

# Unit 5- Other Types of Learning

J.Premalatha

Professor/IT

Kongu Engineering College

Perundurai

# Reinforcement Learning(RL)

➤ Reinforcement Learning is a **feedback-based Machine learning technique** in which an agent(computer program) 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 or rewards** and for **each bad action, the agent gets negative feedback or penalty**.

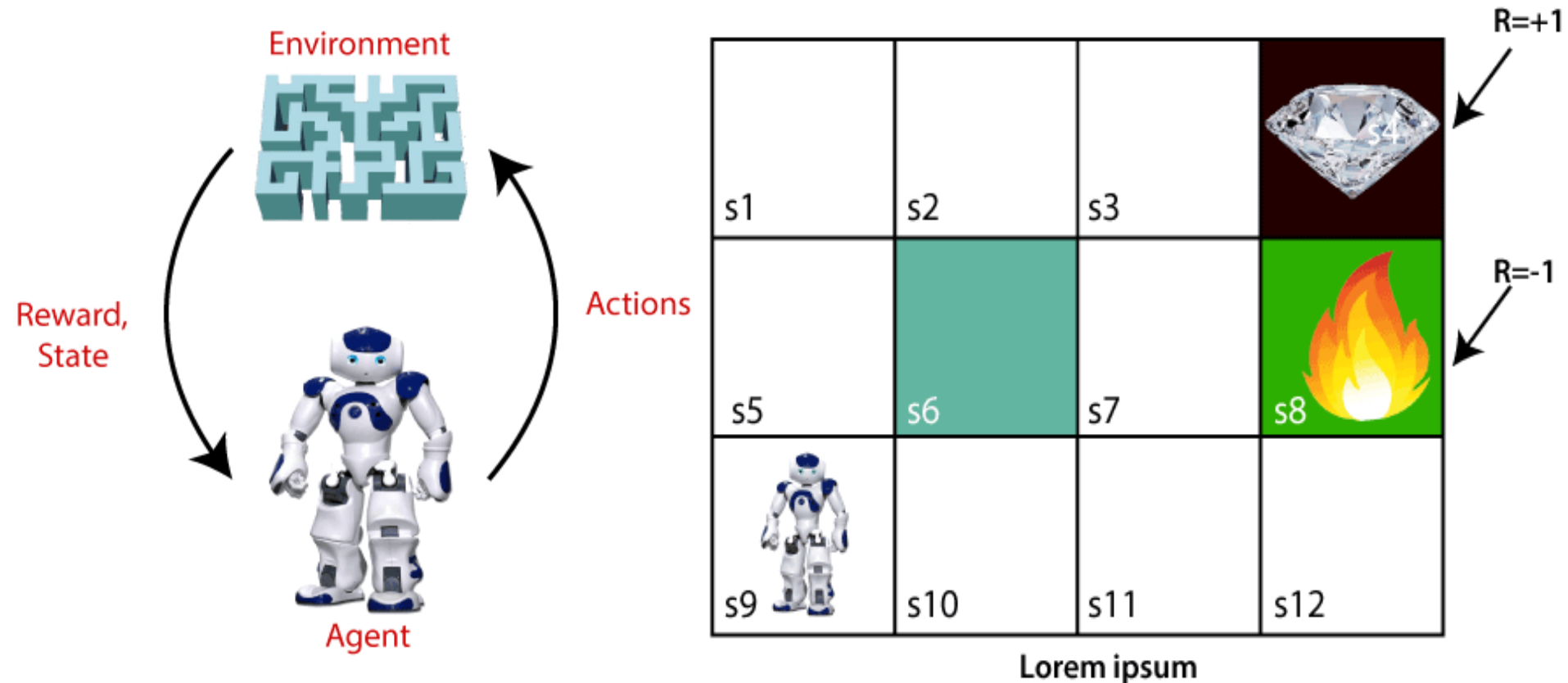
➤ In Reinforcement Learning, **the agent learns automatically using feedbacks without any labeled data**, unlike supervised learning. Since there is no labeled data, so the **agent is to learn by its experience only**.

➤ **RL solves a specific type of problem** where **decision making is sequential and the goal is long-term, such as games, 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**.

➤ **Example:** In maze environment, the agent 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, it learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback or penalty. **As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.**



## 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.

## Elements of Reinforcement Learning

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

- i. Policy
- ii. Reward Signal
- iii. Value Function
- iv. Model of the environment

**1) Policy  $\pi$  :** The policy  $\pi$ , defines the agent's behavior and is a mapping from the states of the environment to actions:  $\pi : S \rightarrow A$ . The **policy defines the action to be taken in any state  $s_t : a_t = \pi(s_t)$** . The value of a policy  $\pi$ ,  $V^\pi(s_t)$ , is the expected cumulative reward that will be received while the agent follows the policy, starting from state  $s_t$ .

**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.**

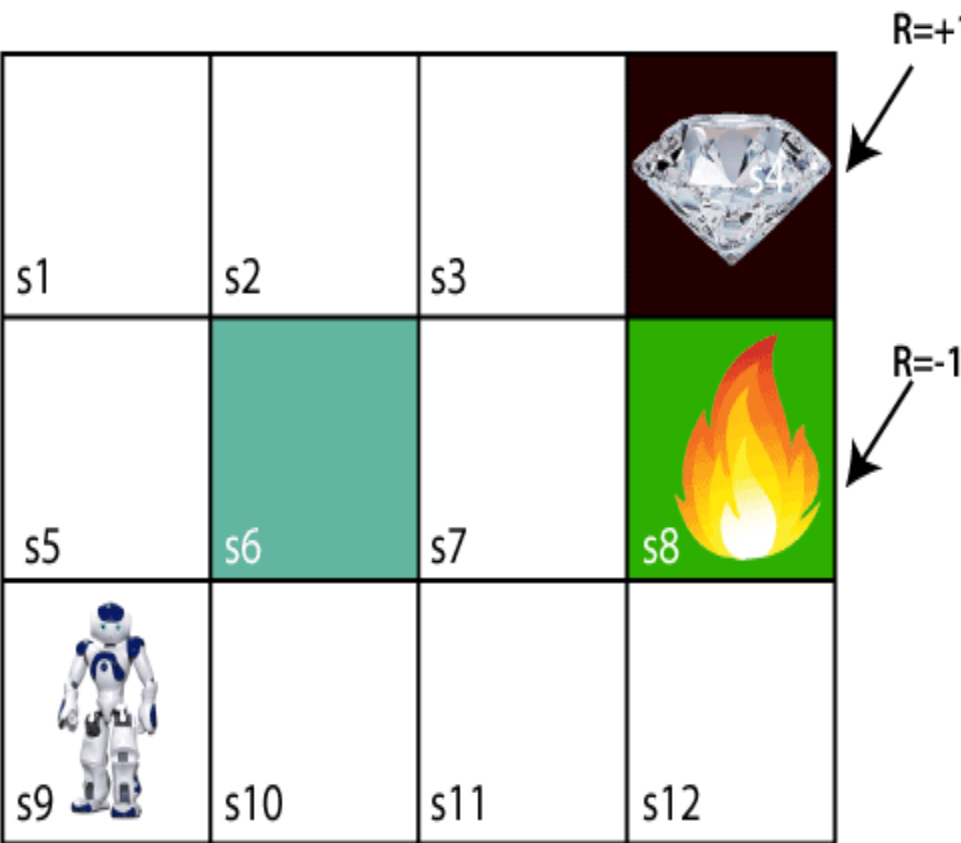
The model is used for planning, which means it provides a way to take a course of action by considering all future situations before actually experiencing those situations. The approaches for solving the **RL problems with the help of the model are termed as the model-based approach**. Comparatively,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

# How does Reinforcement Learning Work?

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

- 1.Environment: It can be anything such as a room, maze, football ground, etc.
- 2.Agent

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







Lorem ipsum

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.







$V=1$ s1	$V=1$ s2	$V=1$ s3	 s4
$V=1$ s5	 s6	s7	 s8
 $V=1$ s9	s10	s11	s12

Now, the agent has successfully stored the previous steps assigning the 1 value to each previous block.

If assigned all blocks as 1, then it is difficult to reach the goal.

[Ex]: If the agent starts moving from the block, which has 1 value block on both sides?



s1 	s2  V=1	s3 V=1	s4 
s5  V=1	s6 	s7	s8 
s9 V=1	s10	s11	s12

It will be a difficult condition for the agent whether it 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, 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.

**Bellman Equation** is used to calculate the value functions in dynamic programming or **environment that leads to modern reinforcement learning.**

## How to represent the agent state?

Represent the agent state using the **Markov State** that contains all the required information from the history. The State  $S_t$  is Markov state if it follows the given condition:

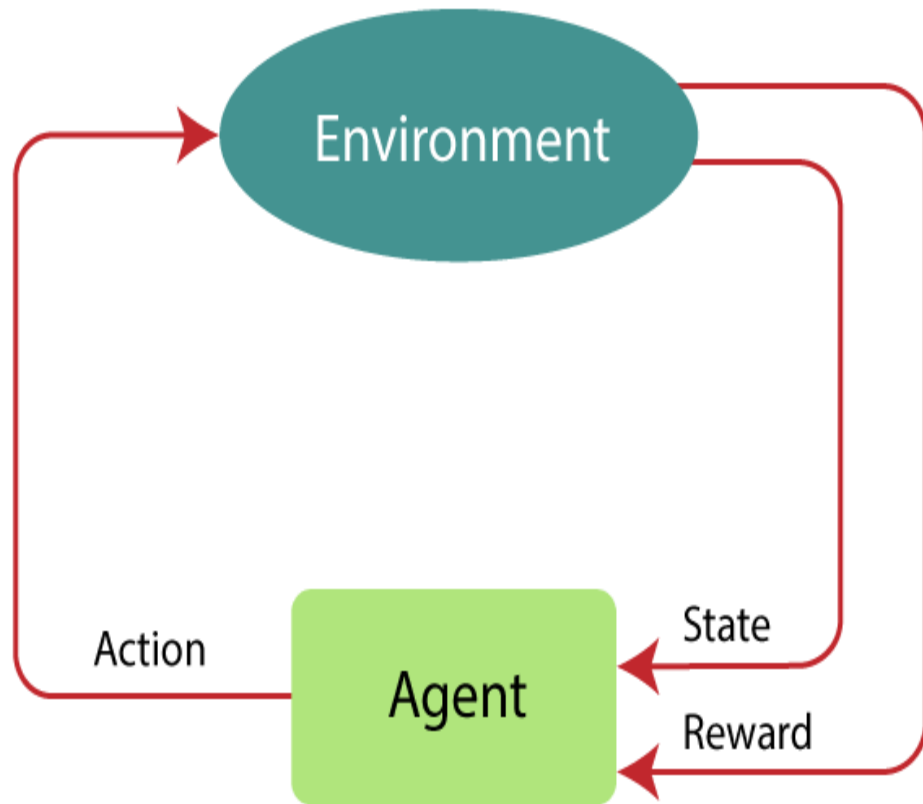
$$P[S_{t+1} | S_t] = P[S_{t+1} | S_1, \dots, S_t]$$

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.

## Markov Decision Process or MDP

**It is used to formalize the reinforcement learning problems. If the environment is completely observable, then it 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  $(S, A, P_a, R_a)$ :

1. A set of **finite States  $S$**
2. A set of **finite Actions  $A$**
3. **Rewards received after transitioning from state  $S$  to state  $S'$ , due to action  $a$ .**
4. **Probability  $P_a$ .**

MDP uses **Markov property**

**Markov Property:** It says that "If the agent is present in the current state  $S_1$ , performs an action  $a_1$  and move to the state  $s_2$ , then the state transition from  $s_1$  to  $s_2$  only depends on the current state and future action and states do not depend on past actions, rewards, or states.

## Definition of initial state, goal state, Episode, Policy and Optimal Policy

Depending on the application, a **certain state may be designated** as the **initial state(search starts)** and in some applications, there is a **goal state where the search ends**; all actions in this terminal state transition to itself with probability 1 and without any reward.

The **sequence of actions from the start to the terminal state is an episode or a trial**.

The **policy,  $\pi$** , defines the agent's behavior and is a **mapping from the states of the environment to actions**:  $\pi : S \rightarrow A$ . The policy defines the action to be taken in any state  $s_t : a_t = \pi(s_t)$ . The **value of a policy  $\pi$ ,  $V^\pi(s_t)$** , is the expected cumulative reward that will be received while the agent follows the policy, starting from state  $s_t$ .

For each policy  $\pi$ , optimal policy there is a  $V^\pi(s_t)$ , and to find the optimal policy  $\pi^*$  such that

$$V^*(s_t) = \max [V^\pi(s_t)] \quad , \quad \forall s_t$$

**Problem : Apply Bellman equation for the given Maze game. Agent is at s9th block , goal is at s4, wall is at s6, fire is at s8.**

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."

The Discount factor is " $\gamma$ "

**The Bellman equation is:**  $V(s) = \max [R(s,a) + \gamma V(s')]$  Where,

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

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

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

**For s3 block:**

$$V(s3) = \max [R(s,a) + \gamma V(s')]$$

Here  $V(s') = 0$  because there is no previous state .

The Reward of  $R(s,a) = 1$  [ **If it move to right side the reward is 1** ]

$$V(s3) = \max [R(s,a) + \gamma V(s')] \quad \gamma = 0.9 (\text{Assume})$$

$$\max [1 + 0.9 \times 0]$$

$$V(s3) = \max [1] \rightarrow V(s3) = 1.$$

### **For s2 block:**

$$V(s_2) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 1$ , and  $R(s, a) = 0$ , because there is no reward at this state.

$$V(s_2) = \max[0 + 0.9(1)]$$

$$V(s_2) = \max[0.9] \rightarrow V(s_2) = 0.9$$

### **For s1 block:**

$$V(s_1) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.9$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_1) = \max[0 + 0.9(0.9)]$$

$$V(s_1) = \max[0.81] \rightarrow V(s_1) = 0.81$$

### **For s5 block:**

$$V(s_5) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.81$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_5) = \max[0 + 0.9(0.81)]$$

$$V(s_5) = \max[0.73] \rightarrow V(s_5) = 0.73$$




### For s9th block:

$$V(s_9) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.73$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_9) = \max[0 + 0.9(0.73)]$$

$$V(s_9) = \max[0.66] \rightarrow V(s_9) = 0.66$$

$V=0.81$ $s_1$	$V=0.9$ $s_2$	$V=1$ $s_3$	 $s_4$
$V=0.73$ $s_5$	$s_6$	$s_7$	 $s_8$
 $V=0.66$ $s_9$	$V=0.59$ $s_{10}$	$V=0.53$ $s_{11}$	$V=0.48$ $s_{12}$

### For s10th block:

$$V(s_{10}) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.66$ , and  $R(s,a) = 0$ , because there is no reward at this state also.

$$V(s_{10}) = \max[0 + 0.9(0.66)]$$

$$V(s_{10}) = \max[0.59] \rightarrow V(s_{10}) = 0.59$$

Similarly find for  $s_{11}, s_{12}, s_7$

$$V(s_{11}) = 0.53$$

$$V(s_{12}) = 0.48$$

For s7th block: The agent is in 7<sup>th</sup> block, there are 4 actions.  
 Left side is wall, Right side is Fire, Up is s3, Bottom is s11. Compare  
 s3 and s11 values, the value of s3 is high.

The **agent may change the route because it always tries to find the optimal path**. So now, let's **consider from s7th block and the previous state is s3**,




$$V(s7) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 1$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s7) = \max[0 + 0.9 \times 1]$$

$$V(s7) = \max[0.99] \rightarrow V(s7) = 0.9$$

For s11, consider the previous state as s7 or s10, find the value for s11 which is higher than the previous value by s7.

V=0.81 s1	V=0.9 s2	V=1 s3	 s4
V=0.73 s5		V=0.9 s7	 s8
 V=0.66 s9	V=0.59 s10	V=0.53 s11	V=0.48 s12



### For s11th block:

$$V(s_{11}) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.9$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_{11}) = \max[0 + 0.9(0.9)]$$

$V(s_{11}) = \max[0.81] \rightarrow V(s_{11}) = \mathbf{0.81}$  which is greater than the previous value **0.53**

### For s10th block:

$$V(s_{10}) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.81$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_{10}) = \max[0 + 0.9(0.81)]$$

$V(s_{10}) = \max[0.73] \rightarrow V(s_{10}) = \mathbf{0.73}$  which is greater than the previous value **0.59**

## For s12th block:

$$V(s_{12}) = \max [R(s,a) + \gamma V(s')]$$

Here  $\gamma = 0.9$ ,  $V(s') = 0.81$ , and  $R(s, a) = 0$ , because there is no reward at this state also.

$$V(s_{12}) = \max[0 + 0.9(0.81)]$$

$V(s_{12}) = \max[0.73] \rightarrow V(s_{12}) = 0.73$  which is greater than the previous value 0.59

$V=0.81$ s1	$V=0.9$ s2	$V=1$ s3	 s4
$V=0.73$ s5	 s6	$V=0.9$ s7	 s8
$V=0.66$ s9	$V=0.73$ s10	$V=0.81$ s11	$V=0.73$ s12

# Types of Reinforcement learning

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

1. Model Based Learning
2. Temporal based Learning

**Model Based Learning** : The agent learn the optimal behavior by learning a model of the environment by taking actions and observing the outcomes that include the next state and the immediate reward. **The agent know the environment model parameters** and do not need any exploration and **can directly solve for the optimal value function and policy using dynamic programming.**

**Temporal based Learning** : The agent learning from an environment through episodes **with no prior knowledge of the environment.** The **difference between the current state Q value and the next state of one time step later is called as Temporal Difference(TD).** Such algorithms are called temporal difference (TD) algorithms

## Model based Reinforcement Learning

In this learning, the **optimal value function is known, the optimal policy is to choose the action that maximizes the value in the next state.**

$$\pi^*(s_t) = \arg \max_{a_t} \left( E[r_{t+1} | s_t, a_t] + \gamma \sum_{s_{t+1} \in S} P(s_{t+1} | s_t, a_t) V^*(s_{t+1}) \right)$$

Where  $\pi$  is the policy

$E[r_{t+1}]$  is the expected reward at  $S_{t+1}$  stage

$V[S_{t+1}]$  is the value function

$\gamma$  is the Discount factor

Model-based algorithms are Dynamic Programming (**Policy Iteration and Value Iteration**) - these all use the model's predictions of next state and reward in order to calculate optimal actions. Specifically in Dynamic Programming, **the model must provide state transition probabilities, and expected reward from any state, action pair. This is rarely a learned model.**

**Value Iteration:** To **find the optimal policy**, **use the optimal value function**, and there is an iterative algorithm called value iteration that has been shown to converge to the correct  $V^*$  values. Its pseudocode is

```
Initialize  $V(s)$  to arbitrary values
Repeat
  For all  $s \in S$ 
    For all  $a \in \mathcal{A}$ 
       $Q(s, a) \leftarrow E[r|s, a] + \gamma \sum_{s' \in S} P(s'|s, a)V(s')$ 
     $V(s) \leftarrow \max_a Q(s, a)$ 
Until  $V(s)$  converge
```

**Figure 18.2** Value iteration algorithm for model-based learning.

Where  $Q$  is the value of the action ‘a’ in the state ‘s’.

**Policy Iteration:** In policy iteration, **store and update the policy**. The idea is to **start with a policy and improve it repeatedly until there is no change. The value function can be calculated**. Then check whether improve the policy or not by considering values of the value function. This step is guaranteed to improve the policy, and when no improvement is possible, the policy is guaranteed to be optimal. The pseudocode is

Initialize a policy  $\pi'$  arbitrarily

Repeat

$\pi \leftarrow \pi'$

Compute the values using  $\pi$  by  
solving the linear equations

$$V^\pi(s) = E[r|s, \pi(s)] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, \pi(s)) V^\pi(s')$$

Improve the policy at each state

$$\pi'(s) \leftarrow \arg \max_a (E[r|s, a] + \gamma \sum_{s' \in \mathcal{S}} P(s'|s, a) V^\pi(s'))$$

Until  $\pi = \pi'$

**Figure 18.3** Policy iteration algorithm for model-based learning.

**Problem: Find the optimal policy for the ship to reach its homeland safely with maximum rewards.**

Consider a pirate ship, which is currently anchored at an island and, has to reach its homeland safely. There are two routes it could choose from.

If it takes the route to the north, then it can reach an island which is full of gold, which it can collect, and then move south to reach the homeland. However, there is an area with a very high gravity (like the Bermuda triangle) to the north of the gold island. If the ship reaches there by mistake, then it will be sucked into it and the ship is lost forever.

If it takes the route to the south, then it can reach an island, which is full of silver, which it can collect, and then move north to reach the homeland. There is a prison island to the south of the silver island. If the ship lands there by mistake, then the ship will be captured and the crew will be imprisoned. The ship's captain possesses a broken compass. . So, every time, the captain makes a move towards north, he moves north with a probability of 0.8, however, he might miss the mark and reach south with a probability of 0.2. Similarly, if he moves south, there is a 0.8 probability of going south and 0.2 probability of going north.

## Rewards (Positive and Negative):

The rewards to each of the ship's landing are,

- Reaching homeland will allow the pirate ship to collect +1 point.
- Landing on the gold island will allow the pirate ship to collect +2 points.
- Landing on the silver island will allow the pirate ship to collect +1 point.
- If the ship gets sucked into Bermuda triangle, then the ship gets -2 points.
- If the ship is captured in the prison island, then the ship get -0.5 points.

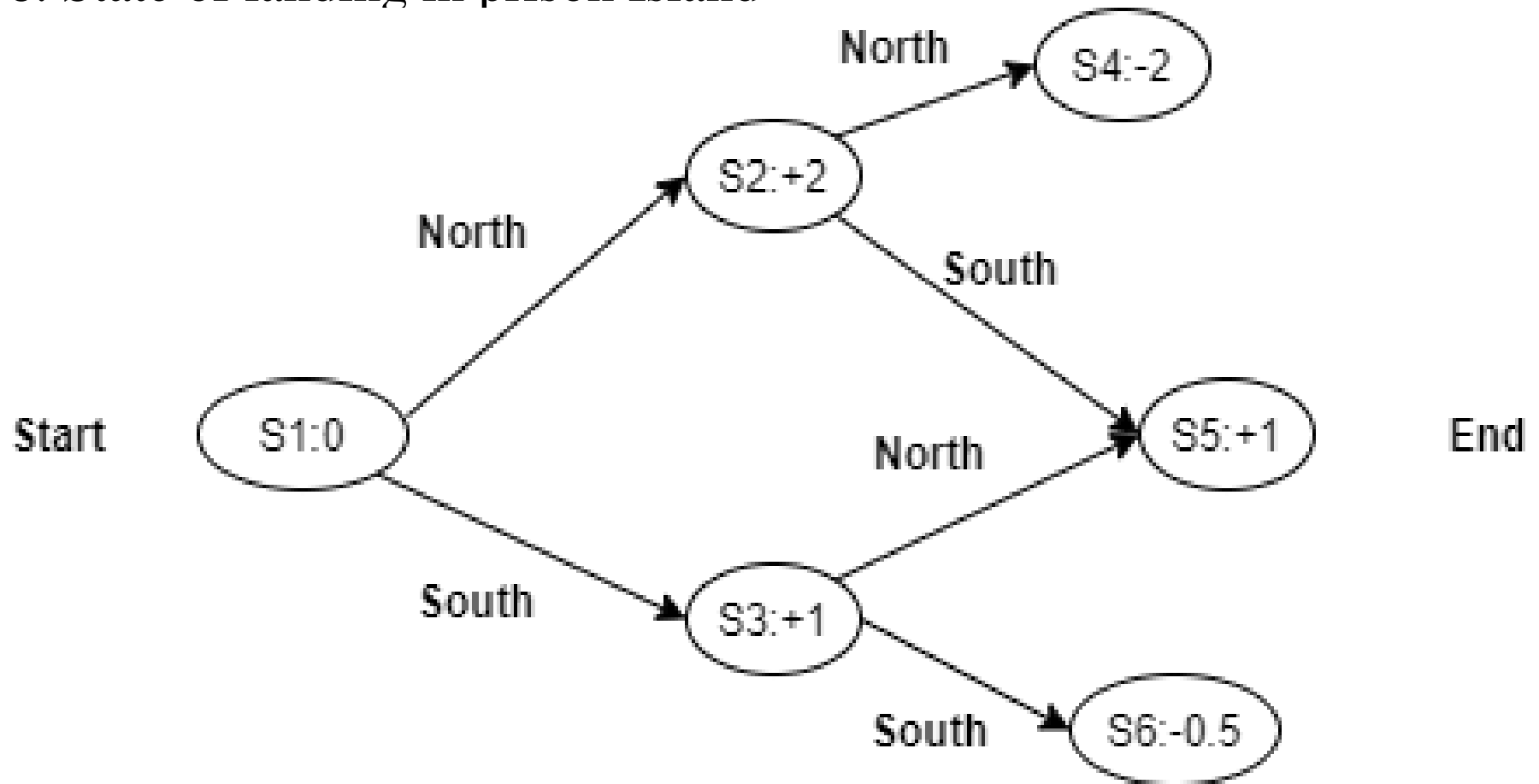
**Solution:** Understand the problem and draw the state transition diagram with states and actions.

Each island is a state and there are two actions, 'north' and 'south'. Rewards for each state should included in the diagram.

There are six states including start and destination states. Let us label the states from S1 to S6 as follows:



S1: Start state  
S2: State of landing in gold Island  
S3: State of landing in silver Island  
S4: State of landing in Bermuda Triangle Island  
S5: State of landing in destination Island  
S6: State of landing in prison Island



**Transition probability matrix:** Compute transition probabilities for each state/action pair. Based on the probabilities provided above, the state transition diagram and two actions[North, South] to build a transition probability matrix for both actions. Applying Bellman's equation,  $T(S, a, S')$  refers to the transition probability of moving from state  $S$  to state  $S'$  after taking an action 'a'.

### Transition Probability Matrix for Action North

$$T[A(North)] \begin{bmatrix} & S1 & S2 & S3 & S4 & S5 & S6 \\ S1 & 0 & 0.8 & 0.2 & 0 & 0 & 0 \\ S2 & 0 & 0 & 0 & 0.8 & 0.2 & 0 \\ S3 & 0 & 0 & 0 & 0 & 0.8 & 0.2 \\ S4 & 0 & 0 & 0 & 0 & 0 & 0 \\ S5 & 0 & 0 & 0 & 0 & 0 & 0 \\ S6 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

## Transition Probability Matrix for Action South

$$T[A(South)] = \begin{bmatrix} & S1 & S2 & S3 & S4 & S5 & S6 \\ S1 & 0 & 0.2 & 0.8 & 0 & 0 & 0 \\ S2 & 0 & 0 & 0 & 0.2 & 0.8 & 0 \\ S3 & 0 & 0 & 0 & 0 & 0.2 & 0.8 \\ S4 & 0 & 0 & 0 & 0 & 0 & 0 \\ S5 & 0 & 0 & 0 & 0 & 0 & 0 \\ S6 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

### Policy Iteration algorithm:

**Step 1:** Randomly initialize the policy. Initialize actions randomly at every state of the system.

**Step 2(Policy Evalation) :** Calculate the value function of each state using Bellman equation[sum of the (reward + total value of each action)]

$$V(s) = r(s) + \gamma * \max \left( \sum_{s', r} p(s', r | s, \pi(s)) * V(s') \right)$$

Here is p is the transition probability, also denoted by T.

**Step 3 (policy improvement)** : For every state, get the best action from value function using

$$\pi(s) = \underset{a}{\operatorname{argmax}} \sum_{s', r} p(s', r | s, a) * V(s')$$

If the best action is better than the present policy action, then replace the current action by the best action.

### Policy Iterations:

Iterate steps 2 and 3, until convergence. If the policy did not change throughout an iteration, then consider that the algorithm has converged.

### Step 1: Randomly initialize the policy.

Let randomly initialize the policy (state to action mapping) as moving north for all states.  $P = \{N, N, N, N, N, N\}$

If observe the state transition diagram, the states S4, S5, S6 do not have any actions supported in these states, as these are end states. So let curtail the policy, to apply to only the first three states where action can taken.  
 $P = \{N, N, N\}$

**The Bellman equation is:**  $V(s) = \max [R(s,a) + \gamma V(s')]$

**First iteration:**

Let us assume the initial value  $V(s)$  for all states as 0 and the discount factor  $\lambda = 0$ . Thus, the **Bellman equation would reduce to  $V(s) = R(s)$ , where  $R(s)$  is the reward for entering a state.**

**Policy Evaluation for first iteration:**

$V[S1] = 0$ ;  $V[S2] = 2$ ;  $V[S3] = 1$ ;  $V[S4] = -2$ ;  $V[S5] = 1$ ;  $V[S6] = -0.5$

**Policy Improvement for first iteration:** Let us apply the equation provided for Policy Improvement.

### **I Iteration : Policy Improvement**

State	Action	$V[S]$	Max Action
S1	North	$V[S] = 0.8 * 2 + 0.2 * 1 = 1.8$	North
	South	$V[S] = 0.2 * 2 + 0.8 * 1 = 1.2$	
S2	North	$V[S] = 0.8 * -2 + 0.2 * 1 = -1.4$	South
	South	$V[S] = 0.2 * -2 + 0.8 * 1 = 0.4$	
S3	North	$V[S] = 0.8 * 1 + 0.2 * -0.5 = 0.7$	North
	South	$V[S] = 0.2 * 1 + 0.8 * -0.5 = -0.2$	

The **policy obtained** based on the above table is as follows:  $P = \{N, S, N\}$   
**Second iteration: Policy Evaluation for the second iteration:**

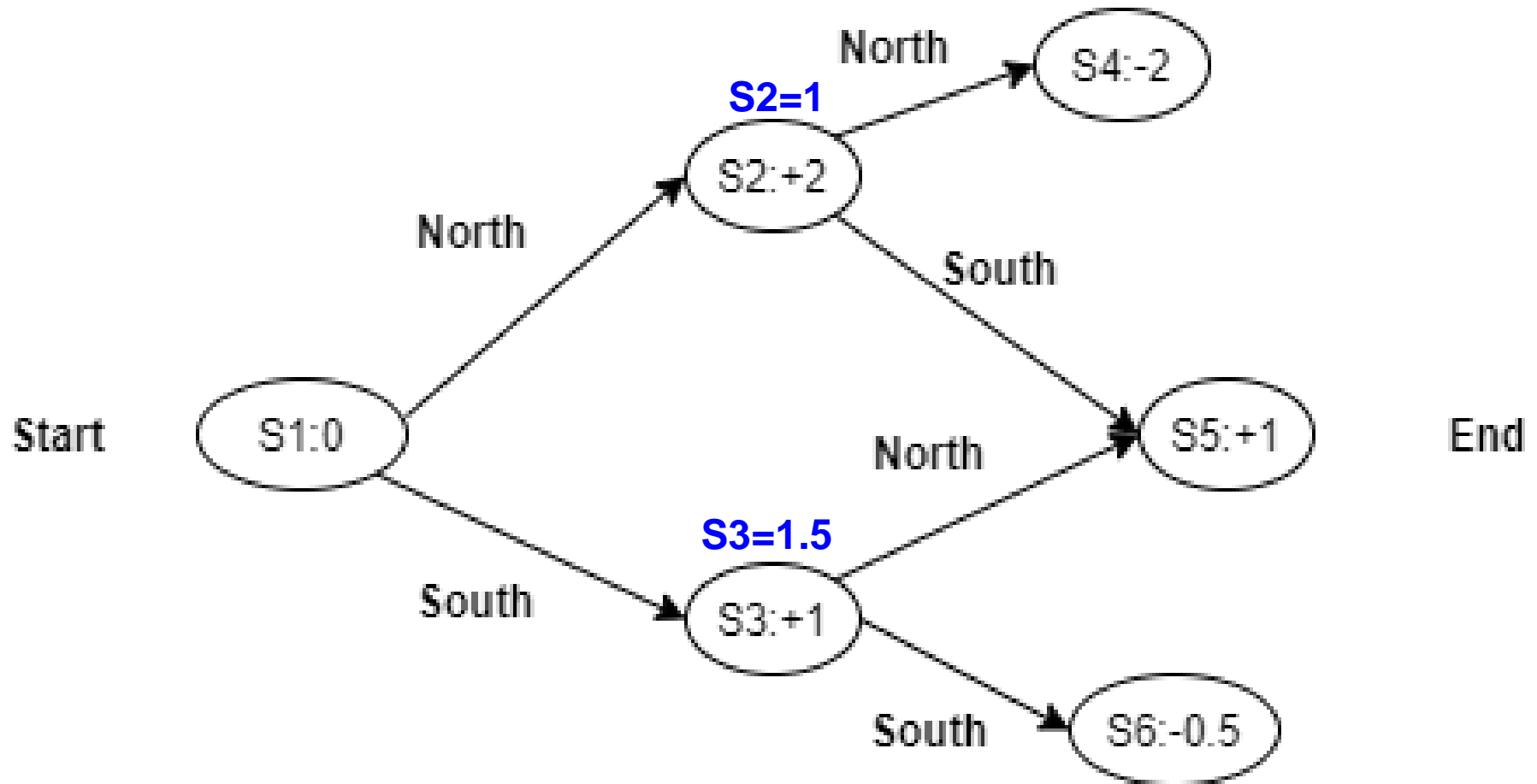
State	Action	V[S]	Total value (Reward + Sum of Value for each Action)
S1	North	$V[S] = (0.8 * 2 + 0.2 * 1) = 1.8$	
	South	$V[S] = (0.2 * 2 + 0.8 * 1) = 1.2$	
			$0 + 1.8 + 1.2 = 3$
S2	North	$V[S] = (0.8 * -2 + 0.2 * 1) = -1.4$	
	South	$V[S] = (0.2 * -2 + 0.8 * 1) = 0.4$	$2 - 1.4 + 0.4 = 1$
S3	North	$V[S] = 0.8 * 1 + 0.2 * -0.5 = 0.7$	
	South	$V[S] = 0.2 * 1 + 0.8 * -0.5 = -0.2$	$1 + 0.7 - 0.2 = 1.5$

**Before second iteration the values are**  $V[S1] = 0$ ;  $V[S2] = 1$ ;  $V[S3] = 1.5$ ;  $V[S4] = -2$ ;  $V[S5] = 1$ ;  $V[S6] = -0.5$ .

**After, second iteration ,values for each state could be changed as**  $V[S1] = 3$ ;  $V[S2] = 1$ ;  $V[S3] = 1.5$ ;  $V[S4] = -2$ ;  $V[S5] = 1$ ;  $V[S6] = -0.5$ .

S1: Start state      S2: State of landing in gold Island  
S3: State of landing in silver Island    S4: State of landing in Bermuda  
Triangle Island      S5: State of landing in destination Island  
S6: State of landing in prison Island

### After second iteration



## Policy Improvement for second iteration:

State	Action	V[S]	Max Action
S1	North	$V[S] = 0.8 * 1 + 0.2 * 1.5 = 1.1$	
	South	$V[S] = 0.2 * 1 + 0.8 * 1.5 = 1.4$	
			South
S2	North	$V[S] = 0.8 * -2 + 0.2 * 1 = -1.4$	
	South	$V[S] = 0.2 * -2 + 0.8 * 1 = 0.4$	
			South
S3	North	$V[S] = 0.8 * 1 + 0.2 * -0.5 = 0.7$	North
	South	$V[S] = 0.2 * 1 + 0.8 * -0.5 = -0.2$	

The policy obtained based on above table is as follows:  $P = \{S, S, N\}$  ;  
The policy is different from previous iteration of  $P = \{N, S, N\}$  ;  
Repeat the iteration when policy never change.



### Third iteration:

Policy Evaluation for third iteration:

State	Action	V[S]	Total value (Reward +Sum of Value for each Action)
S1	North	$V[S] = (0.8 * 1 + 0.2 * 1.5) = 1.1$	
	South	$V[S] = (0.2 * 1 + 0.8 * 1.5) = 1.4$	
			$0 + 1.1 + 1.4 = 2.5$
S2	North	$V[S] = (0.8 * -2 + 0.2 * 1) = -1.4$	
	South	$V[S] = (0.2 * -2 + 0.8 * 1) = 0.4$	$2 - 1.4 + 0.4 = 1$
S3	North	$V[S] = 0.8 * 1 + 0.2 * -0.5 = 0.7$	
	South	$V[S] = 0.2 * 1 + 0.8 * -0.5 = -0.2$	$1 + 0.7 - 0.2 = 1.5$

Values for each state could be summarized as below:

$$V[S1] = 2.5; V[S4] = -2$$

$$V[S2] = 1; V[S5] = 1$$

$$V[S3] = 1.5; V[S6] = -0.5$$

## Policy Improvement for third iteration:

State	Action	V[S]	Max Action
S1	North	$V[S] = 0.8 * 1 + 0.2 * 1.5 = 1.1$	
	South	$V[S] = 0.2 * 1 + 0.8 * 1.5 = 1.4$	
			South
S2	North	$V[S] = 0.8 * -2 + 0.2 * 1 = -1.4$	
	South	$V[S] = 0.2 * -2 + 0.8 * 1 = 0.4$	
			South
S3	North	$V[S] = 0.8 * 1 + 0.2 * -0.5 = 0.7$	
	South	$V[S] = 0.2 * 1 + 0.8 * -0.5 = -0.2$	
			North

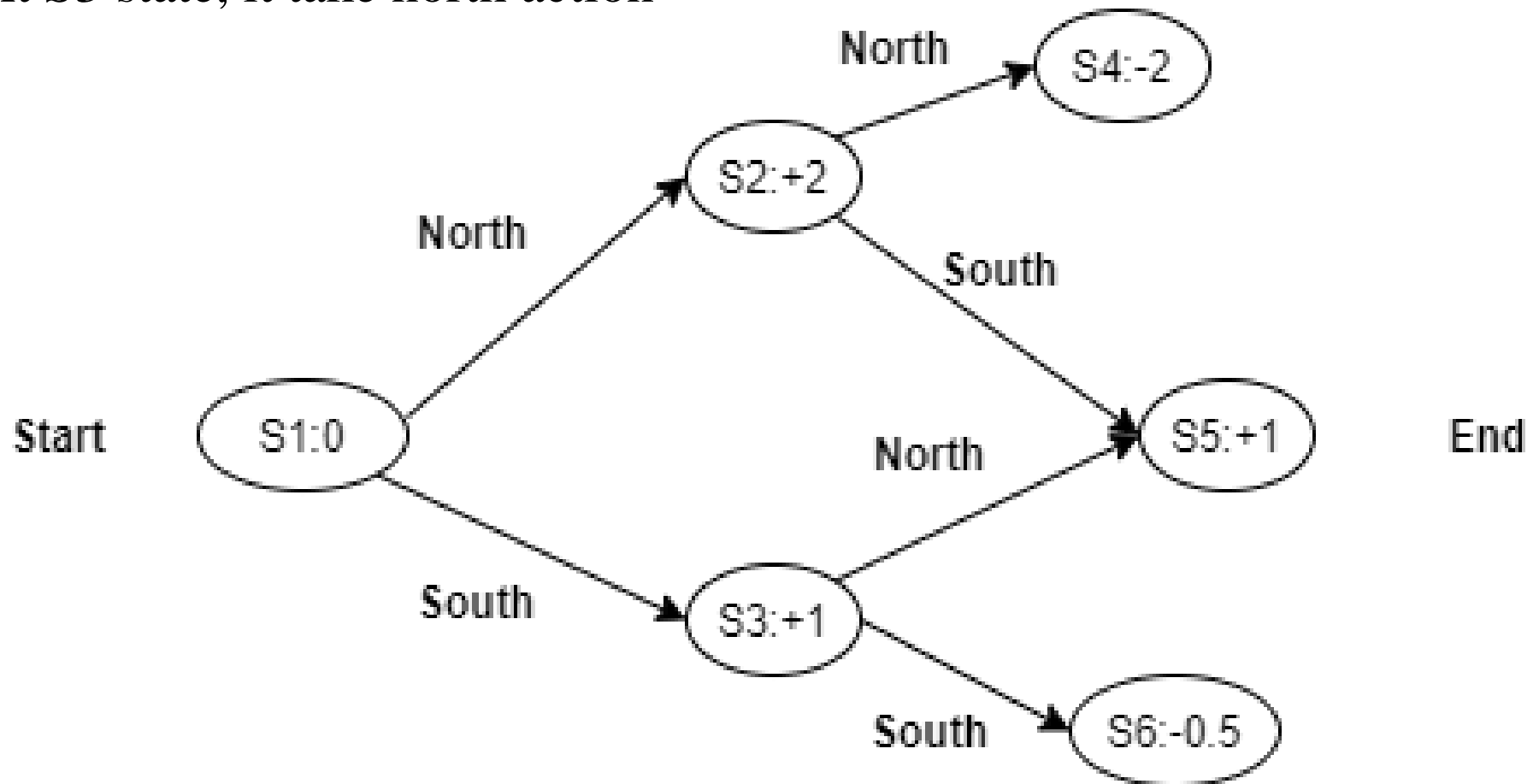
The policy obtained based on above table is as follows:

$P = \{S, S, N\}$

If compare this policy, to the policy obtained in second iteration.

**Observe that policies did not change, which implies algorithm has converged and this is the optimal policy.**

The policy obtained is  $P = \{S, S, N\}$   
The policy indicates  $S1 = S, S2 = S, S3 = N$   
The policy maps the states with actions.  
At  $S1$  state, it take south action  
At  $S2$  state, it take south action  
At  $S3$  state, it take north action



# Temporal based Reinforcement Learning Algorithms

The main used algorithms are:

**Q-Learning:** Q-learning is an **Off policy temporal based RL algorithm**. **These methods are the way of comparing temporally successive predictions**. It learns the value function  $Q(S, a)$ , which means to take action "a" at a particular state "s".

**State Action Reward State action (SARSA):** SARSA stands for **State Action Reward State action**, which is an **on-policy temporal based 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  $Q^\pi(s, a)$  for the selected current policy  $\pi$  and all pairs of (s,a).

In **off-policy method**, the **value of the best next action is used without using the policy**. In **on-policy method**, the **policy is used to determine the next action**. The on-policy version of Q learning is the Sarsa algorithm.

## Q-Learning Algorithm

for each  $s, a$  initialize the table entry  $Q(s, a)$  to zero.

Observe the current state  $s$

Do forever :

- \* Select an action and execute it

- \* Receive immediate reward  $r$

- \* Observe the new state  $s'$

- \* Update the table for  $Q(s, a)$  as follows:

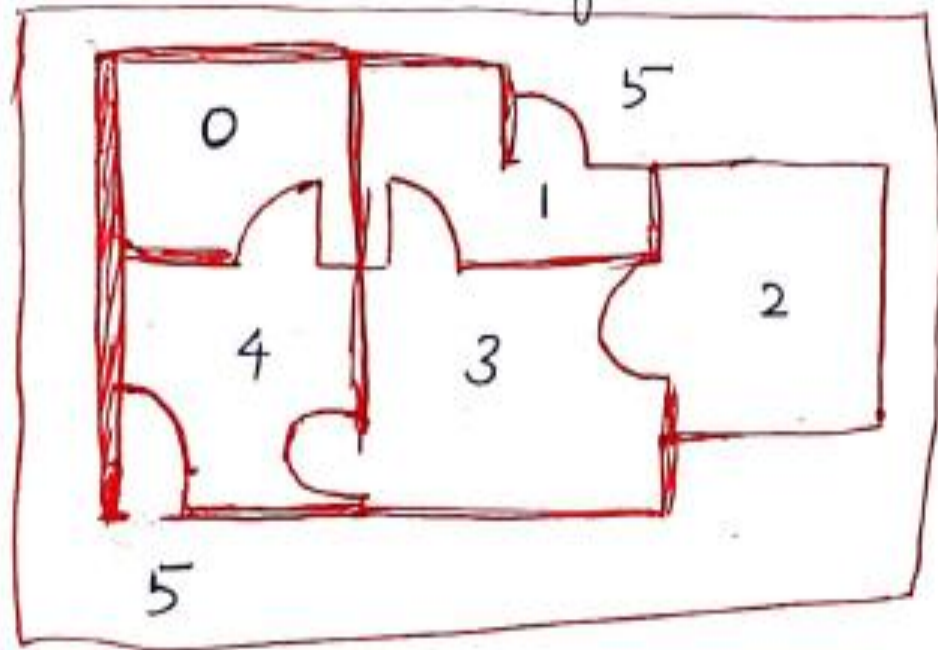
policy  $\rightarrow Q(s, a) \leftarrow r + \underbrace{\lambda \max_{a'} [Q(s', a')]}_{\text{Bellman equation}}$

- $s \leftarrow s'$

Until goal state is reached.



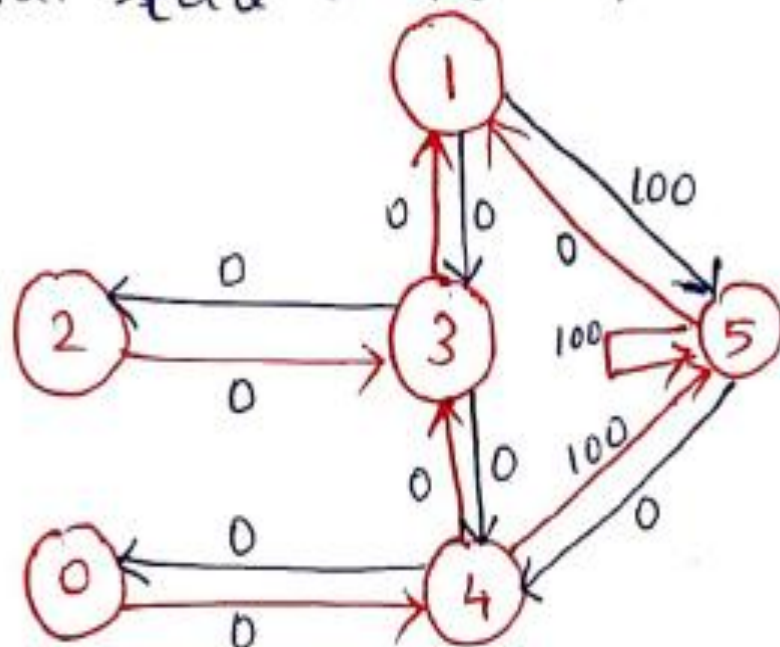
Problem: There are 5 rooms in a building connected by doors as shown in the figure. The rooms are numbered from 0 to 4. The outside of the building is big room and numbered as 5. The doors 1 and 4 lead into the building room 5 which is the goal state. Apply Q-learning algorithm?



There are 6 rooms as represented as 6 states and there are 6 doors represented as actions. Represent the rooms on a graph,

each room as a node and each door as a link. Assume discount factor as 0.8 and the

Initial state as Room 1.



Place the instant reward values into the reward table

'matrix  $R$ ' table states are in rows, rewards as columns. The -1 in the table represent null values [There is no link b/w nodes].

\* Initialize 'Q' table as zeros.

$R =$

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

reward table

$Q =$   
(Q Table)

State	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0



\* The initial state is room 1 and look at the second row of 'matrix R'.

\* There are two possible actions for the current state 1; go to state 3 or go to state 5.

\* By random selection, select as '5' as action.

\* What would happen if the agent is in state 5 (next state)

\* Look at the sixth row of the reward matrix R (state 5)

\* It has 3 possible actions; go to state 1, 4, 5

Apply Bellman equation,

$$Q(\text{state}, \text{action}) = R(\text{state}, \text{action}) + \lambda * \max_{\text{all actions}} [Q(\text{next state}, \text{all actions})]$$

$$Q(1, 5) = R(1, 5) + 0.8 * \max [Q(5, 1), Q(5, 4), Q(5, 5)]$$

See from Q table

$$= 100 + 0.8 * 0 = 100$$

update in Q table as  $Q(1, 5) = 100$



- \* The next state 5 now becomes the current state.
- \* Because 5 is the goal state, one episode is finished.

actions

state	0	1	2	3	4	5
Q = 0	0	0	0	0	0	0
1	0	0	0	0	0	100
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

- \* For the next episode, randomly choose the initial state 3 and look at 'matrix R'.
- \* There are three possible actions for the current state 3; go to state 1, go to state 2, or go to state 4.
- \* What would happen if the agent is in state 1.
- \* Look at the second row of 'matrix R' (next state)

\* It has 2 possible actions : go to state 3 or state 5

\* Compute Q value

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

$$Q(3, 1) = R(3, 1) + 0.8 * \max [Q(1, 3), Q(1, 5)]$$

$$= 0 + 0.8 * \max [0, 100] = 80$$

Q =

State	action					
	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	100
3	0	80	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0

\* If the agent learns more through further episodes it will finally reach convergence values in matrix Q.

$$Q(1,3) = R(1,3) + 0.8 * \max [ (3,1), (3,4) ]$$

$$= 0 + 0.8 * \max [ (80, 0) ] = 0.64$$

$$Q(2,3) = R(2,3) + 0.8 * \max [ (3,1), (3,4), (3,2) ]$$

$$= 0 + 0.8 * \max [ 80, 0, 0 ] = 0.64$$

$$Q(3,2) = R(3,2) + 0.8 * \max [ 2, 3 ]$$

$$= 0 + 0.8 * 0.64 = 0.51$$

$$Q(3,4) = R(3,4) + 0.8 * \max [ (4,3), (4,5) ]$$

$$= 0 + 0.8 * \max [ (0, 100) ] = 80$$

$$Q(4,3) = R(4,3) + 0.8 * \max [ (3,1), (3,4) ]$$

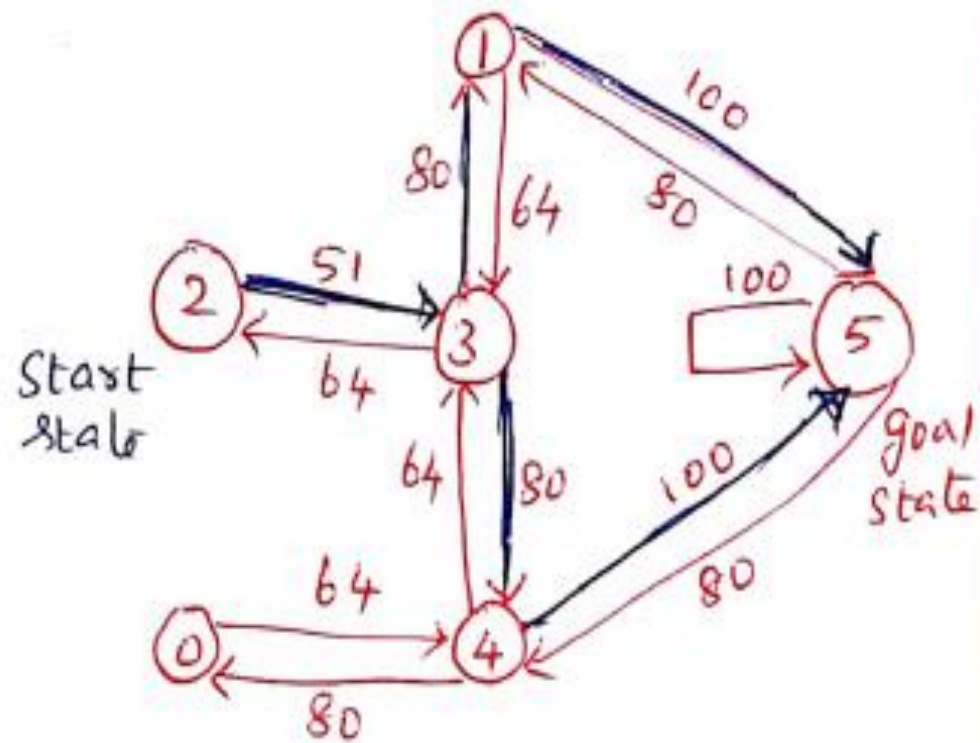
$$= 0 + 0.8 * \max [ (80, 80) ] = 0.64$$



The final 'Q' matrix is,

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

Tracing the best sequences of states is as simple as following the links with the highest values at each state.



# Q Learning and SARSA Learning

```
Initialize all  $Q(s, a)$  arbitrarily
For all episodes
  Initialize  $s$ 
  Repeat
    Choose  $a$  using policy derived from  $Q$ , e.g.,  $\epsilon$ -greedy
    Take action  $a$ , observe  $r$  and  $s'$ 
    Update  $Q(s, a)$ :
       $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma \max_{a'} Q(s', a') - Q(s, a))$ 
       $s \leftarrow s'$ 
  Until  $s$  is terminal state
```

Figure 18.5 Q learning, which is an off-policy temporal difference algorithm.

```
Initialize all  $Q(s, a)$  arbitrarily
For all episodes
  Initialize  $s$ 
  Choose  $a$  using policy derived from  $Q$ , e.g.,  $\epsilon$ -greedy
  Repeat
    Take action  $a$ , observe  $r$  and  $s'$ 
    Choose  $a'$  using policy derived from  $Q$ , e.g.,  $\epsilon$ -greedy
    Update  $Q(s, a)$ :
       $Q(s, a) \leftarrow Q(s, a) + \eta(r + \gamma Q(s', a') - Q(s, a))$ 
       $s \leftarrow s', a \leftarrow a'$ 
  Until  $s$  is terminal state
```

Figure 18.6 Sarsa algorithm, which is an on-policy version of Q learning.

## Other Types of Learning

- Representation learning
- Active learning
- Instance-based Learning
- Ensemble learning
- Regularization algorithm

### Representation learning

**Another name for representation learning is feature learning.** The main **objective of representation learning is to find an appropriate representation of data based on features to perform a machine learning task.** Representation learning has become a field in itself because of its popularity.

**Example for Representation Learning:** Classify/identify the types of triangles . Find unique characteristics of the type of triangle given as input.

S. No	Types of triangle	Characteristics
1	Acute triangle	A triangle in which all three angles are less than $90^\circ$
2	Obtuse triangle	An obtuse triangle is a triangle in which one of the angles is greater than $90^\circ$ .
3	Right triangle	A right triangle is triangle with an angle of $90^\circ$ .
4	Scalene triangle	A triangle with three unequal sides.
5	Isosceles triangle	An isosceles triangle is a triangle with (at least) two equal sides.
6	Equilateral triangle	An equilateral triangle is a triangle with all three sides of equal length.
7	Equiangular triangle	An equiangular triangle is a triangle, which has three equal angles.

The Representation Learning Model work as,

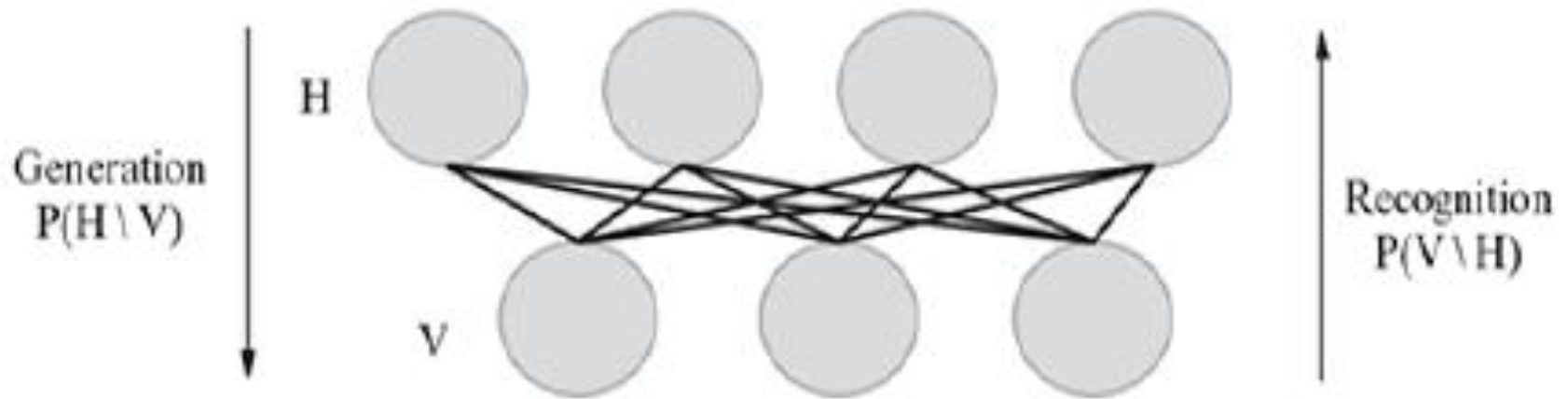
**Input** – Triangular image

**Representation** – Three angles, length of sides.

**Model** – It gets an input representation (angles, length of sides) and applies rules as per Table , to detect the type of triangle.

- What happens when the input shapes like square, rectangle, circle, or all sort of shapes instead of a triangle? What if the image contains both a triangle and a circle? Designing to adapt to this kind of features requires deep domain expertise for real-world applications.
- Deep Learning goes one step further and learns/tries to learn features on its own. To feed in an image as input and the system learn features like human beings do.
- Representation learning are most widely used in words, speech recognition, signal processing, music, and image identification.

## Generation and Recognition in Representation Learning





**V represents the input data, and H represents the causes. When the causes (H) explains the data (V) , it is called as Recognition. When unknown causes (H) are combined, it can also generate the data (V), it is called as generative (or) Generation.**

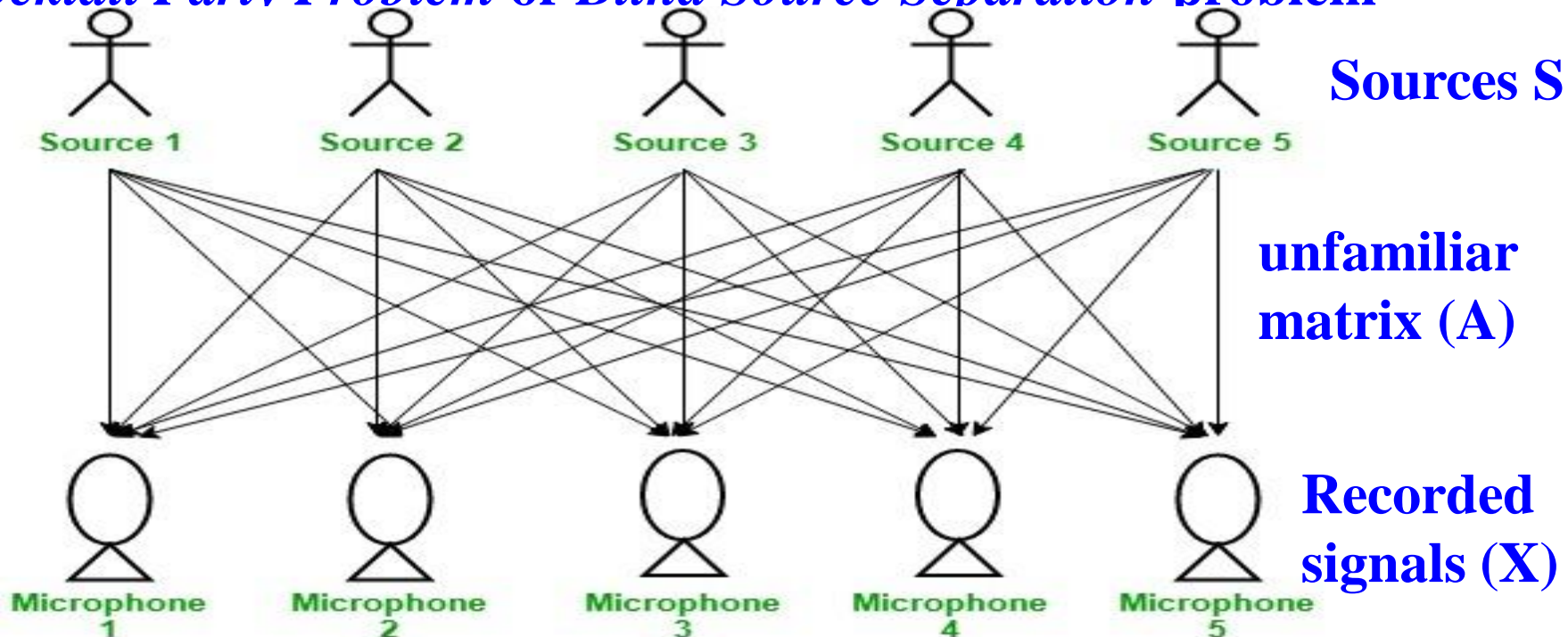
Representation learning can be either supervised or unsupervised. The different types of Representation learning are,

1. Supervised neural networks and multilayer perceptron
2. Independent component analysis (unsupervised)
3. Autoencoders (unsupervised) and
4. Various forms of clustering

## Independent component analysis(ICA)

ICA is used primarily for separating unknown source signals from their observed linear mixtures after the linear mixture with an unfamiliar matrix (A) . Nil information is known about the sources or the mixing process except that there are *N different recorded* mixtures. The job is to recuperate a version(U) of the sources (S) which are identical except for scaling, by finding a square matrix (W) specifying filters that linearly invert the mixing process, i.e.  $U = WX$ .

### *Cocktail Party Problem or Blind Source Separation problem*



**Recorded  
signals (X)**

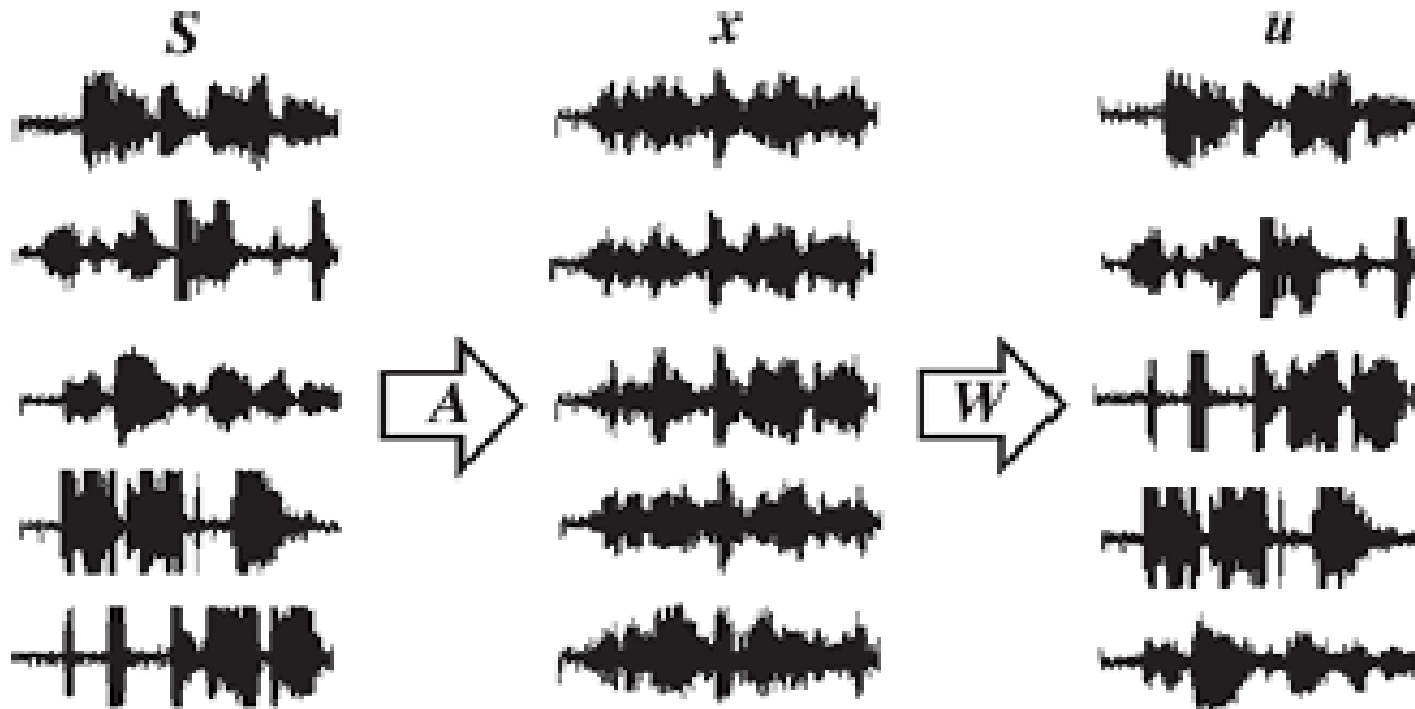
**Matrix (W)**

**recuperate version(U)**

$$X = SA \rightarrow S = A^{-1} X$$

$$W = A^{-1}$$

$$U = WX = A^{-1} S A = S$$



**FIG. 11.2** Independent component analysis

## Autoencoders

Autoencoders belong to the neural network family. The neural network's (autoencoder) target output is its input ( $x$ ) in a different form ( $x'$ ). In Autoencoders, the dimensionality of the input is equal to the dimensionality of the output, and essentially

$$x' = x. \quad x' = \text{Decode}(\text{Encode}(x))$$

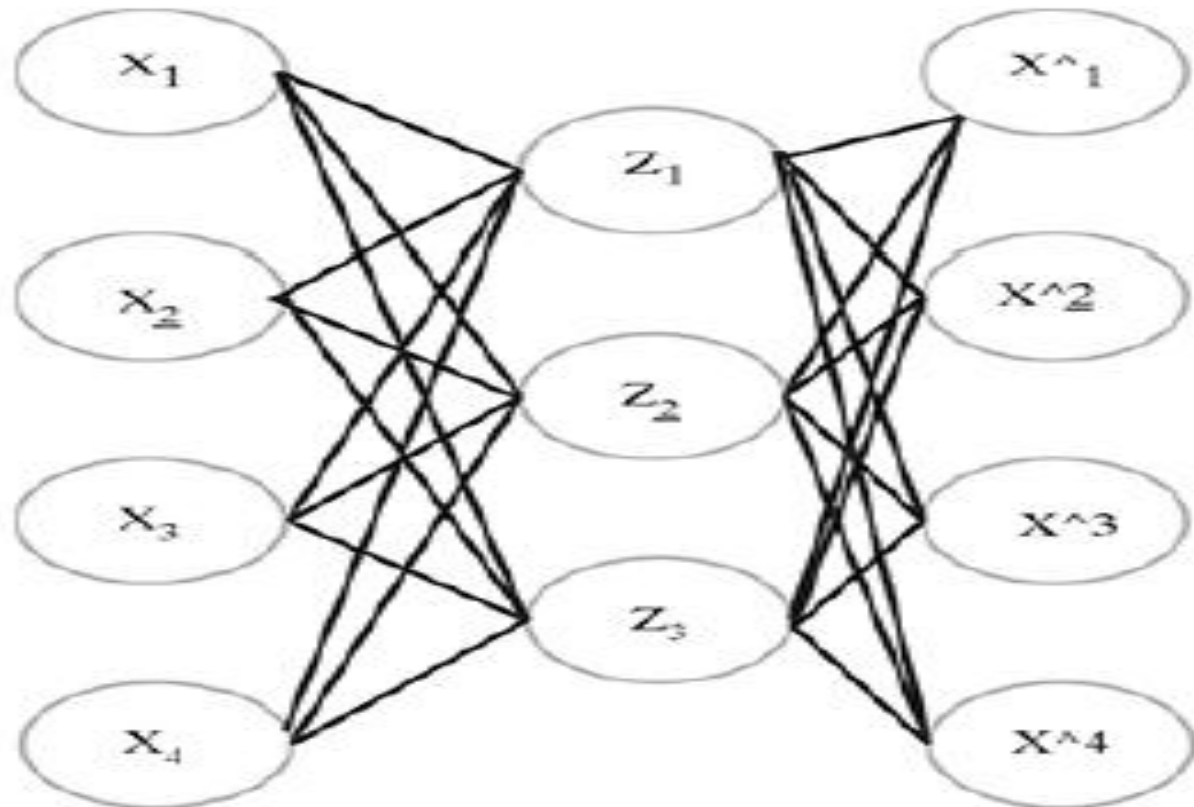


FIG. 11.3 Autoencoders

To get the value from the hidden layer, multiply the (input to hidden) weights by the input. To get the value from the output, multiply (hidden to output) weights by the hidden layer values.

$$Z = f(WX)$$

$$Y = g(VZ)$$

$$\text{So } Y = g(Vf(WX)) = VWX$$

## Heuristics for Autoencoders

Autoencoder training algorithm can be summarized as below:

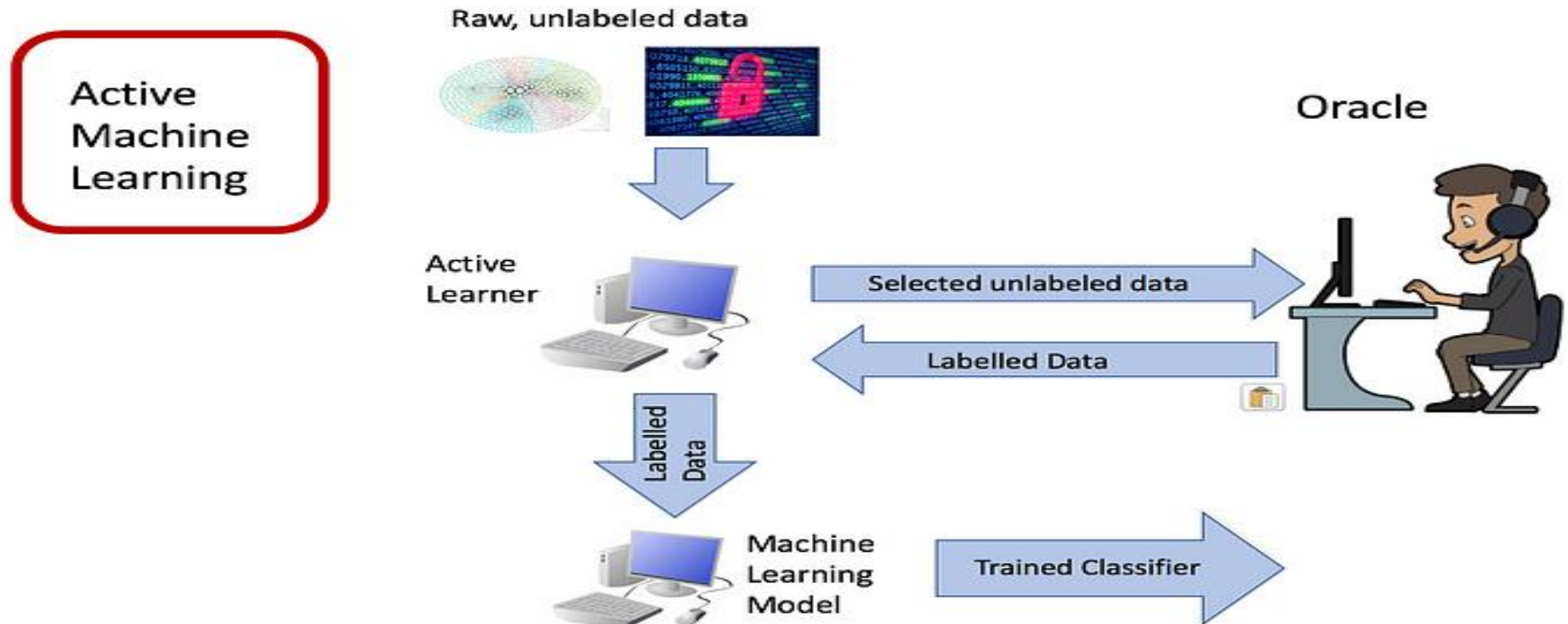
For each input  $x$

1. Do a feed-forward pass to compute activations at all hidden layers, then at the output layer to obtain an output  $X$
2. Measure the deviation from output  $X$  *to the input*  $X$
3. Back propagate the error through the net and perform weight updates.

# ACTIVE LEARNING

Active learning is a type of semi-supervised learning in which a learning algorithm can interactively query the user (or some other information source) to obtain the desired outputs at new data points. The information source is also called teacher or oracle.

This approach is used to construct a high-performance classifier **while keeping the size of the training dataset to a minimum by actively selecting the valuable data points.**



## Where should apply active learning?

- Small amount or a huge amount of dataset.
- Annotation of the unlabeled dataset cost human effort, time, and money.
- Limited processing power.

Let  $P$  be the population set of all data under consideration. During each iteration,  $P$  (total population) is broken up into three subsets.

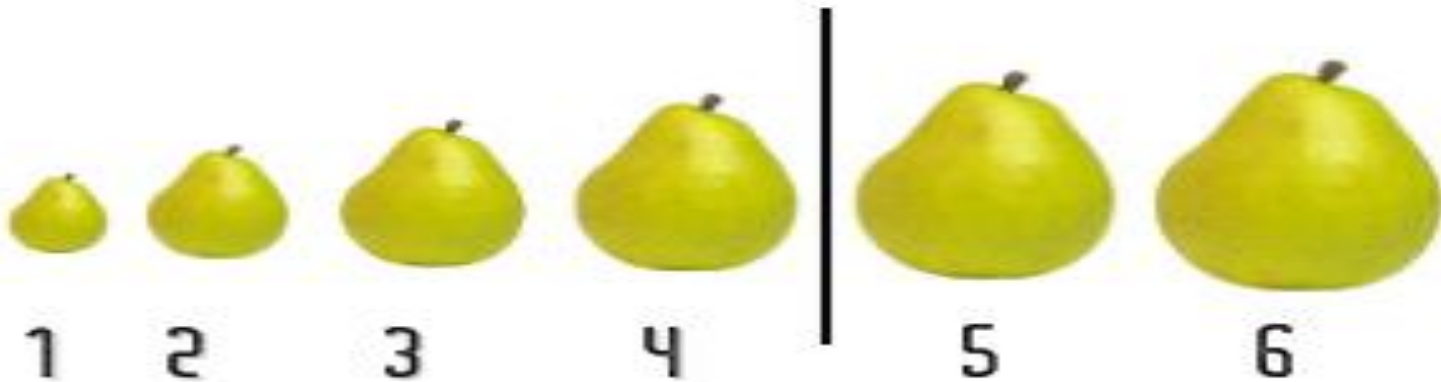
1.  $P(K, i)$ : Data points where the label is known (K).
2.  $P(U, i)$ : Data points where the label is unknown (U)
3.  $P(C, i)$ : A subset of  $P(U, i)$  that is chosen (C) to be labelled.

Active learning concentrates on identifying the best method  $P(C, i)$  to choose (C) the data points.

## Heuristics for active learning

1. Start with a pool of unlabelled data  $P(U, i)$
2. Pick a few points at random and get their labels  $P(C, i)$
3. Repeat

**Example:** On a certain planet, there are various fruits of different size(1-5), some of them are poisonous and others don't. The only criteria to decide a fruit is poisonous or not is ,its size. Train the classifier that predicts the given fruit is poisonous or not. A fruit with size 1 is not poisonous, the fruit of size 5 is poisonous and after a particular size, all fruits are poisonous.



There are two ways to train the model.

1. check each and every size of the fruit, which consumes time and resources.
2. Apply the binary search and find the transition point (decision boundary). This approach uses fewer data and gives the same results as of linear search.



## Algorithm :

1. Train classifier with the initial training dataset
2. Calculate the accuracy
3. While( $\text{accuracy} < \text{desired accuracy}$ )
4. Select the most valuable data points (in general, points close to decision boundary)
5. Query that data point/s (ask for a label) from human oracle
6. Add that data point/s to initial training dataset
7. Re-train the model
8. Re-calculate the accuracy

## Active learning query strategies

Few logics for determining **which data points should be labelled are**

1. Uncertainty sampling
2. Query by committee
3. Expected model change
4. Expected error reduction
5. Variance reduction

**Uncertainty sampling:** In this method, the active learning algorithm first tries to label the points for which the current model is least specific on the correct output.

**Query by committee:** A range of models are trained on the current labelled data, and vote on the output for unlabelled data; label those points for which the ‘committee’ disagrees the most.

**Expected model change:** In this method, the active learning algorithm first tries to label the points that would most change the existing model itself.

**Expected error reduction:** In this method, the active learning algorithm first tries to label the points that would mostly reduce the model’s generalization error.

**Variance reduction:** In this method, the active learning algorithm first tries to label the points that would reduce output variance

# INSTANCE-BASED LEARNING

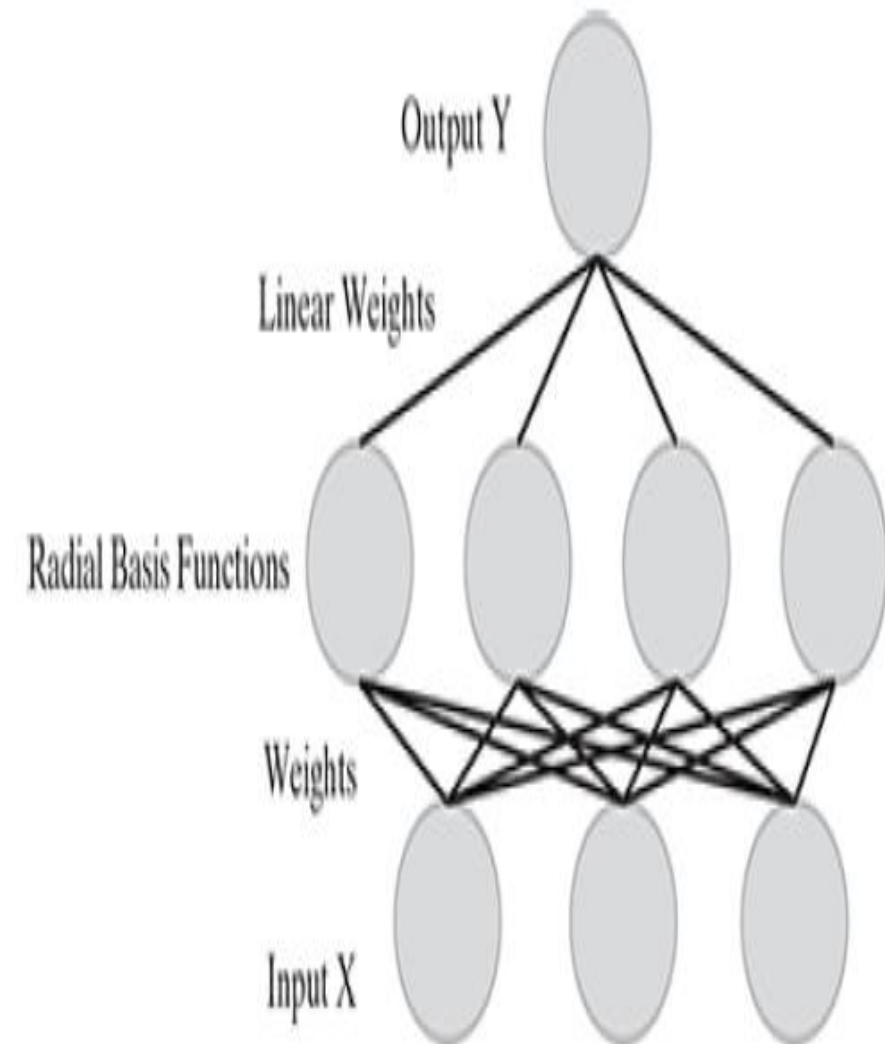
It is a family of learning algorithms that, **instead of performing explicit generalization, compare new problem instances with instances seen in training, which have been stored in memory**. The computation is postponed until a new instance is observed, these algorithms are sometimes referred to as "lazy". It is called instance-based because it constructs hypotheses directly from the training instances themselves. It is also called as **memory-based learning**.

Examples of instance-based learning include K-nearest neighbour algorithm (K-NN), kernel machines (support vector machine, PCA), and radial basis function (RBF) networks.

All these algorithms store already known instances and compute distances or similarities between this instance and the training instances to make the final decision.

## Radial basis function

Radial basis function (RBF) networks typically have three layers: one input layer, one hidden layer with a non-linear RBF activation function, and one linear output layer.



An input vector( $X$ ) is used as input to all radial basis functions, each with different parameters. The input layer ( $X$ ) is merely a fan-out layer, and no processing happens here. The second layer (hidden) performs radial basis functions, which are the non-linear mapping from the input space (Vector  $X$ ) into a higher order dimensional space. The output of the network ( $Y$ ) is a linear combination of the outputs from radial basis functions. If pattern classification is required (in  $Y$ ), then a hard-limiter or sigmoid function could be placed on the output neurons to give 0/1 output values

The **distinctive part of the radial basis function network (RFFN) is the procedure implemented in the hidden layer.** The Gaussian function is the most commonly used radial basis function. In an RBF network, *r* is the distance from the cluster centre.

Space (distance) computed from the cluster centre is usually the Euclidean distance. For each neuron that is part of the hidden layer, the weights represent the coordinates of the centre of the cluster. Therefore, when that neuron receives an input pattern, *X*, the distance is found using the following equation:

$$r_j = \sqrt{\sum_{i=1}^n (x_i - w_{ij})^2}$$

The **variable sigma,  $\sigma$ , denotes the width or radius of the bell-shape** and is to be determined by calculation.

$$(hidden_{unit})\Phi_j = \exp\left(-\frac{\sum_{i=1}^n (x_i - w_{ij})^2}{2\sigma^2}\right)$$

**The output  $y(t)$  is a weighted sum of the outputs of the hidden layer, given by**

$$y(t) = \sum_{i=1}^n w_i \phi(\|u(t) - c_i\|),$$

Where  $u(t)$  is the input

$\phi(\cdot)$  is an arbitrary non-linear radial basis function

$\|\cdot\|$  denotes the norm that is usually assumed to be Euclidean

$c$  are the known centres of the radial basis functions

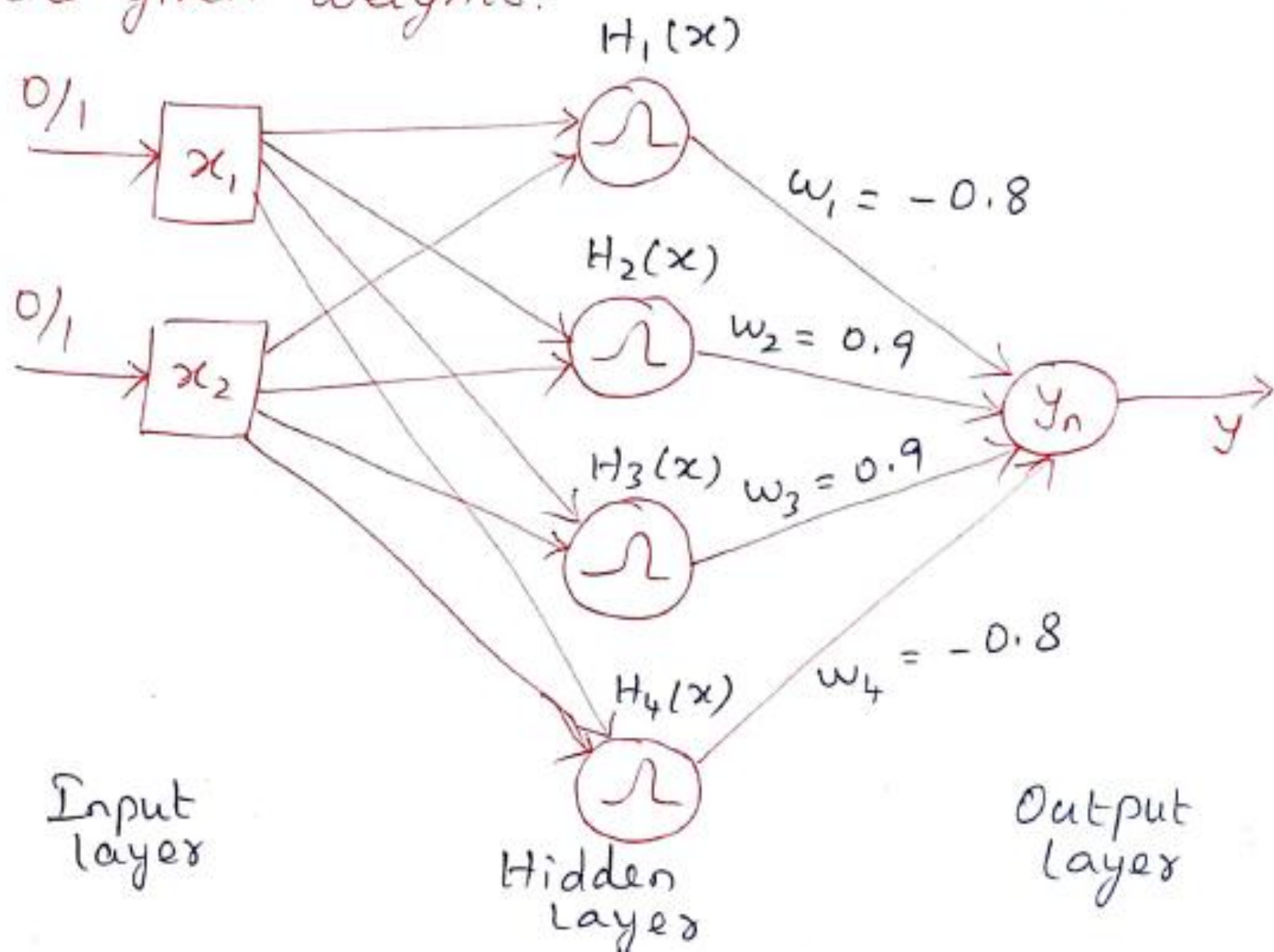
$w$  are the weights of the function

**The most commonly used radial function is the Gaussian radial filter**, which in case of a scalar input is

$$h(x) = \exp\left(-\frac{(x - c)^2}{2\beta^2}\right)$$

**Where  $c$  is the centre and equal to 0  
 $\beta$  is the radius (width) and equal to 1  
in Gaussian RBF**

Problem! Construct a RBFNN for XOR gate with the given weights.





The 4 hidden layer neurons with gaussian RBF.

$$H_1(x) = e^{-\frac{(x-c_1)^2}{2\sigma^2}} ; c_1 = (0, 0)$$

$$H_2(x) = e^{-\frac{(x-c_2)^2}{2\sigma^2}} ; c_2 = (0, 1)$$

$$H_3(x) = e^{-\frac{(x-c_3)^2}{2\sigma^2}} ; c_3 = (1, 0)$$

$$H_4(x) = e^{-\frac{(x-c_4)^2}{2\sigma^2}} ; c_4 = (1, 1)$$

For input pattern (0, 0) ← This is  $x$  value.

Euclidean distance of  $x$  from  $c_1 = (0, 0)$   
 $= (0-0)^2 + (0-0)^2 = 0$

$$H_1(x) = e^{-\frac{(x-c_1)^2}{2\sigma^2}} = e^{-\frac{0}{2}} = 1.0$$

Euclidean distance of  $x$  from  $c_2 = (0, 1)$

$2\sigma^2 = 2$   
for RBF  
 $\sigma = 1$   
(Gaussian)



$$= (0-0)^2 + (0-1)^2 = 1$$

$$H_2(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} = e^{-1/2} = 0.6$$

distance of  $x$  from  $c_3 = (1, 0)$

$$= (0-1)^2 + (0-0)^2 = 1$$

$$H_3(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} = e^{-1/2} = 0.6$$

distance of  $x$  from  $c_4 = (1, 1)$

$$= (0-1)^2 + (0-1)^2 = 2$$

$$H_4(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} = e^{-2/2} = 0.4$$

The i/p at the output layer is,

$$\begin{aligned} \sum_{i=1}^m w_i H_i(x) &= w_1 H_1(x) + w_2 H_2(x) + w_3 H_3(x) + w_4 H_4(x) \\ &= (-0.8 * 1) + (0.9 * 0.6) + (0.9 * 0.6) + \\ &\quad (-0.8 * 0.4) = -0.04 \end{aligned}$$

for input pattern  $(0,1) \leftarrow x$

distance of  $x$  from  $c_1 = (0,0)$

$$= (0-0)^2 + (1-0)^2 = 1 \quad ; \quad H_1(x) = e^{\frac{-(x-c)^2}{2\sigma^2}} = e^{-1/2} = 0.6$$

distance of  $x$  from  $c_2 = (0,1)$

$$= (0-0)^2 + (1-1)^2 = 0 \quad ; \quad H_2(x) = e^{\frac{-(x-c)^2}{2\sigma^2}} = e^{-0/2} = 1.0$$

distance of  $x$  from  $c_3 = (1,0)$

$$= (0-1)^2 + (1-0)^2 = 2 \quad ; \quad H_3(x) = e^{\frac{-(x-c)^2}{2\sigma^2}} = e^{-2/2} = 0.4$$

distance of  $x$  from  $c_4 = (1,1)$

$$= (0-1)^2 + (1-1)^2 = 1 \quad ; \quad H_4(x) = e^{\frac{-(x-c)^2}{2\sigma^2}} = e^{-1/2} = 0.6$$

The i/p at the output layer is,

$$\begin{aligned} \sum_{i=1}^m w_i H_i(x) &= w_1 H_1(x) + w_2 H_2(x) + w_3 H_3(x) + w_4 H_4(x) \\ i=1 \quad &= (-0.8 * 0.6) + (0.9 * 1) + (0.9 * 0.4) + (-0.8 * 0.6) \\ &= 0.3 \end{aligned}$$

For input pattern  $(1, 0) \leftarrow x$

distance of  $x$  from  $c_1 = (0, 0)$

$$= (1-0)^2 + (0-0)^2 = 1 \quad ; \quad H_1(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} = e^{-1/2} = 0.6$$

distance of  $x$  from  $c_2 = (0, 1)$

$$= (1-0)^2 + (0-1)^2 = 2 \quad ; \quad H_2(x) = e^{-2/2} = 0.4$$

distance of  $x$  from  $c_3 = (1, 0)$

$$= (1-1)^2 + (0-0)^2 = 0 \quad ; \quad H_3(x) = e^{-0/2} = 1.0$$

distance of  $x$  from  $c_4 = (1, 1)$

$$= (1-1)^2 + (0-1)^2 = 1 \quad ; \quad H_4(x) = e^{-1/2} = 0.6$$

The input at the output layer is,

$$\begin{aligned} \sum_{i=1}^m w_i H_i(x) &= w_1 H_1(x) + w_2 H_2(x) + w_3 H_3(x) + w_4 H_4(x) \\ (2) \quad &= (-0.8 * 0.6) + (0.9 * 0.4) + (0.9 * 1) + \\ &\quad (-0.8 * 0.6) = 0.3 \end{aligned}$$



for input pattern  $(1, 1) \leftarrow x$

distance of  $x$  from  $c_1 = (0, 0)$

$$= (1-0)^2 + (1-0)^2 = 2 ; H_1(x) = e^{-\frac{(x-c)^2}{2\sigma^2}} = e^{-\frac{2}{2}} = 0.4$$

distance of  $x$  from  $c_2 = (0, 1)$

$$= (1-0)^2 + (1-1)^2 = 1 ; H_2(x) = e^{-1/2} = 0.6$$

distance of  $x$  from  $c_3 = (1, 0)$

$$= (1-1)^2 + (1-0)^2 = 1 ; H_3(x) = e^{-1/2} = 0.6$$

distance of  $x$  from  $c_4 = (1, 1)$

$$= (1-1)^2 + (1-1)^2 = 0 ; H_4(x) = e^{-0/2} = 1.0$$

The input at the output layer is,

$$\begin{aligned} \sum_{i=1}^m w_i H_i(x) &= w_1 H_1(x) + w_2 H_2(x) + w_3 H_3(x) + w_4 H_4(x) \\ &= (-0.8 \times 0.4) + (0.9 \times 0.6) + (0.9 \times 0.6) + (-0.8 \times 1.0) \\ &= -0.04 \end{aligned}$$

The values are tabulated as follows

Input		$H_1(x)$	$H_2(x)$	$H_3(x)$	$H_4(x)$	$\sum_{i=1}^m w_i H_i(x)$	Output
0	0	1.0	0.6	0.6	0.4	-0.04	0
0	1	0.6	1.0	0.4	0.6	0.3	1
1	0	0.6	0.4	1.0	0.6	0.3	1
1	1	0.4	0.6	0.6	1.0	-0.04	0

## **Pros and cons of instance-based learning method**

**Advantage:** Instance-based learning (IBL) methods are particularly well suited to problems where the target function is highly complex

**Disadvantages** : 1. The cost of classification of new instances can be high (a majority of the calculation takes place at this stage).  
2. Many IBL approaches usually study all characteristics of the instances leading to dimensionality-related problems.

# ENSEMBLE LEARNING ALGORITHM

Ensemble means collaboration/joint/group. Ensemble methods are models that contain many weaker models that are autonomously trained and whose forecasts are combined approximately to create the overall prediction model.

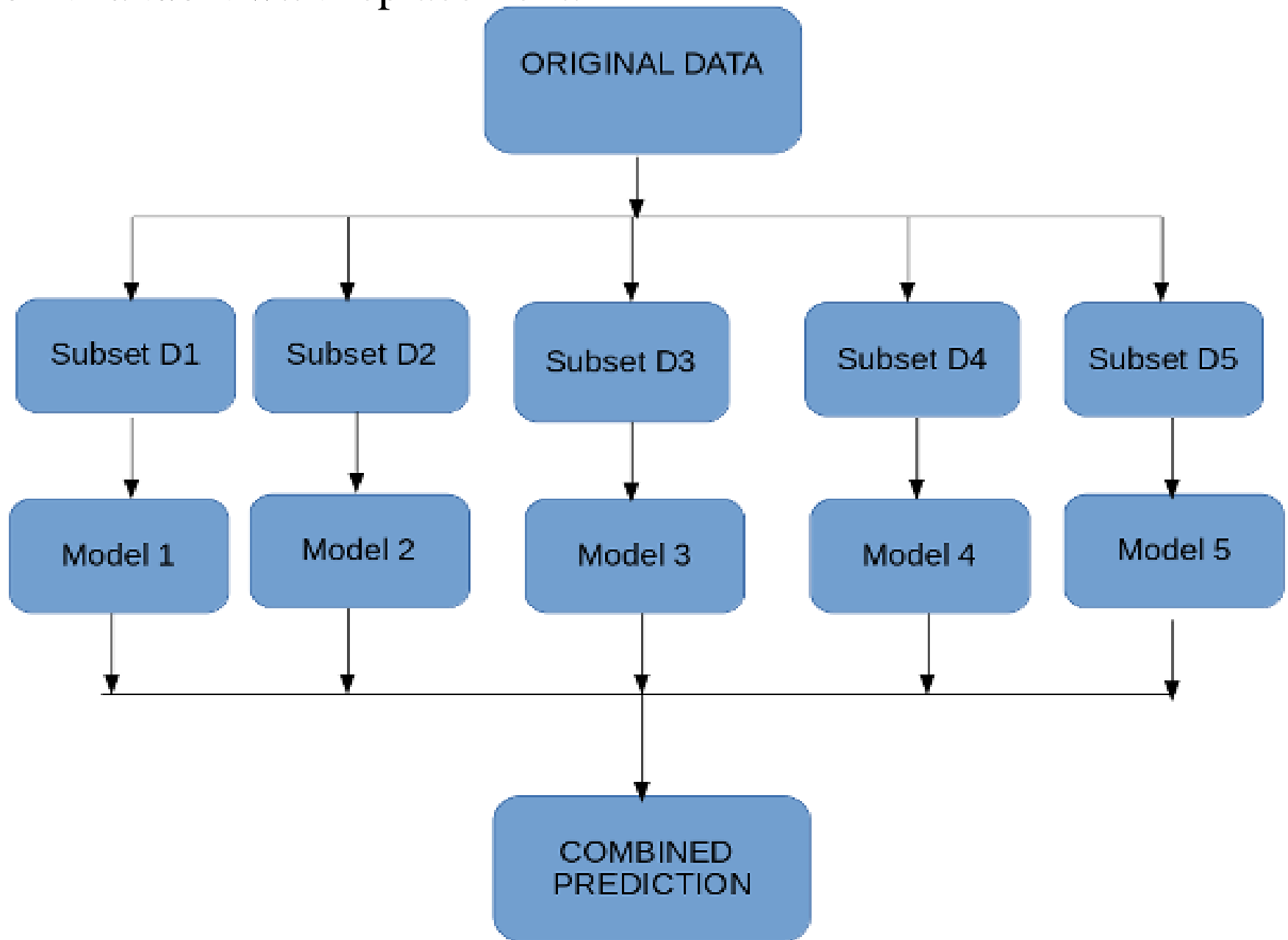
**Weaker models :** A model that achieves slightly better than 50 percent accuracy.

**The most popular ensemble learning algorithms are**

- Bootstrap Aggregation (bagging)
- Boosting
- AdaBoost
- Stacked Generalization (blending)
- Gradient Boosting Machines (GBM)
- Gradient Boosted Regression Trees (GBRT)
- Random Forest

**Bootstrap Aggregation (bagging):** Bootstrap Aggregation (bagging) works using the principle of the Bootstrap sampling method. Given a training data set  $D$  containing  $m$  examples, bootstrap drawing method

draws a sample of training examples  $D$  , by selecting  $m$  *examples* in *uniform random with replacement*.





It comprises two phases namely Training phase and Classification Phase.

### **Training Phase:**

1. Initialize the parameters
2.  $D = \{ \Phi \}$
3.  $H =$  the number of classification
4. For  $k = 1$  to  $h$
5. Take a bootstrap sample  $S_k$  from training set  $S$
6. Build the classifier  $D_k$  using  $S_k$  as a training set
7.  $D = D \cup D_i$
8. Return  $D$

### **Classification Phase:**

1. Run  $D_1, D_2, \dots, D_k$  on the input  $k$
2. The class with a maximum number of the vote is chosen as the label for  $X$ .

<b>Original Sample</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>
------------------------	----------	----------	----------	----------	----------	----------	----------	----------

<b>Bootstrap Sample 1</b>	<b>1</b>	<b>1</b>	<b>2</b>	<b>8</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>3</b>
<b>Bootstrap Sample 2</b>	<b>1</b>	<b>4</b>	<b>5</b>	<b>7</b>	<b>4</b>	<b>5</b>	<b>1</b>	<b>2</b>
<b>Bootstrap Sample 3</b>	<b>5</b>	<b>2</b>	<b>1</b>	<b>2</b>	<b>1</b>	<b>8</b>	<b>4</b>	<b>2</b>
<b>Bootstrap Sample 4</b>	<b>7</b>	<b>4</b>	<b>2</b>	<b>8</b>	<b>5</b>	<b>6</b>	<b>6</b>	<b>6</b>

The original sample has eight data points. All the Bootstrap samples also have eight data points. Within the same bootstrap sample, data points are repeated due to selection with the replacement.

## Boosting

Like bagging, boosting is another key ensemble based technique. Boosting is an iterative technique. It decreases the biasing error. If an observation was classified incorrectly, then it boosts the weight of that observation. In this type of ensemble, weaker learning models are trained on resampled data, and the outcomes are combined using a weighted voting approach based on the performance of different models. Adaptive boosting or AdaBoost is a special variant of a boosting algorithm. It is based on the idea of generating weak learners and learning slowly.

## Gradient boosting machines (GBM)

Gradient Boosting or GBM is another ensemble machine learning algorithm that works for both regression and classification problems. GBM uses the boosting technique, combining a number of weak learners to form a strong learner. Decision trees used as a base learner, each subsequent tree in series is **built on the errors calculated by the previous tree.**

**Example: Predict the age of a group of people using the below data:**

ID	Married	Gender	Current City	Monthly Income	Age (target)	Mean Age (prediction 1)	Residual 1
1	Y	M	A	51,000	35	32	3
2	N	F	B	25,000	24	32	-8
3	Y	M	A	74,000	38	32	6
4	N	F	A	29,000	30	32	-2
5	N	F	B	37,000	33	32	1

- 1.The mean age is assumed to be the predicted value for all observations in the dataset.
- 2. The residual errors are calculated using this mean prediction and actual values of age

3. A tree model is created using the errors calculated for the target variable. **The objective is to find the best split to minimize the error.**
4. The predictions by this model are combined with the predictions 1

ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)
1	35	32	3	3	35
2	24	32	-8	-5	27
3	38	32	6	3	35
4	30	32	-2	-5	27
5	33	32	1	3	35

5. This value calculated above is the new prediction.
6. **New errors are calculated using this predicted value and actual value.**
7. **Steps 2 to 6 are repeated till the maximum number of iterations is reached** (or error function does not change).

ID	Age (target)	Mean Age (prediction 1)	Residual 1 (new target)	Prediction 2	Combine (mean+pred2)	Residual 2 (latest target)
1	35	32	3	3	35	0
2	24	32	-8	-5	27	-3
3	38	32	6	3	35	-3
4	30	32	-2	-5	27	3
5	33	32	1	3	35	-2

# REGULARIZATION ALGORITHM

This is an extension made to another method (typically regression method) that penalizes models based on their complexity, favouring simpler models that are also better at generalizing. Regularization algorithms have been listed separately here because they are famous, influential, and generally simple modifications made to other methods.

**The most popular regularization algorithms are**

1. Ridge Regression,
2. Least Absolute Shrinkage and Selection Operator (LASSO),
3. Elastic Net
4. Least-Angle Regression (LARS)

## **Elastic Net**

When working with high-dimensional data sets with a large number of independent variables, correlations (relationships) among the variables can be often result in multicollinearity. These correlated variables which sometimes form groups or clusters called as an elastic net of correlated variables. To include the complete the model selection even if just one variable has been selected.

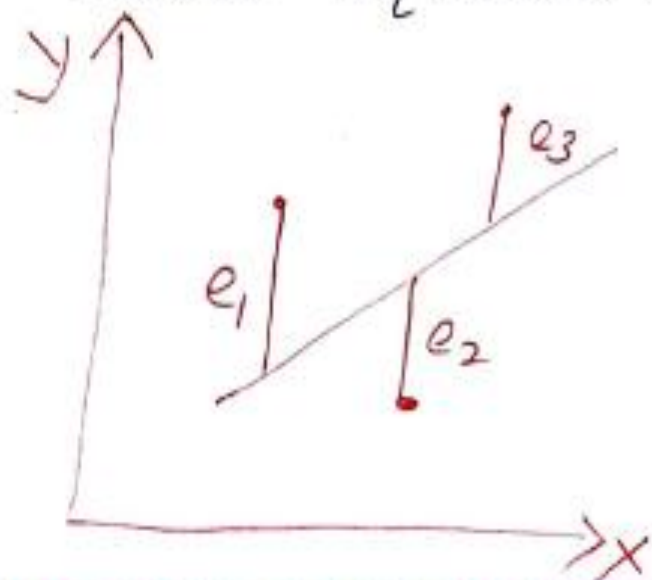


## Linear Regression

The Linear Regression model can be created by fitting a line among the scattered points. The line is of the form,

$$y = a_0 + a_1 x + e \quad \text{where } \begin{array}{l} a_0 - \text{intercept} \\ a_1 - \text{slope} \\ e - \text{error} \end{array}$$

Linear Regression is based on Ordinary Least Squares (OLS) method.



The vertical distance between each point (actual value) and the line (predicted value by the equation  $y = a_0 + a_1 x$ ) is called an error  $e$ .

The errors can be calculated as

1) These individual errors are added to compute the total error is called sum of residuals (based on absolute values).

2) The squares of the individual errors and added to give a sum of squared error, is called sum of squared error (SSE) or the lowest sum of squared error is called Ordinary least square (OLS).

The line with the lowest sum of squared error is called line of best fit.

In another words, OLS is an optimization technique where the difference b/w the data points and the line is optimized.

## Metrics used in Regression methods

### Standard error (or) Residual error:

The difference b/w the actual ( $y$ ) and predicted value ( $\bar{y}$ ).

Mean absolute error (MAE): It is the mean of residual error. It can be written as,

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \bar{y}_i|$$

Mean Squared error (MSE): It is the mean of ~~sum~~ square of residual error. It can be written as,

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2$$

Root Mean Square Error (RMSE):

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2}$$

Relative MSE: It is the ratio of the prediction ability ( $\bar{y}$ ) to the average of the trivial population. The value of  $RelMSE=0$ , the model is perfect. If  $RelMSE$  is b/w 0 and 1, the model is acceptable and  $RelMSE > 1$ , the model is not acceptable.

Coefficient of Variation (CV):

$$CV = \frac{RMSE}{\bar{y}} \quad \begin{array}{l} \text{(Root mean square error)} \\ \text{(average of the trivial population)} \end{array}$$



Problem! find MAE, MSE, RMSE, RelMSE and CV for predicting the sales of items. Also, suggest whether the model is acceptable or not.

Training item table

Items	Actual sales (in lakhs)
$I_1$	80
$I_2$	90
$I_3$	100
$I_4$	110
$I_5$	120

Test item Table

Test Items	Actual value $y_i$	Predicted value $\hat{y}_i$
$I_6$	80	75
$I_7$	75	85

$$\begin{aligned}\underline{\text{MAE}} &: \frac{1}{2} [ |80 - 75| + |75 - 85| ] \\ &= \frac{1}{2} [ 5 + 10 ] = 7.5\end{aligned}$$

$$\underline{\text{MSE}} : \frac{1}{2} \times [ (80 - 75)^2 + (75 - 85)^2 ] = \frac{125}{2} = 62.5$$

$$\underline{RMSE} = \sqrt{MSE} = \sqrt{62.5} = 7.91$$

for finding RelMSE and CV, the training table should be used to find average of  $y$ .

$$y = \frac{80 + 90 + 100 + 110 + 120}{5} = \frac{500}{5} = 100$$

$$\underline{RelMSE} = \frac{(80 - 75)^2 + (75 - 85)^2}{(80 - 100)^2 + (75 - 100)^2} = \frac{125}{1025} = 0.1219$$

It lies b/w 0 to 1. So, this model is acceptable.

$$\underline{CV} = \frac{RMSE}{\bar{y}} = \frac{7.91}{100} = 0.79 \text{ (or)} = 0.8$$

## RIDGE, LASSO and Elasticnet Regression

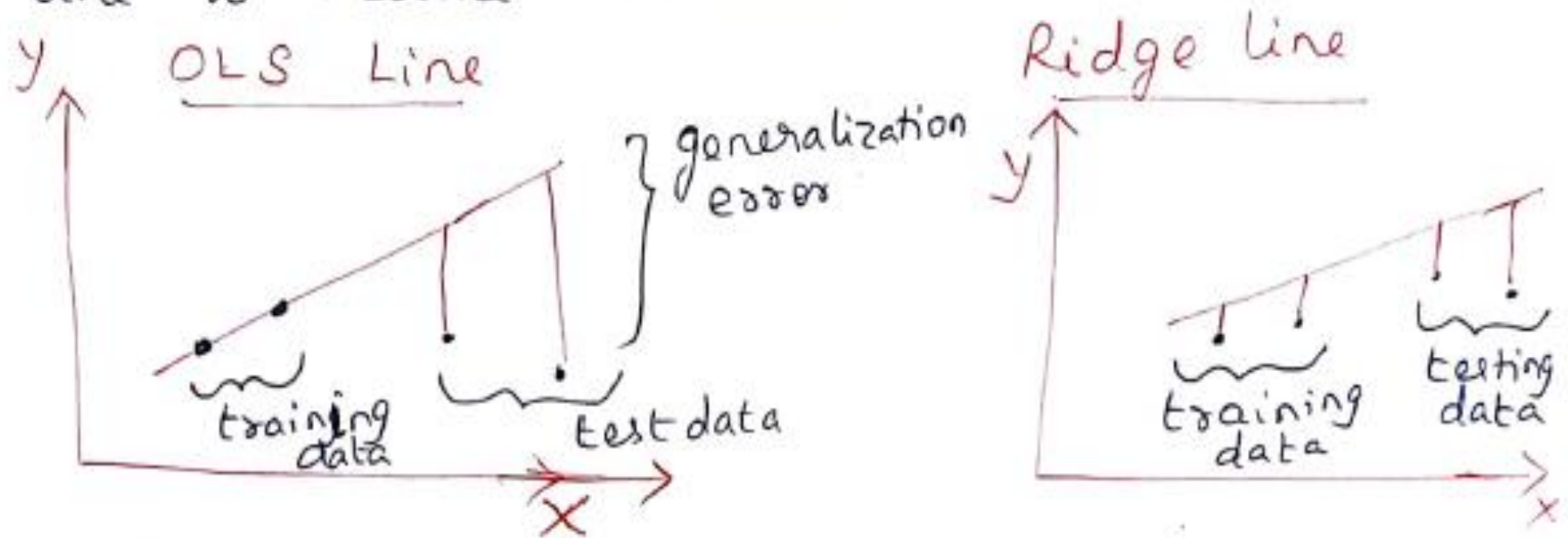
- \* Ordinary Least Square (OLS) fits a line for the datapoints that minimize the sum of squared errors b/w data points and the line of fit.
- \* There are two issues that need to be considered.

Bias - If the selected model does not fit both training and testing data, then this error is called bias. Also known as underfitting.

Variance - If the selected model works nicely for the training data but is not fit for testing data, then this error is called variance, <sup>(or)</sup> generalization error. Also known as overfitting.



\* The main idea to solve variance is to introduce a small bias to create a ridge line to reduce variance.



\* In order to get ridge line by changing the slope of the OLS line. This introduces a bias in the model and because of the bias, the line does not fit for training data, but the overall variance gets reduced.

\* So, the aim of regularization is to reduce the test data error.

\* The regularization methods that add a penalty to penalize the regression coefficients.

### Ridge Regularization:

The ridge regression is given as,

$$\text{Ridge regularization } J = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2 + \lambda * \text{slope}^2$$

where  $\sum_{i=1}^n (y_i - \bar{y}_i)^2$  is the sum of squared residuals

$\lambda * \text{slope}^2$  is the penalty term added to change the slope of the line of fit to reduce variance. But it is difficult to select  $\lambda$ .  
When  $\lambda = 0$  it is simply OLS technique.

## LASSO [Least Absolute Shrinkage and Selection Operator]

The LASSO regression is given as,

$$\text{LASSO regression } \} = \frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2 + \lambda * |\text{slope}|$$

Difference b/w Ridge and LASSO regression

### Ridge Regression

1. Penalty term is  $\lambda * \text{slope}^2$
2. This method shrinks the coefficients of less important variables close to zero.
3. Not suitable for feature selection.

### LASSO Regression

1. penalty term is  $\lambda * \text{slope}$
2. This method shrinks the coefficients of less important variables to zero.
3. Good feature selector by removing all irrelevant variables.



Elastic Net: It is hybrid method of combining both Ridge and LASSO regression methods. The Elastic Net is given as,

$$\text{Elastic Net Regression} \} = \underbrace{\frac{1}{n} \sum_{i=1}^n (y_i - \bar{y}_i)^2}_{\text{Sum of Squared residuals}} + \underbrace{\lambda_1 * |\text{slope}|}_{\text{LASSO}} + \underbrace{\lambda_2 * |\text{slope}|^2}_{\text{Ridge}}$$

When  $\lambda_1$  and  $\lambda_2$  are zero, then Elastic Net is simple OLS method. When  $\lambda_1 = 0$ , Elastic Net serves as ridge regression and when  $\lambda_2 = 0$ , Elastic Net serves as LASSO regression. When both  $\lambda_1$  and  $\lambda_2$  are greater than zero, then it serves as hybrid technique.

**Ridge regression:** It performs L2 regularization, i.e. it adds penalty equivalent to **square of the magnitude of coefficients**.

Minimization objective of ridge = LS Obj +  $\lambda \times$  (sum of square of coefficients) where

LS Obj = Least square objective loss function =  $\frac{1}{N} \sum_{i=1}^N (\hat{Y} - Y)^2$

$\lambda$  = any scalar value starts from 1, 2, 3 ....

(sum of square of coefficients) =

**Lasso regression:** It performs L1 regularization, i.e. it adds penalty equivalent to the **absolute value of the magnitude of coefficients**.

Minimization objective of ridge = LS Obj +  $\lambda \times$  (absolute value of the magnitude of coefficients)

where

LS Obj = Least square objective loss function =  $\frac{1}{N} \sum_{i=1}^N (\hat{Y} - Y)^2$

$\lambda$  = any scalar value starts from 1, 2, 3 ....

(sum of square of coefficients) =  $\sum_{i=1}^N |\theta_i|$