UNIT 3

REINFORCEMENT LEARNING

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. 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.

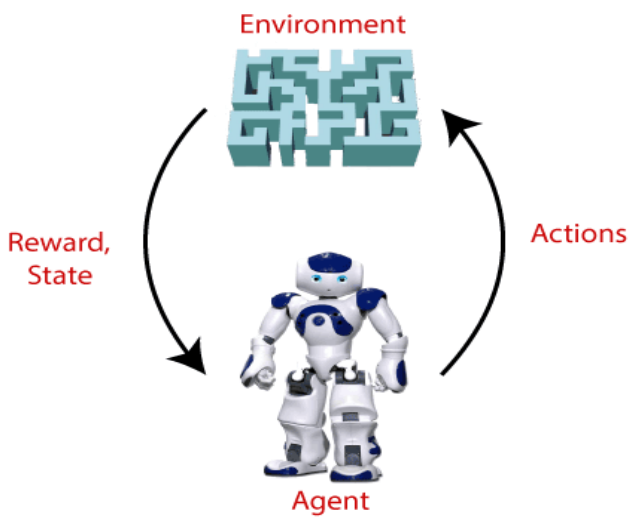
It is a core part of Artificial intelligence

a. , and all AI agent
b. 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 Learning

- In RL, the agent is not instructed about the environment and what actions need to be taken.
- It is based on the 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:

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.

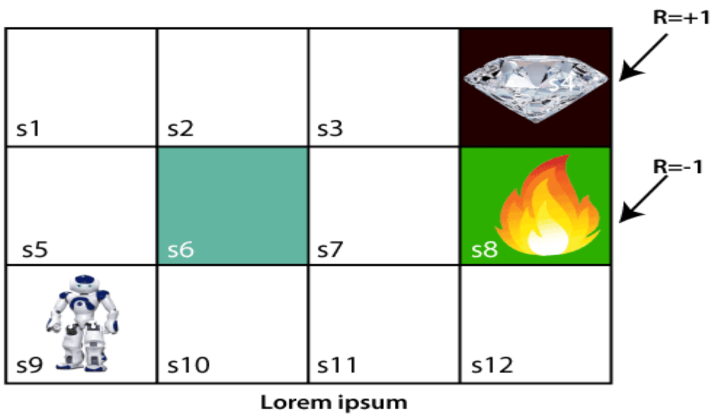
4) Model: The last element of reinforcement learning is the model, which mimics the behavior of the environment. With the help of the model, one can make inferences about how the environment will behave. Such as, if a state and an action are given, then a model can predict the next state and reward.

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.

WORKING OF REINFORCEMENT LEARNING:

- **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:

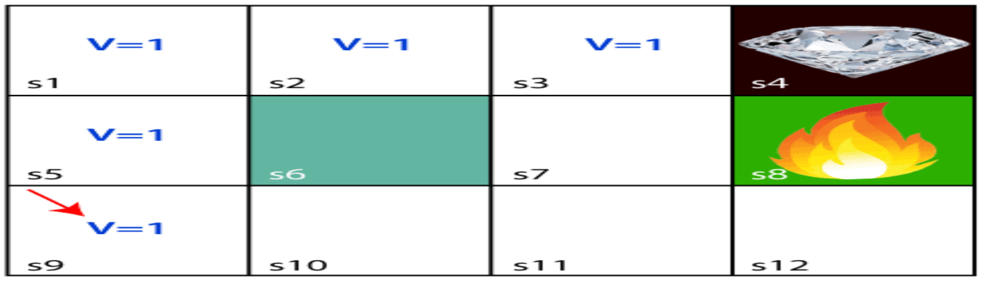


In the above image, the agent is at the very first block of the maze. The maze is consisting of an S_6 block, which is a **wall**, S_8 a **fire pit**, and S_4 a **diamond block.**

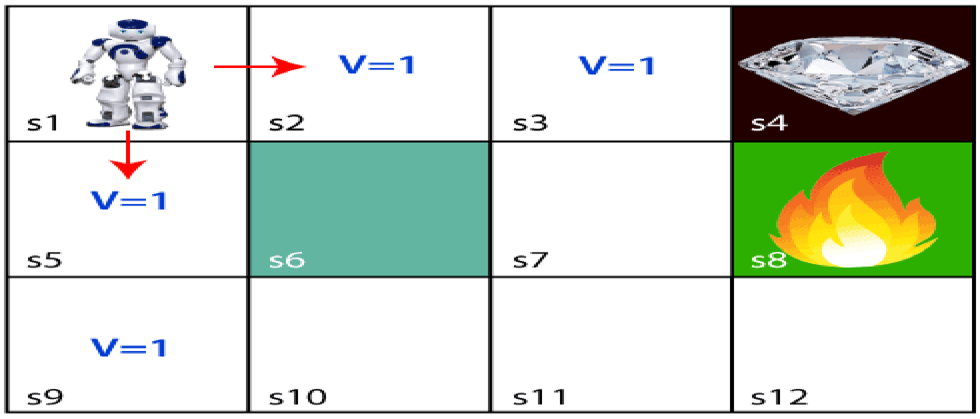
The agent cannot cross the S_6 block, as it is a solid wall. If the agent reaches the S_4 block, then get the **+1 reward;** if it reaches the fire pit, then gets **-1 reward point**. It can take four actions**: move up, move down, move left, and move right.**

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 steps assigning the 1 value to each previous block. But what will the agent do if he starts moving from the block, which has 1 value block on both sides? Consider the below 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 Mathematician **Richard Ernest Bellman in the year 1953**, and hence it is called as a Bellman equation. It is associated with dynamic programming and used to calculate the values of a decision problem at a certain point by including the values of previous states.

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:

V(s3) = max [R(s,a) + γV(s`)], here V(s')= 0 because there is no further state to move.

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.

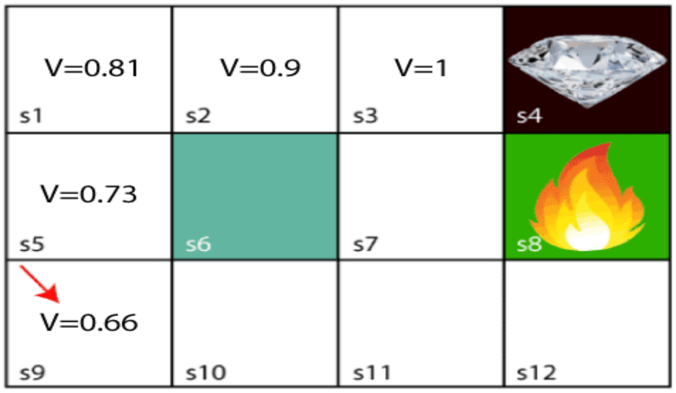
V(s5)= max[0.9(0.81)]=> V(s5)= max[0.81]=> **V(s5) =0.73**

For 5th block:

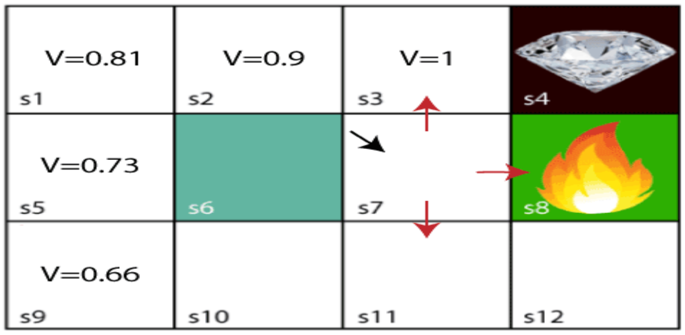
V(s9) = max [R(s,a) + γV(s`)], here γ= 0.9(lets), V(s')= 0.73, and R(s, a)= 0, because there is no reward at this state also.

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

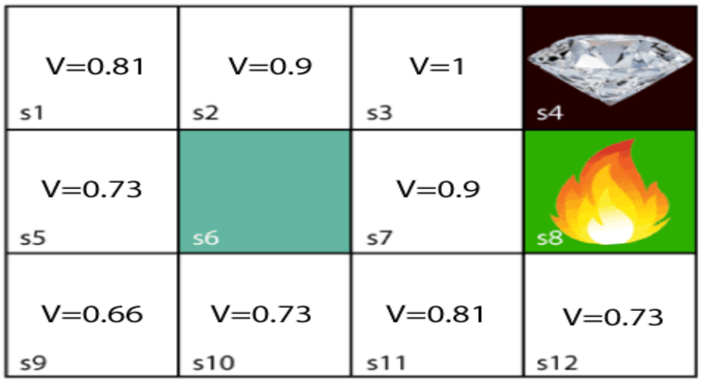
Consider the below image:



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 to the blue box, then he will feel a bump if he moves to the fire pit, then he will get the -1 reward. But here we are taking only positive rewards, so for this, he will move to upwards only. The complete block values will be calculated using this formula. 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 reduce the consequences.

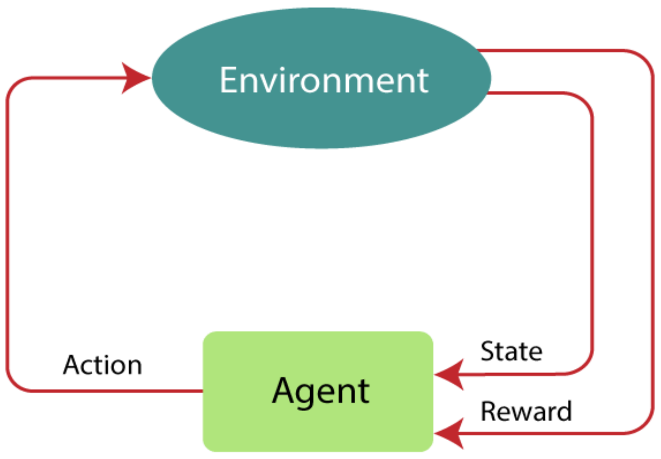
Negative Reinforcement:

The negative reinforcement learning is opposite to the positive reinforcement as it increases the tendency that the specific behavior will occur again by avoiding the negative condition.

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

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 S1, performs an action a1 and move to the state s2, then the state transition from s1 to s2 only depends on the current state and future action and states do not depend on past actions, rewards, or states."

Or, in other words, as per Markov Property, the current state transition does not depend on any past action or state. Hence, MDP is an RL problem that satisfies the Markov property. Such as in a Chess game, the players only focus on the current state and do not need to remember past actions or states**.**

Finite MDP:

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

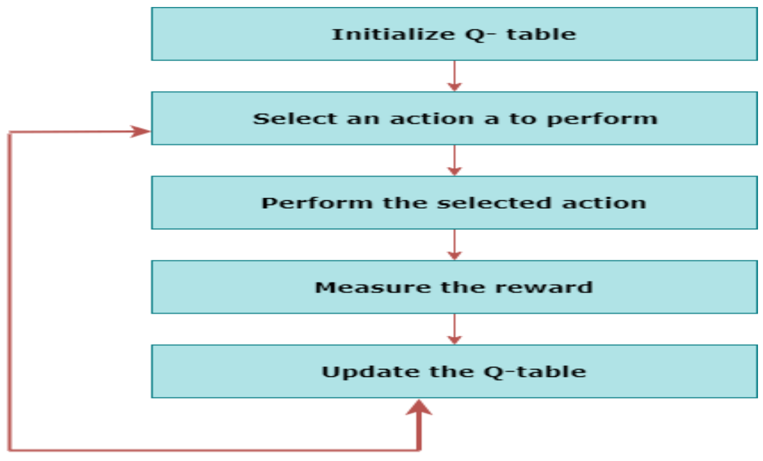
Markov Process:

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:originastate
  a:Originalaction
  r:rewardobservedwhilefollowingthestates
  s' and a': New state, action pair.

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

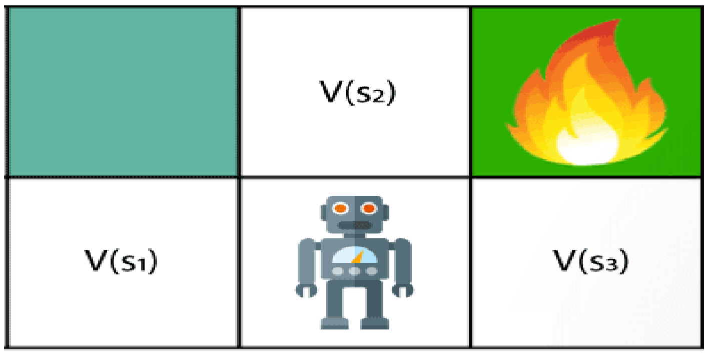
Now, we will expand the Q-learning.

Q-Learning Explanation:

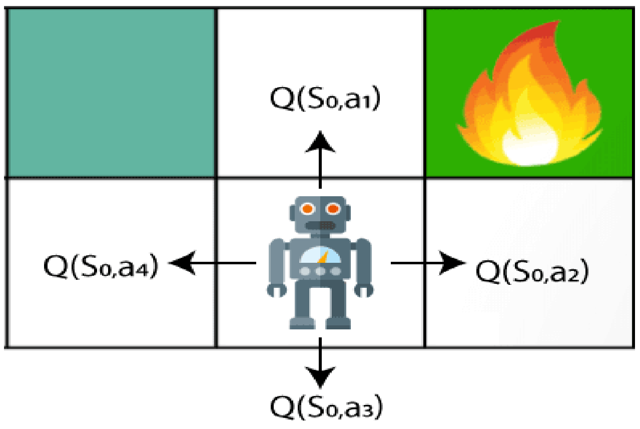
- Q-learning is a popular model-free reinforcement learning algorithm based on the Bellman equation.
- **The main objective of Q-learning is to learn the policy which can inform the agent that what actions should be taken for maximizing the reward under what circumstances.**
- It is an **off-policy RL** 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 has three values options, $V(s_1)$, $V(s_2)$, $V(s_3)$. As this is MDP, so agent only cares for the current state and the future state. The agent can go to any direction (Up, Left, or Right), so he needs to decide where to go for the optimal path. Here agent will 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 specifies the quality of an action taken by the agent.

Q-table:

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.

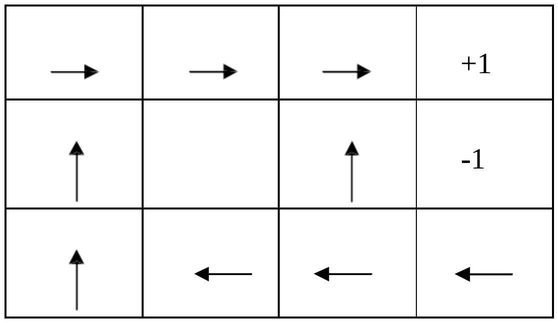
The RL agent uses this Q-table as a reference table to select the best action based on the q-values.

Passive Reinforcement Learning

1. In this learning, the agent's policy is fixed and the task is to learn the utilities of states.
2. It could also involve learning a model of the environment.
3. In passive learning, the agent's policy is fixed (i.e.) in state s, it always executes the action
   i. (s).

Its goal is simply to learn the utility function U (s).

4. For example: - Consider the 4 x 3 world.
5. The following figure shows the policy for that world.



- The following figure shows the corresponding utilities

0.812	0.868	0.918	+1
0.762		0.560	-1
0.705	0.655	0.611	0.388

- Clearly, the passive learning task is similar to the policy evaluation task.
- The main difference is that the passive learning agent does not know
  - Neither the transition model T(s, a,s'), which specifies the probabilit y of reaching state's from state s after doing action a;
  - Nor does it know the reward function R(s), which specifies the reward for each state.
- The agent executes a set of trials in the environment using its policy □ .
- In each trial, the agent starts in state (1,1) and experiences a sequence of state transitions until it reaches one of the terminal states, (4,2) or (4,3).
- Its percepts supply both the current state and the reward received in that state.
- Typical trials might look like this:

(1 ,1)-0.4 → (1, 2)-0.4 → (1,3)-0.4 → (1,2)-0.4 → (1,3)-0.4 → (2,3)-0.4 → (3,3)-0.4 → (4,3)+1

(1 ,1)-0.4 → (1, 2)-0.4 → (1,3)-0.4 → (2,3)-0.4 → (3,3)-0.4 → (3,2)-0.4 → (3,3)-0.4 → (4,3)+1

(1 ,1)-0.4 → (2, 1)-0.4 → (3,1)-0.4 → (3,2)-0.4 → (4,2)-1

- Note that each state percept is subscripted with the reward received.
- The object is to use the information about rewards to learn the expected utility U □ (s) associated with each nonterminal states.
- The utility is defined to be the expected sum of (discounted) rewards obtained if policy is □ followed, the utility function is written a s□

$$U^{\pi}(s) = E\left[\sum \gamma^{t} R(s_t) \mid \pi, s_0 = s\right]$$

- For the 4 x 3 world set $\gamma = 1$

Direct utility estimation:-

- A simple method for direct utility estimation is in the area of adaptive control theory by Widrow and Hoff(1960).
- The idea is that the utility of a state is the expected total reward from that state onward, and each trial provides a sample of this value for each state visited.
- Example:- The first trial in the set of three given earlier provides a sample total reward of 0.72 for state (1,1), two samples of 0.76 and 0.84 for (1,2), two samples of 0.80 and 0.88 for (1,3) and so on.
- Thus at the end of each sequence, the algorithm calculates the observed reward- to-go for each state and updates the estimated utility for that state accordingly.
- In the limit of infinitely many trails, the sample average will come together to the true expectations in the utility function.
- It is clear that direct utility estimation is just an instance of supervised learning.
- This means that reinforcement learning have been reduced to a standard inductive learning problem.

Advantage:-

- Direct utility estimation succeeds in reducing the reinforcement learning problem to an inductive learning problem.
- **Disadvantage:-**
  a. It misses a very important source of information, namely, the fact that the utilities of states are not independent
     i. **Reason:-** The utility of each state equals its own reward plus the expected utility of its successor states. That-is, the utility values obey the Bellman equations for a fixed policy
     ii. $U^{\pi}(s) = R(s) + \gamma \sum T(s, \pi(s), s`)U^{\pi}(s`)$
  b. It misses opportunities for learning
     i. **Reason:-** It ignores the connections between states
  c. The algorithm often converges very slowly.
     i. **Reason:-** More broadly, direct utility estimation can be viewed as searching in a hypothesis space for U that is much larger that it needs to be, in that it includes many functions that violate the Bellman equations.

ADAPTIVE DYNAMIC PROGRAMMING:-

- Agent must learn how states are connected.
- Adaptive Dynamic Programming agent works by learning the transition model of the environment as it goes along and solving the corresponding Markov Decision process using a dynamic programming method.
- For passive learning agent, the transition model $T(s, \pi(s), s`)$ and the observed rewards R(S) into Bellman equation to calculate the utilities of the states.
- The process of learning the model itself is easy, because the environment is fully observable i.e. we have a supervised learning task where the input is a state-action pair and the output is

the resulting state.

- We can also represent the transition model as a table of probabilities.
- The following algorithm shows the passive ADP agent,

```
Function PASSIVE-ADP-AGENT(percept) returns an action

    Inputs:    percept,a percept indicating the current state s' and reward signal r'
    Static:    π a,fixed policy

               Mdb,an MDP with model T,rewards R,discount γ
               U,a table of utilities,initially empty
               N_sa,a table of frequencies for state-action pairs,initially zero
               N_sas,a table of frequencies for state-action-state triples,initially zero

S,a,the previous state and action,initially null
If s' is new then do U[s']←r' ; R[s']←r'

   If s is not null then do

      Increment N_sa[s,a]andN_sas'[s,a,s']
      For each t such that N_sas'[s,a,t]is nonzero do

         T[s,a,t]←N_sas'[s,a,t]/N_sa[s,a]
   U←VALUE-DETERMINATION(π,U,mdb)
   If TERMINALS?[s']then s,a←null else s,a←s',π[s']
   return a
```

- Its performance on the 4 * 3 world is shown in the following figure.
- The following figure shows the root-mean square error in the estimate for U(1,1), averaged over 20 runs of 100 trials each.

Advantages:-

It can converges quite quickly

Reason:- The model usually changes only slightly with each observation, the value iteration process can use the previous utility estimates as initial values.

The process of learning the model itself is easy

Reason:- The environment is fully observable. This means that a supervised learning task exist where the input is a state-action pair and the output is the resulting state.

It provides a standard against which other reinforcement learning algorithms can be measured.

Disadvantage:-

It is intractable for large state spaces

Temporal Difference Learning:-

- In order to approximate the constraint equation $U^{\pi}(S)$, use the observed transitions to adjust the values of the observed states, so that they agree with the constraint equation.
- When the transition occurs from S to S^1, we apply the following update to $U^{\pi}(S)$

  $U^{\pi}(S) \leftarrow U^{\pi}(S) + \alpha(R(S) + \gamma U^{\pi}(S^1) - U^{\pi}(S))$

- Where = learning rate parameter.
- The above equation is called Temporal difference or TD equation.
- The following algorithm shows the passive reinforcement learning agent using temporal differences,

```
Function PASSIVE-TD-AGENT(precept) returns an action

  Inputs: percept, a percept indicating the current state s' and reward signal r'
  Static: π, a fixed policy
          U, a table of utilities, initially empty
          N_s, a table of frequencies for states, initially zero
          S, a, r, the previous state, action, and reward, initially null
  If s' is new then U[s'] ← r'
  If s is not null then do
        Increment N_s[s]
        U[s] ← U[s] + α(N_s[s])(r + γU[s'] - U[s])
  If TERMINAL?[s'] then s,a,r ← null else s,a,r ← s',π[s'],r'
  return a
```

- **Advantages:-**
    - It is much simpler
    - It requires much less computation perobservation
- **Disadvantages:-**
    - It does not learn quite as fast as the ADP agent
    - It shows much higher variability
- The following table shows the difference between ADP and TD approach,

ADP Approach	**TD Approach**
ADP adjusts the state to agree with all of the successors that might occur, weighted by their probabilities	TD adjusts a state to agree with its observed successor
ADP makes as many adjustments as it needs to restore consistency between the utility estimates U and the environment model T	TD makes a single adjustment per observed transition

- The following points shows the relationship between ADP and TD approach,
    - Both try to make local adjustments to the utility estimates in order to make each state "agree" with its successors.
    - Each adjustment made by ADP could be seen, from the TD point of view, as a result of a "pseudo-experience" generated by simulating the current environment model.
    - It is possible to extend the TD approach to use an environment model to generate several "pseudo-experiences-transitions that the TD agent can imagine might happen, given its current model.

- For each observed transition, the TD agent can generate a large number of imaginar y transitions. In this way the resulting utility estimates will approximate more and more closely those of ADP- of course, at the expense of increased computationtime.

Active Reinforcement learning:-

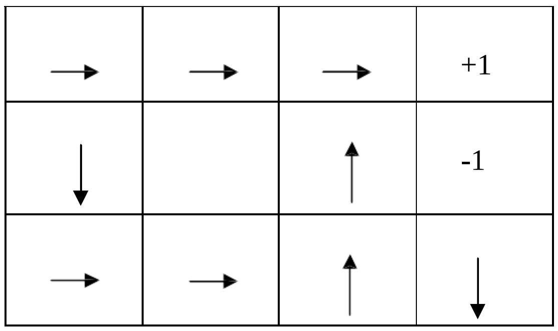
- A passive learning agent has a fixed policy that determines its behavior.
- **“An active agent must decide what actions to do”**
- An ADP agent can be taken an considered how it must be modified to handle this new freedom.
- The following are the **required modifications:-**
  - First the agent will need to learn a complete model with outcome probabilities for all actions. The simple learning mechanism used by PASSIVE-ADP-AGENT will do just fine for this.
  - Next, take into account the fact that the agent has a choice of actions. The utilities it needs to learn are those defined by the optimal policy.

U (s) □ R(s) □ □ max T(s, a, s`)U (s`)

  - These equations can be solved to obtain the utility function U using he value iteration or policy iteration algorithms.
  - Having obtained a utility function U that is optimal for the learned model, the agent can extract an optimal action by one-step look ahead to maximize the expected utility;
  - Alternatively, if it uses policy iteration, the optimal policy is already available, so it should simply execute the action the optimal policy recommends.

Exploration:-

- Greedy agent is an agent that executes an action recommended by the optimal policy for the learned model.
- The following figure shows the suboptimal policy to which this agent converges in this particular sequence of trials.



- The agent does not learn the true utilities or the true optimal policy! what happens is that, in the 39th trial, it finds a policy that reaches +1 reward along the lower route via (2,1), (3,1),(3,2), and (3,3).

- After experimenting with minor variations from the 276th trial onward it sticks to that policy, never learning the utilities of the other states and never finding the optimal route via (1,2),(1.3) and(2,3).
- Choosing the optimal action cannot lead to suboptimal results.
- The fact is that the learned model is not the same as the true environment; what is optimal in

the learned model can therefore be suboptimal in the true environment.

- Unfortunately, the agent does not know what the true environment is, so it cannot compute the optimal action for the true environment.
- Hence this can be done by the means of **Exploitation.**
- The greedy agent can overlook that actions do more than provide rewards according to the current learned model; they also contribute to learning the true model by affecting the percepts that are received.
- An agent therefore must make a trade-off between **exploitation** to maximize its reward and **exploration** to maximize its long-term well being.

- Pure exploitation risks getting stuck in a rut.
- Pure exploitation to improve ones knowledge id of no use if one never puts that knowledge into practice.

GLIE Scheme:-

- To come up with a reasonable scheme that will eventually lead to optimal behavior by the agent a GLIE Scheme can be used.
- A GLIE Scheme must try each action in each state an unbounded number of times to avoid having a finite probability that an optimal action is missed because of an unusually bad series of outcomes.

- An ADP agent using such a scheme will eventually learn the true environment model.
- A GLIE Scheme must also eventually become greedy, so that the agents actions become optimal with respect to the learned (and hence the true) model.
- There are several GLIE Scheme as follows,
    - The agent can choose a random action a fraction 1/t of the time and to follow the greedy policy otherwise.

Advantage:- This method eventually converges to an optimal policy
Disadvantage:- It can be extremely slow

- Another approach is to give some weight to actions that the agent has not tried very often, while tending to avoid actions that are believed to be of low utility. This can be implemented by altering the constraint equation, so that it assigns a higher utilit y estimate to relatively UP explored state-action pairs.

- Essentially, this amounts to an optimistic prior over the possible environments and causes the agent to behave initially as if there were wonderful rewards scattered all over the place.

Exploration function:-

- Let U+ denotes the optimistic estimate of the utility of the state s, and let N(a,s) be the number of times action a has been tried in state s.
- Suppose that value iteration is used in an ADP learning agent; then rewrite the update equation to incorporate the optimistic estimate.
- The following equation does this,

$$U^{+}(s) \leftarrow R(s) + \gamma \max_{a} f\left(\sum_{s`} T(s, a, s`)U^{+}(s`), N(a, s)\right)$$

- Here f(u ,n) is called the **exploration** function.
- It determines how greed is trade off against curiosity.
- The function f(u, n) should be increasing in u and decreasing in n.
- The simple definition is
  f(u, n) = R^+ in n<N_C
  u otherwise
  where R^+ = optimistic estimate of the best possible reward obtainable in any state and N_C is a fixed parameter.
- The fact that U^+ rather than U appears on the right hand side of the above equation is very important.
- If U is used, the more pessimistic utility estimate, then the agent would soon become unwilling to explore further a field.
- The use of U^+ means that benefits of exploration are propagated back from the edges of unexplored regions, so that actions that lead toward unexplored regions are weighted more highly, rather than just actions that are themselves unfamiliar.

Learning an action value function:-

- To construct an active temporal difference learning agent, it needs a change in the passive TD approach.
- The most obvious change that can be made in the passive case is that the agent is no longer equipped with a fixed policy, so if it learns a utility function U, it will need to learn a model in order to be able to choose an action based on U via one step look ahead.
- The update rule of passive TD remains unchanged. This might seem old.
- **Reason:-**
  - Suppose the agent takes a step that normally leads to a good destination, but because of non determinism in the environment the agent ends up in a disastrous state.
  - The TD update rule will take this as seriously as if the outcome had been the normal result of the action, where the agent should not worry about it too much since the outcome was a fluke.
  - It can be shown that the TD algorithm will converge to the same values as ADP as the number of training sequences tends to infinity.

Generalization in Reinforcement Learning:-

- The utility function and Q-functions learned by the agents are represented in tabular form with one output value for each input tuple.
- This approach works well for small set spaces.
- **Example:-** The game of chess where the state spaces are of the order 10^{50} states. Visiting all the states to learn the game is tedious.
- One way to handle such problems is to use **FUNCTION APPROXIMATION.**
- **Function approximation** is nothing but using any sort of representation for the function other than the table.
- **For Example:-** The evaluation function for chess is represented as a weighted linear function of set of **features or basic functions f1,….fn**
  $U_{\theta}(S) = \theta_1 f_1(S) + \theta_2 f_2(S) + \cdots + \theta_n f_n(S)$

- The reinforcement learning can learn value for the parameters $\square\ \square\ \square_1 \ldots\ldots \square_n$.
- Such that the evaluation function $U_\square$ approximates the true utility function.
- As in all inductive learning, there is a tradeoff between the size of the hypothesis space and the time it takes to learn the function.
- For reinforcement learning, it makes more sense to use an online learning algorithm that updates the parameter after each trial.
- Suppose we run a trial and the total reward obtained starting at (1, 1) is 0.4.
- This suggests that $U_\square$ (1,1) , currently 0.8 is too large and must be reduced.
- The parameter should be adjusted to achieve this. This is done similar to neural network learning where we have an error function which computes the gradient with respect to the parameters.
- If Uj(S) is the observed total reward for state S onward in the jth trial then the error is defined as half the squared difference of the predicted total and the actual total. $E^j(S) \square (U(S) \square U_j(S))^2 / 2$
- The rate of change of error with respect to each parameter $\square_i$ i$\square$ $E_j / \square\square_j$, is to move the s parameter in the direction of the decreasing error.

$$\square_i \square \square_i \square\square (\square E_j(S) / C\square_j) \square \square_i \square\square(U_j(S) \square U_\square(S))(\square U_\square(S) / \square\square_i)$$

- This is called **Widrow-Hoff Rule or Delta Rule.**

Advantages:-

- It requires less space.
- Function approximation can also be very helpful for learning a model of the environment.
- It allows for inductive generalization over input states.

Disadvantages:-

- The convergence is likely to be displayed.
- It could fail to be any function in the chosen hypothesis space that approximates the true utility function sufficiently well.
- Consider the simplest case, which is direct utility estimation. With function approximation, this is an instance of supervised learning.

APPLICATIONS OF REINFORCEMENT LEARNING

1. RL in Marketing

Marketing is all about promoting and then, selling the products or services either of your brand or someone else's. In the process of marketing, finding the right audience which yields larger returns on investment you or your company is making is a challenge in itself.

And, it is one of the reasons companies are investing dollars in managing digitally various marketing campaigns. Through **real-time bidding** supporting well the fundamental capabilities of RL, your and other companies, smaller or larger, can expect: –

- more display ad impressions in real-time.
- increased ROI, profit margins.
- predicting the choices, reactions, and behavior of customers towards your products/services.

2. RL in Broadcast Journalism

Through **different types of Reinforcement Learning**, attracting likes and views along with tracking the reader's behavior is much simpler. Besides, recommending news that suits the frequently-changing preferences of readers and other online users can possibly be achieved since journalists can now be equipped with an RL-based system that keeps an eye on intuitive news content as well as the headlines. Take a look at other advantages too which Reinforcement Learning is offering to readers all around the world.

- News producers are now able to receive the feedback of their users instantaneously.
- Increased communication, as users are more expressive now.
- No space for disinformation, hatred.

3. RL in Healthcare

Healthcare is an important part of our lives and through DTRs (a sequence-based use-case of RL), doctors can discover the treatment type, appropriate doses of drugs, and timings for taking such doses. Curious to know how is this possible!! See, DTRs are equipped with: –

- a sequence of rules which confirm the current health status of a patient.
- Then, they optimally propose treatments that can diagnose diseases like diabetes, HIV, Cancer, and mental illness too.

If required, these DTRs (i.e. Dynamic Treatment Regimes) can reduce or remove the delayed impact of treatments through their multi-objective healthcare optimization solutions.

4. RL in Robotics

Robotics without any doubt facilitates **training a robot** in such a way that a robot can perform tasks – just like a human being can. But still, there is a bigger challenge the robotics industry is facing today – Robots aren't able to use common sense while making various moral, social decisions. Here, a combination of Deep Learning and Reinforcement Learning i.e. **Deep Reinforcement Learning** comes to the rescue to enable the robots with, "Learn How To Learn" model. With this, the robots can now: –

- manipulate their decisions by grasping well various objects visible to them.
- solve complicated tasks which even humans fail to do as robots now know what and how to learn from different levels of abstractions of the types of datasets available to them.

5. RL in Gaming

Gaming is something nowadays without which you, me, or a huge chunk of people can't live. With **games optimization through Reinforcement Learning** algorithms, we may expect better performances of our favorite games related to adventure, action, or mystery.

To prove it right, the Alpha Go example can be considered. This is a computer program that defeated the strongest Go (a challenging classical game) Player in October 2015 and itself became the strongest Go player. The trick of Alpha Go to defeat the player was Reinforcement Learning which kept on developing stronger as the game is constantly exposed to unexpected gaming challenges. Like Alpha Go, there are many other games available. Even you can also optimize your favorite games by applying appropriately prediction models which learn how to win in even complex situations through RL-enabled strategies.

6. RL in Image Processing

Image Processing is another important method of enhancing the current version of an image to extract some useful information from it. And there are some steps associated like:

- Capturing the image with machines like scanners.
- Analyzing and manipulating it.
- Using the output image obtained after analysis for representation, description-purposes.

Here, ML models like **Deep Neural Networks** (whose framework is Reinforcement Learning) can be leveraged for simplifying this trending image processing method. With Deep Neural Networks, you can either enhance the quality of a specific image or hide the info. of that image. Later, use it for any of your computer vision tasks.

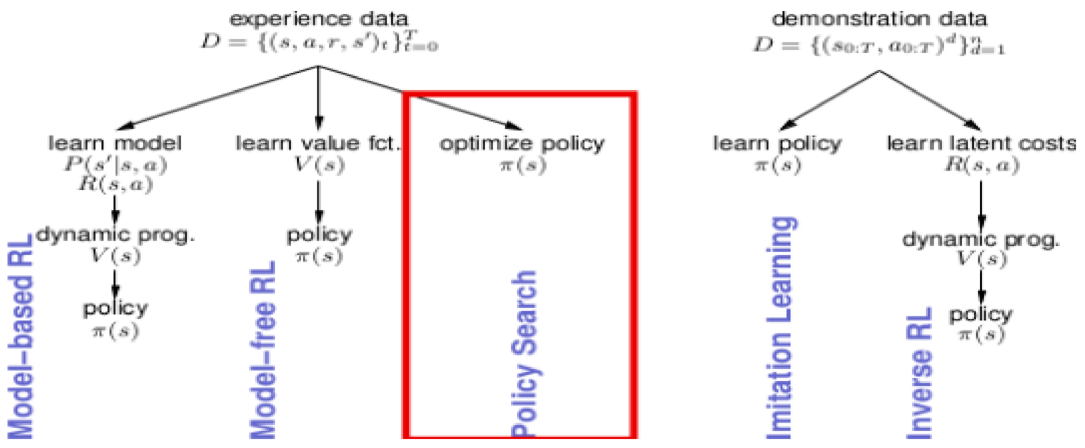
7. RL in Manufacturing

Manufacturing is all about producing goods that can satisfy our basic needs and essential wants. **Cobot Manufacturers** (or Manufacturers of **Collaborative Robots** that can perform various manufacturing tasks with a workforce of more than 100 people) are helping a lot of businesses with their own RL solutions for packaging and quality testing. Undoubtedly, their use is making the process of manufacturing quality products faster that can say a big no to negative customer feedback. And the lesser negative feedbacks are, the better is the product's performance and also, sales margin too.

POLICY SEARCH

Policy search is a subfield in reinforcement learning which focuses on finding good parameters for a given policy parametrization. It is well suited for robotics as it can cope with high-dimensional state and action spaces, one of the main challenges in robot learning.

1. Policy search methods are a family of systematic approaches for continuous (or large) actions and state space.
2. With policy search, expert knowledge is easily embedded in initial policies (by demonstration, imitation).
3. Policy search is more prefered than other RL methods in practical applications (e.g. robotics).



Policy gradient

• Family of randomized policy μ(s, a) = Pr(a|s) (deterministic policy is a special case).

• the performance measure is J(μ) = Eμ r0 + γr1 + γ 2 r2 + . . .

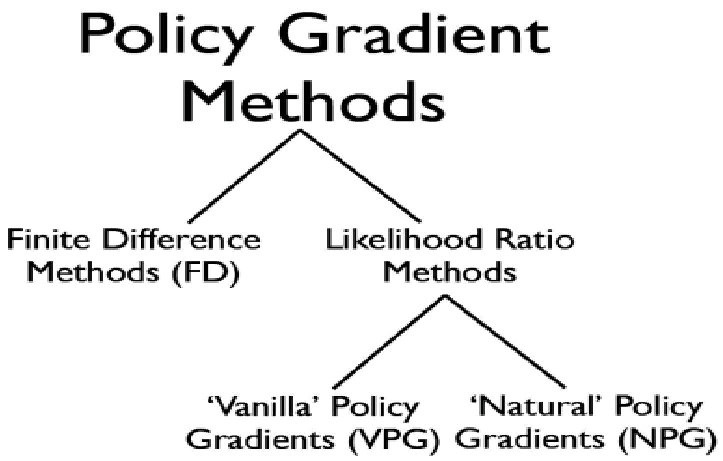
• Policy μθ(s, a) is parameterized by a parameter space θ ∈

The parametric performance measure becomes J(θ) = Eθ r0 + γr1 + γ 2 r2 + . . .

• Solution: gradient-descent algorithms. θk+1 = θk + α∇θJ(θk)

Policy gradient updates θμ0 = θμ + α∇θJ(θμ)

• Guarantee the performance improvement: J(θμ0) ≥ J(θμ) ⇒ μ 0 at least better than or equal to μ


Policy Gradient Methods
Finite Difference Methods (FD)
Likelihood Ratio Methods
'Vanilla' Policy Gradients (VPG)
'Natural' Policy Gradients (NPG)

Policy gradient: Black-box approaches

• Approximate the gradient using supervised learning (regression).

• Collect data D = {δθi , δJi} (the sampled gradients). By – perturbing the parameters: θ + δθ – applying the new policy μ(θ + δθ) to get δJi = J(θ + δθ) − J(θ)

• the finite different (FD) gradient estimation by regression gFD(θ) = (∆Θ>∆Θ)−1∆Θ>∆J

• gradient update θ ← θ + αgFD(θ)

Policy gradient: Likelihood Ratio Gradient

• rewrite the performance measure

$J(\theta)$ $J(\theta) = E\theta\ r0 + \gamma r1 + \gamma\ 2\ r2 + \ldots = Z\ p(\xi|\mu\theta)R(\xi)d\xi$

where $\xi = \{s0, a0, r0, s1, a1, r1, \ldots\}$ is a trajectory.

$R(\xi) = r0 + \gamma r1 + \gamma\ 2\ r2 + \ldots$

$p(\xi|\mu\theta) = p(s0)\ Yp(st+1|st, at)\mu\theta(at|st)$

• Gradient derivation

$\nabla\theta J(\theta) = Z\ \nabla\theta p(\xi|\mu\theta)R(\xi)d$

$\xi = Z\ p(\xi|\mu\theta)\nabla\theta \log p(\xi|\mu\theta)R(\xi)d\xi$

(the trick is $\nabla f = f\nabla \log f$)

$= E\ h\ \nabla\theta \log p(\xi|\mu\theta)R(\xi)\ i$

• Using Monte-Carlo simulations: sampling M tracjectories ξi from policy $\mu\theta$.

$\nabla\theta J(\theta) \approx 1\ M\ X\ M\ i=1\ \nabla\theta \log p(\xi i\ |\mu\theta)R(\xi i)$

$= 1\ M\ X\ M\ i=1\ X\ Ti\ t=0\ \nabla\theta \log \mu\theta(at|st)R(\xi i)$

because $p(st+1|st, at)$ not depends on θ, so its gradient w.r.t θ is 0

A vanilla policy gradient algorithm

• initialize $\theta 0$

• for $k = 0 : \infty$ (until convergence)

=generate M tracjectories ξi from policy $\mu\theta k$

=compute $\nabla\theta J(\theta k)$ from $\{\xi i\}M\ i=1$

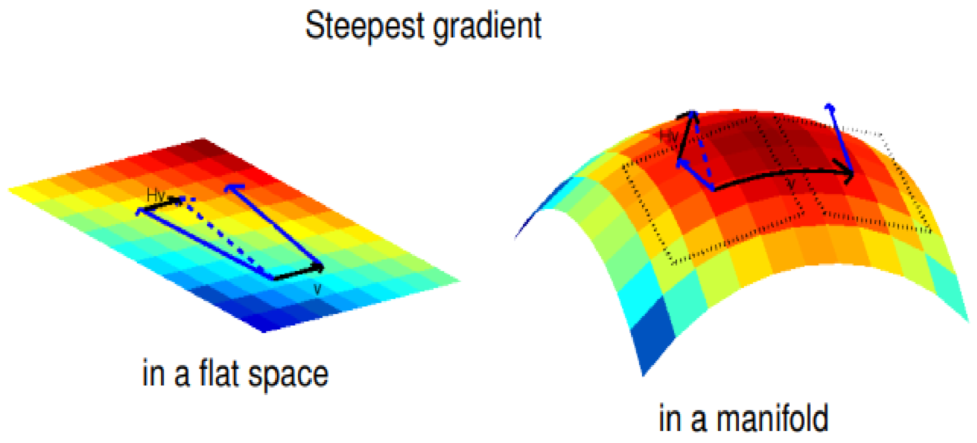
– update θ: $\theta k+1 = \theta k + \alpha\nabla\theta J(\theta k)$

• end for

Natural policy gradient

$$\nabla\tilde{}\ \theta J(\theta) = G -1\ (\theta)\nabla\theta J(\theta)$$

where $G(\theta)$ is the Fisher Information Matrix (Amari, 1998



Natural Policy Gradient: Steepest descent

• The optimization (maxmize $J(\theta)$) is over the space of trajectories, when considering $p(\xi|\mu\theta)$ is a function of parameter ξ (instead of θ) of dimension $|\theta|$.

• We try to define a Riemannian structure on the manifold of trajectories ξ

The optimization (maxmize $J(\theta)$) is over the space of trajectories, when considering $p(\xi|\mu\theta)$ is a function of parameter ξ (instead of θ) of dimension $|\theta|$.

• We try to define a Riemannian structure on the manifold of trajectories ξ

• The steepest descent should minimize the $J(\theta + \delta\theta)$ after update. This is formulated as an optimization problem min $J(\theta + \delta\theta) = J(\theta) + \delta\theta\nabla J(\theta)$ subject to $h\delta\theta, \delta\theta ip(\xi|\mu\theta) =$

Using Lagrange multiplier

$$L(\theta, \lambda) = J(\theta) + \delta\theta\nabla J(\theta) + \lambda(X\ i,j\ Gi,j\ \delta\theta i, \delta\theta j -)$$

• Taking derivative

$$\theta i, \nabla J(\theta) + \lambda Gi,j\ \delta\theta j = 0$$

• Therefore, $\delta = G -1\nabla J(\theta)$

The metric is the distances on probability spaces.

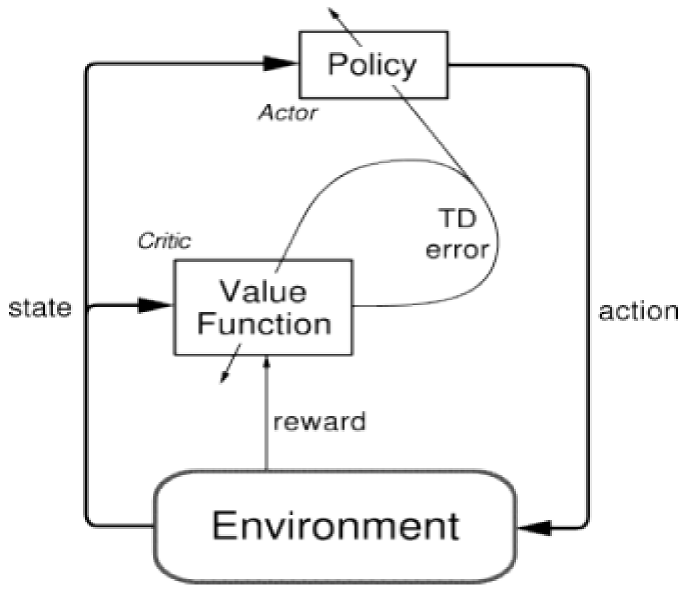
The KL-divergence between two distributions is a natural divergence on changes in a distribution. $G(\theta) = h\delta i \log p(\xi|\mu\theta), \delta j \log p(\xi|\mu\theta)ip(\xi|\mu\theta)$

• Estimate the Fisher information matrix using sampled trajectories.

G(θ) ≈ 1 M X M i=1 [∇θ log p(ξi |μθ)R(ξi)] × [∇θ log p(ξi |μθ)R(ξi)]>

= 1 M X M i=1 [X Ti t=0 ∇θ log μθ(at|st)R(ξi)] × [X Ti t=0 ∇θ log μθ(at|st)R(ξi)]

Actor-Critic methods



1. The policy structure is known as the actor, (→ tuned by gradient updates.)
2. The estimated value function is known as the critic (→ tuned by TD error).

Policy Gradient Theorem Algorithm

• R(ξ) can be estimated by Qπ t (st, at)
∇θJ(θ) = E hX T t=0 ∇θ log μθ(at|st) X T j=t rj i
= E hX T t=0 ∇θ log μθ(at|st)Q π t (st, at)

Policy gradient with function approximation

Q π t (st, at) = φ(st, at) > · w
• Compatible function approximation: how to choose φ(st, at)? such that
– does not introduce bias
– reduce the variance