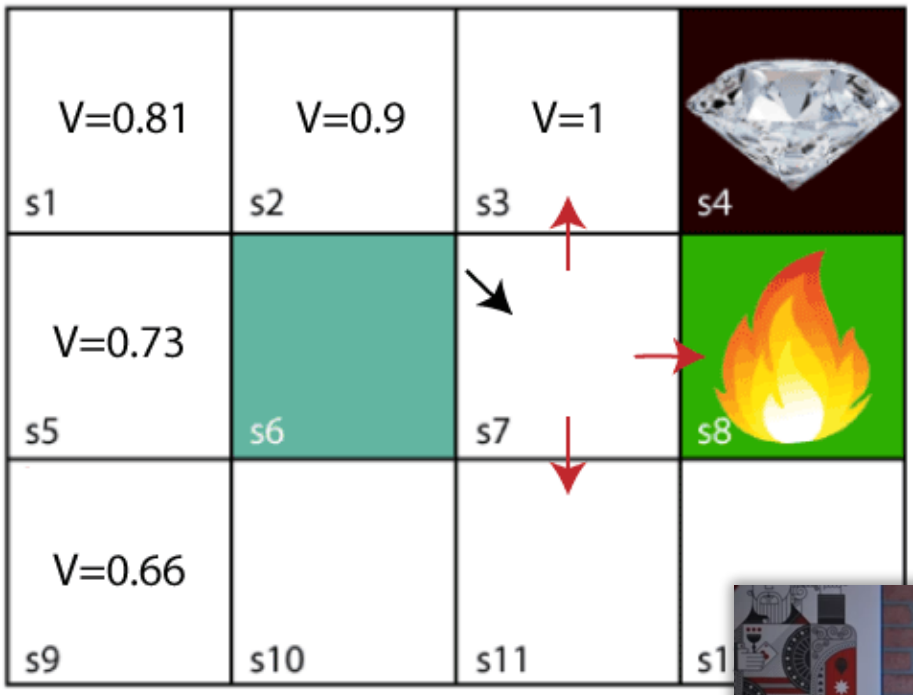
V(s9) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.73, and R(s, a)= 0, because there is no reward at this state also.

V(s9)= max[0.9(0.73)]=> V(s4)= max[0.81]=> **V(s4) =0.66**

Consider the below image:

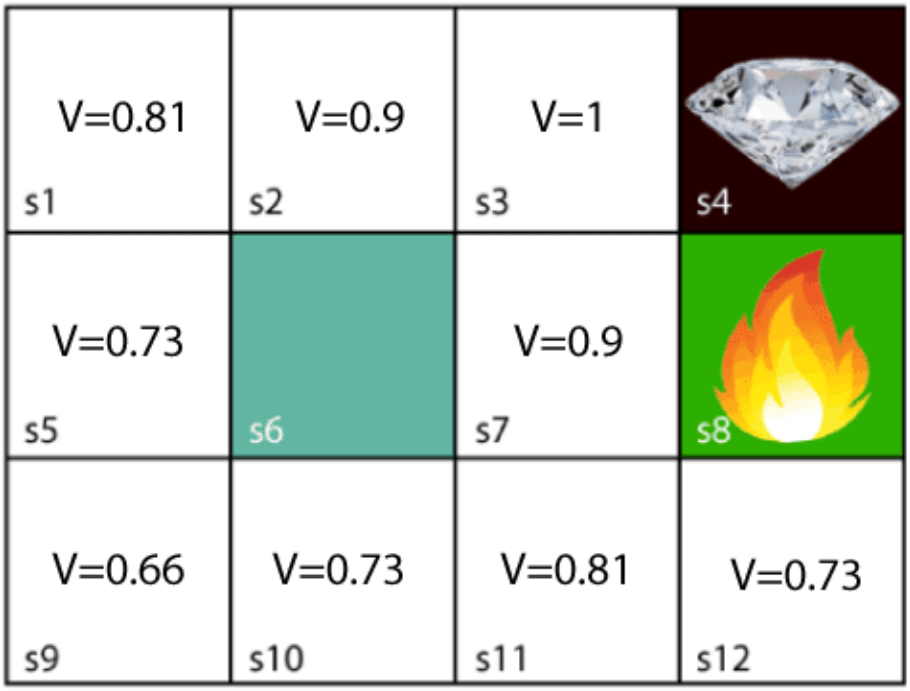


Now, we will move further to the 6th block, and here agent may change the route because it always tries to find the optimal path. So now, let's consider from the block next to the fire pit.



Now, the agent has three options to move; if he moves moves to the fire pit, then he will get the -1 reward. Bu to upwards only. The complete blo

Consider the below image:



Types of Reinforcement learning

There are mainly two types of reinforcement learning, which are:

- **Positive Reinforcement**
- **Negative Reinforcement**

Positive Reinforcement:

The positive reinforcement learning means adding something to increase the tendency that expected behavior would occur again. It impacts positively on the behavior of the agent and increases the strength of the behavior.

This type of reinforcement can sustain the changes for a long time, but too much positive reinforcement may lead to an overload of states that can

Negative Reinforcement:

The negative reinforcement learning is opposite to th tendency that the specific behavior will occur again by a

It can be more effective than the positive reinforcement depending on situation and behavior, but it provides reinforcement only to meet minimum behavior.

How to represent the agent state?

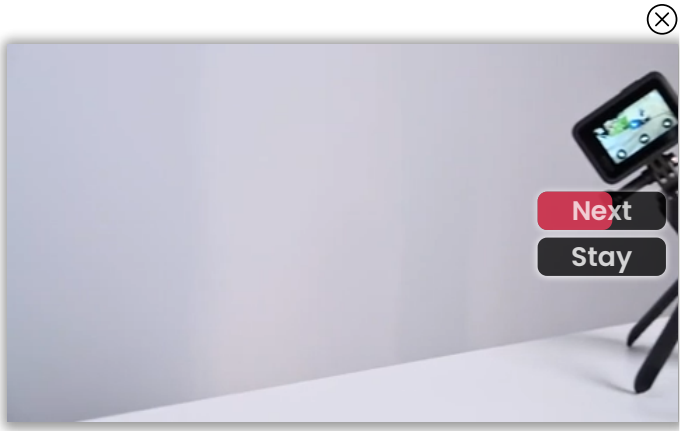
We can represent the agent state using the **Markov State** that contains all the required information from the history. The State St is Markov state if it follows the given condition:

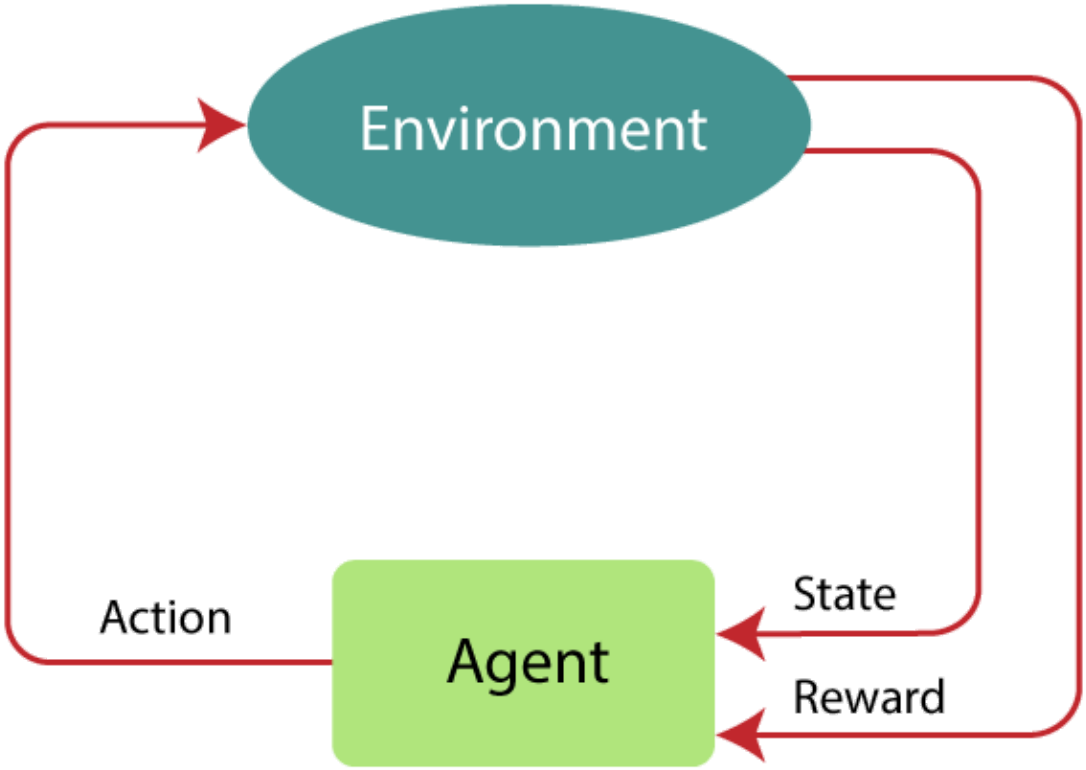
$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1, \ldots\ldots, S_t]$$

The Markov state follows the **Markov property**, which says that the future is independent of the past and can only be defined with the present. The RL works on fully observable environments, where the agent can observe the environment and act for the new state. The complete process is known as Markov Decision process, which is explained below:

Markov Decision Process

Markov Decision Process or MDP, is used to **formalize the reinforcement learning problems**. If the environment is completely observable, then its dynamic can be modeled as a **Markov Process**. In MDP, the agent constantly interacts with the environment and performs actions; at each action, the environment responds and generates a new state.





MDP is used to describe the environment for the RL, and almost all the RL problem can be formalized using MDP.

MDP contains a tuple of four elements (S, A, P_a, R_a):

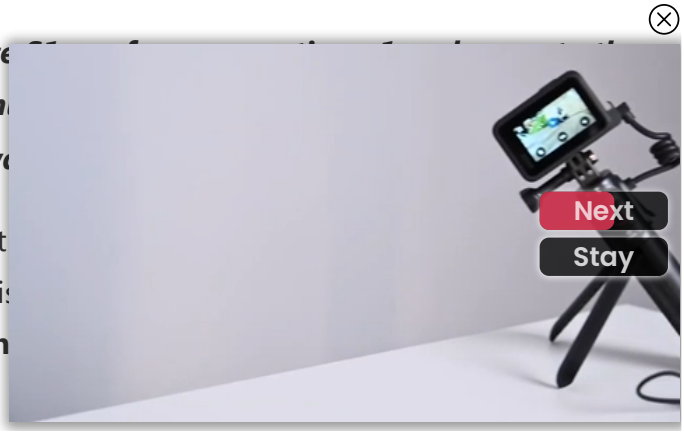
- A set of finite States S
- A set of finite Actions A
- Rewards received after transitioning from state S to state S', due to action a.
- Probability P_a.

MDP uses **Markov property**, and to better understand the MDP, we need to learn about it.

Markov Property:

It says that ***"If the agent is present in the current state***
state s2, then the state transition from s1 to s2 on
action and states do not depend on past actions, rewe

Or, in other words, as per Markov Property, the current
action or state. Hence, MDP is an RL problem that satis
nly focus on the current state an



Finite MDP:

A finite MDP is when there are finite states, finite rewards, and finite actions. In RL, we consider only the finite MDP.

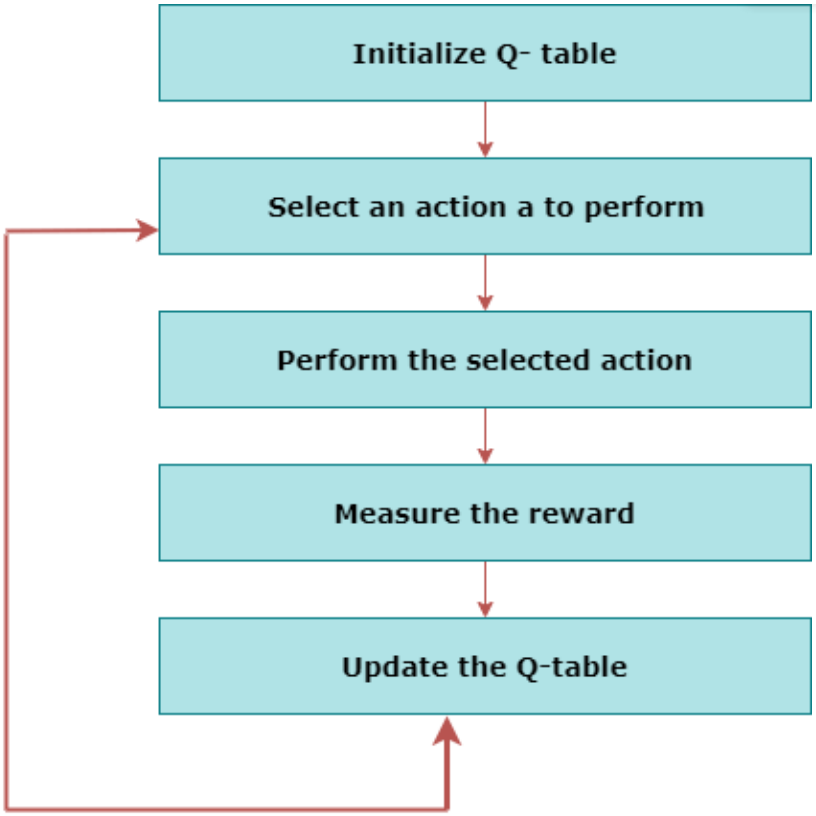
Markov Process:

Markov Process is a memoryless process with a sequence of random states S_1, S_2, ..., S_t that uses the Markov Property. Markov process is also known as Markov chain, which is a tuple (S, P) on state S and transition function P. These two components (S and P) can define the dynamics of the system.

Reinforcement Learning Algorithms

Reinforcement learning algorithms are mainly used in AI applications and gaming applications. The main used algorithms are:

- **Q-Learning:**
  - Q-learning is an **Off policy RL algorithm**, which is used for the temporal difference Learning. The temporal difference learning methods are the way of comparing temporally successive predictions.
  - It learns the value function Q (S, a), which means how good to take action "**a**" at a particular state "**s**."
  - The below flowchart explains the working of Q- learning:



- **State Action Reward State action (SARSA):**
  - SARSA stands for **State Action Reward State action**, which is an **on-policy** temporal difference learning method. The on-policy control method selects the action for each state while learning using a specific policy.
  - The goal of SARSA is to calculate the **Q π (s, a) for the selected current policy π and all pairs of (s-a).**
  - The main difference between Q-learning and SARSA algorithms is that **unlike Q-learning, the maximum reward for the next state is not required for updating the Q-value in the table.**
  - In SARSA, new action and reward are selected using the same policy, which has determined the original action.
  - The SARSA is named because it uses the quintuple **Q(s, a, r, s', a').** Where,

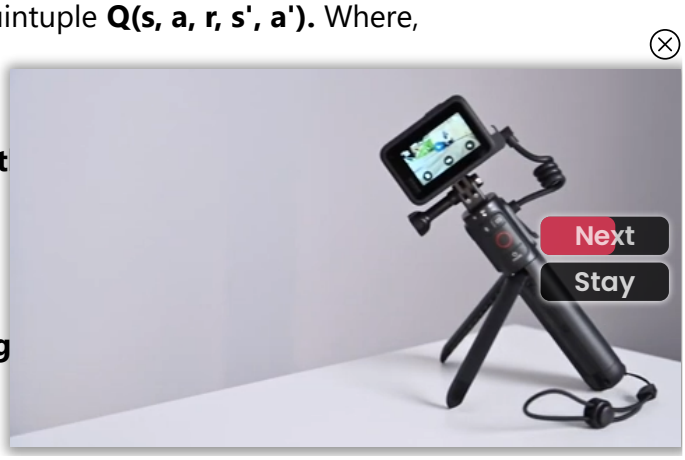
    s: original state

    a: Original action

    r: reward observed while following t

    s' and a': New state, action pair.

- **Deep Q Neural Network (DQN):**
  - As the name suggests, DQN is a **Q-learning**



- For a big state space environment, it will be a challenging and complex task to define and update a Q-table.
- To solve such an issue, we can use a DQN algorithm. Where, instead of defining a Q-table, neural network approximates the Q-values for each action and state.

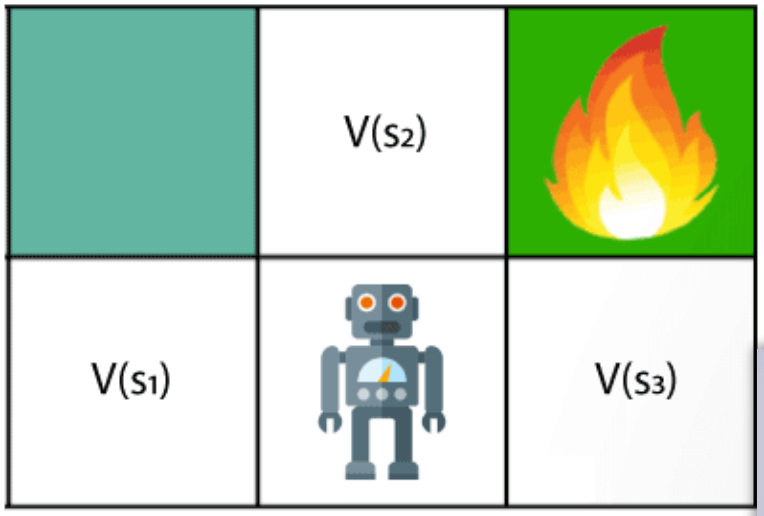
Now, we will expand the Q-learning.

Q-Learning Explanation:

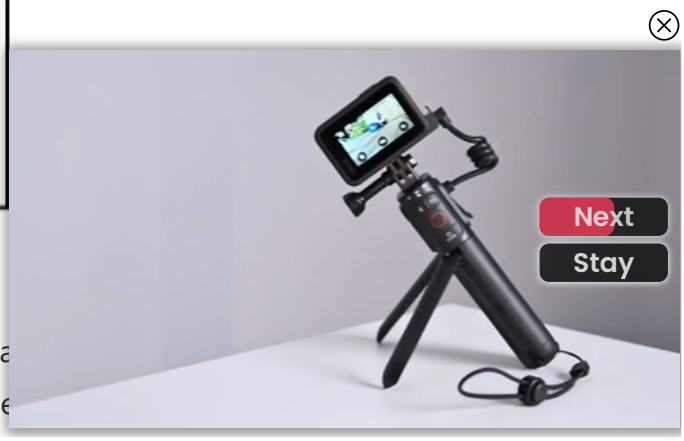
- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- **The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.**
- It is an **off-policy RL** that attempts to find the best action to take at a current state.
- The goal of the agent in Q-learning is to maximize the value of Q.
- The value of Q-learning can be derived from the Bellman equation. Consider the Bellman equation given below:

$$V(s) = \max [R(s,a) + \gamma \sum_{s'} P(s,a,s')V(s`)]$$

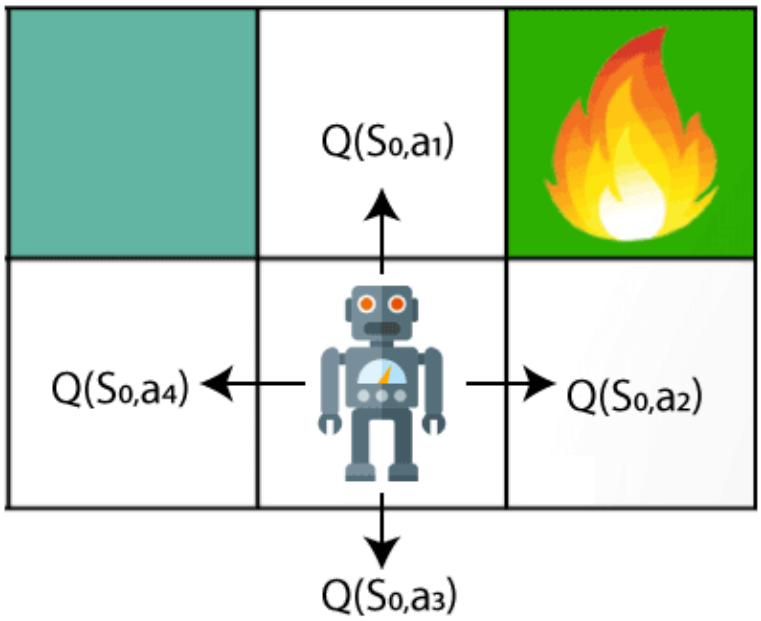
In the equation, we have various components, including reward, discount factor (γ), probability, and end states s'. But there is no any Q-value is given so first consider the below image:



In the above image, we can see there is an agent who
t only cares for the current state a
Right), so he needs to decide whe



take a move as per probability bases and changes the state. But if we want some exact moves, so for this, we need to make some changes in terms of Q-value. Consider the below image:



Q- represents the quality of the actions at each state. So instead of using a value at each state, we will use a pair of state and action, i.e., Q(s, a). Q-value specifies that which action is more lubricative than others, and according to the best Q-value, the agent takes his next move. The Bellman equation can be used for deriving the Q-value.

To perform any action, the agent will get a reward R(s, a), and also he will end up on a certain state, so the Q -value equation will be:

$$Q(S, a)= R(s, a)+ \gamma\sum_{s'} P(s, a, s')V(s`)$$

Hence, we can say that, ***V(s) = max [Q(s, a)]***

$$Q(S, a)= R(s, a)+ \gamma\sum_{s'}(P(s, a, s')\max Q(s',a`))$$

The above formula is used to estimate the Q-values in Q-Learning

What is 'Q' in Q-learning?

The Q stands for **quality** in **Q-learning**, which means it agent.

A Q-table or matrix is created while performing the Q-learning. The table follows the state and action pair, i.e., [s, a], and initializes the values to zero. After each action, the table is updated, and the q-values are stored within the table.

The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

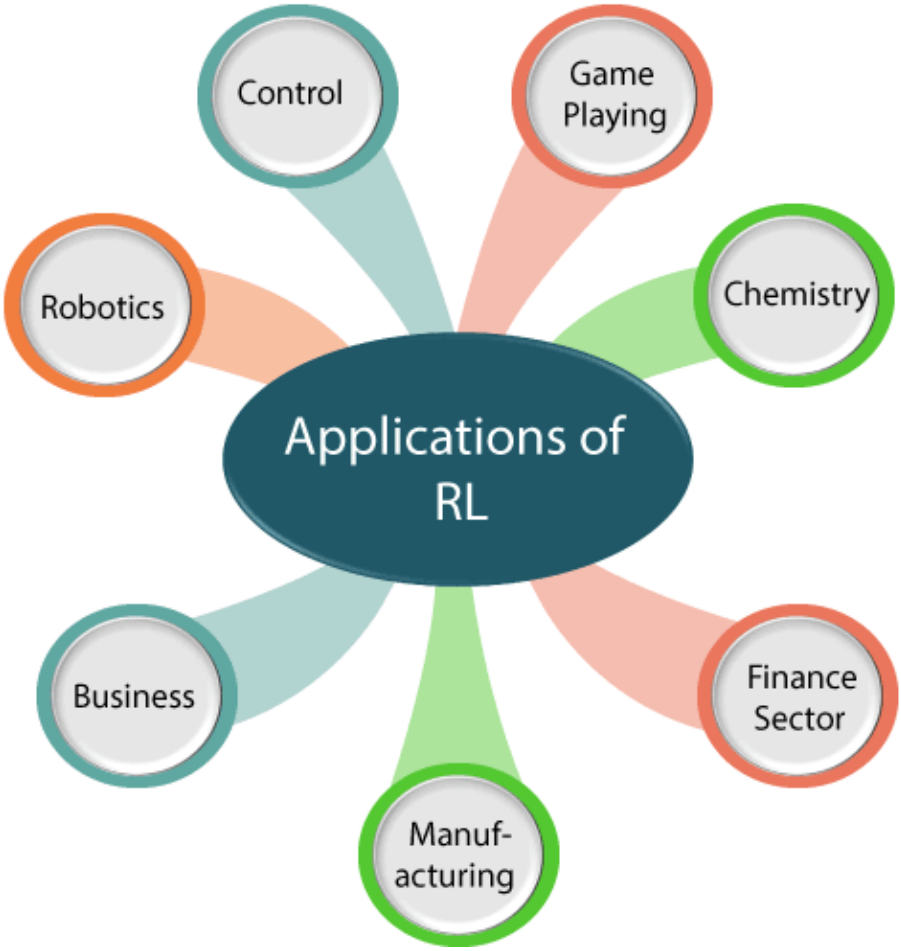
Difference between Reinforcement Learning and Supervised Learning

The Reinforcement Learning and Supervised Learning both are the part of machine learning, but both types of learnings are far opposite to each other. The RL agents interact with the environment, explore it, take action, and get rewarded. Whereas supervised learning algorithms learn from the labeled dataset and, on the basis of the training, predict the output.

The difference table between RL and Supervised learning is given below:

Reinforcement Learning	Supervised Learning
RL works by interacting with the environment.	Supervised learning works on the existing dataset.
The RL algorithm works like the human brain works when making some decisions.	Supervised Learning works as when a human learns things in the supervision of a guide.
There is no labeled dataset is present	The labeled dataset is present.
No previous training is provided to the learning agent.	Training is provided to the algorithm so that it can predict the output.
RL helps to take decisions sequentially.	In Supervised learning, decisions are made when input is given.

Reinforcement Learning Applications



1. **Robotics:**
    a. RL is used in **Robot navigation, Robo-soccer, walking, juggling**, etc.
2. **Control:**
    a. RL can be used for **adaptive control** such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.
3. **Game Playing:**
    a. RL can be used in **Game playing** such as tic-tac-toe, chess, etc.
4. **Chemistry:**
    a. RL can be used for optimizing the chemical reactions.
5. **Business:**
    a. RL is now used for business strategy plannin
6. **Manufacturing:**
    a. In various automobile manufacturing com
    to pick goods and put them in som

7. **Finance Sector:**
    a. The RL is currently used in the finance sector for evaluating trading strategies.

Conclusion:

From the above discussion, we can say that Reinforcement Learning is one of the most interesting and useful parts of Machine learning. In RL, the agent explores the environment by exploring it without any human intervention. It is the main learning algorithm that is used in Artificial Intelligence. But there are some cases where it should not be used, such as if you have enough data to solve the problem, then other ML algorithms can be used more efficiently. The main issue with the RL algorithm is that some of the parameters may affect the speed of the learning, such as delayed feedback.

Help Others, Please Share

Learn Latest Tutorials

Preparation

Aptitude

Aptitude

Logical Reasoning

Reasoning

Verbal Ability

Verbal Ability

Interview Questions

Interview Questions

Company Interview Questions

Company Questions

Trending Technologies

Artificial Intelligence Tutorial

Artificial Intelligence

AWS Tutorial

AWS

Selenium tutorial

Selenium

Cloud Computing tutorial

Cloud Computing

Hadoop tutorial

Hadoop

ReactJS Tutorial

ReactJS

Data Science Tutorial

Data Science

Angular 7 Tutorial

Angular 7

Blockchain Tutorial

Blockchain

Git Tutorial

Git

Machine Learning Tutorial

Machine Learning

DevOps Tutorial

DevOps

⇧ SCROLL TO TOP

B.Tech / MCA

DBMS tutorial DBMS	Data Structures tutorial Data Structures	DAA tutorial DAA	Operating System tutorial Operating System
Computer Network tutorial Computer Network	Compiler Design tutorial Compiler Design	Computer Organization and Architecture Computer Organization	Discrete Mathematics Tutorial Discrete Mathematics
Ethical Hacking Tutorial Ethical Hacking	Computer Graphics Tutorial Computer Graphics	Software Engineering Tutorial Software Engineering	html tutorial Web Technology
Cyber Security tutorial Cyber Security	Automata Tutorial Automata	C Language tutorial C Progra	C++ tutorial C++
Java tutorial Java	.Net Framework tutorial .Net	Pythor Pyth	


Control Systems tutorial

Control System


Data Mining Tutorial

Data Mining


Data Warehouse Tutorial

Data Warehouse

⇧ SCROLL TO TOP