SCHOOL OF COMPUTER SCIENCE, ENGINEERING AND APLICATIONS
BHARATHIDASAN UNIVERSITY, TIRUCHIRAPPALLI - 620023

DEEP LEARNING

M.SC AI (II)

Dr. S. KALAIVANI

GUEST LECTURER

UNIT I

WHAT IS DEEP LEARNING?

Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans knowledge. It uses multi-layered structures of algorithms called neural networks to analyzing data.

WHAT IS ARTIFICIAL INTELLIGENCE?

Artificial Intelligence is a application which can do its own task without any human intervention.

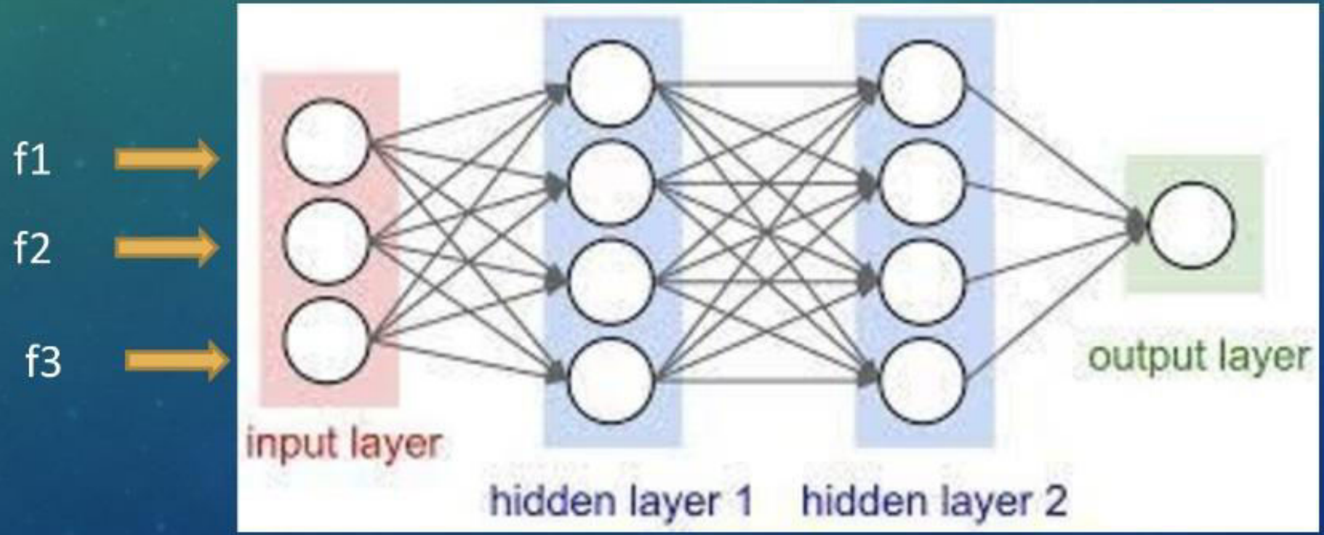
Ex: Netflix app, Self driving car, amazon app, sofia, chatbot

Machine learning is a subfield of artificial intelligence. It's a statistical tool, analysis data, visualize data, prediction, forecasting, clustering data.

WHAT IS MACHINE LEARNING?

HOW DOES DEEP LEARNING WORKS?

- Most deep learning methods use neural network architectures, which is why deep learning models are often referred to as **deep neural networks**.
- The term "deep" usually refers to the number of hidden layers in the neural network.
- Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction. One of the most popular types of deep neural networks is known as convolutional neural networks (CNN)



TYPES OF NEURAL NETWORKS

- CNN (Convolutional Neural Network)
- RNN (Recurrent Neural Network)
- ANN (Artificial Neural Network)

APPLICATION ON DEEP LEARNING

- Computer vision
- Natural language processing (NLP)
- Reinforcement learning

CHALLENGES IN DEEP LEARNING

1. Data availability: It requires large amounts of data to learn from.

2. Computational Resources: For training the deep learning model, it is computationally expensive because it requires specialized hardware like GPUs and TPUs.

3. Time-consuming: While working on sequential data depending on the computational resource it can take very large even in days or months.

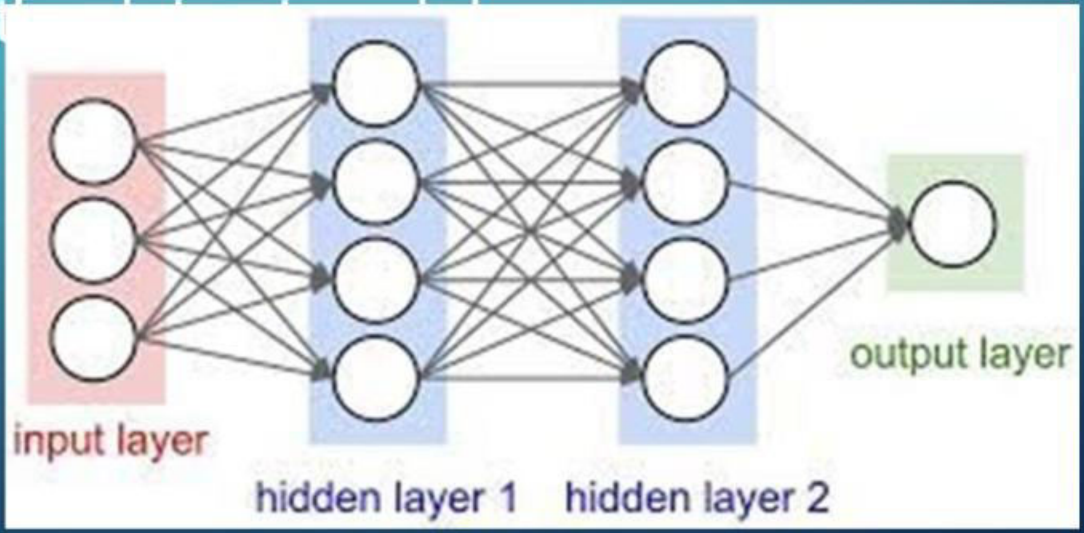
4. Interpretability: Deep learning models are complex, it works like a black box. It is very difficult to interpret the result.

5. Overfitting: when the model is trained again and again, it becomes too specialized for the training data, leading to overfitting and poor performance on new data.

NEURON NETWORK

- **Deep Learning** is a computer software that mimics the network of neurons in a brain.
- It is a subset of machine learning based on artificial neural networks with representation learning. This learning can be supervised, semi-supervised or unsupervised.
- **Deep learning algorithms are constructed with connected layers.**
- The first layer is called the Input Layer
- The last layer is called the Output Layer
- All layers in between are called Hidden Layers or Neurons.
- The word deep means the network join neurons in more than two layers.

- Each Hidden layer is composed of neurons. The neurons are connected to each other. The neuron will process and then propagate the input signal it receives the layer above it. The strength of the signal given the neuron in the next layer depends on the weight, bias and activation function.
- The network consumes large amounts of input data and operates them through multiple layers; the network can learn increasingly complex features of t



SOME OF THE DEEP LEARNING FRAMEWORKS OR TOOLS

- TensorFlow
- Keras
- PyTorch
- Caffe2
- CNTK
- MXNet
- Theano

NEED FOR BACKPROPAGATION

- Two methods are used to train a neural network: forward and backward. The network error is calculated at the end of the forward pass and should be as small as feasible.
-
- The network did not learn adequately from the data if the current error is significant. What exactly does this imply? It means that the existing weights are insufficiently accurate to reduce network error and produce accurate predictions. As a result, network weights should be updated to reduce network error.
- One of the deep learning algorithms responsible for changing network weights with the goal of lowering network error is the backpropagation algorithm. It's quite significant.

HOW TO TRAIN NEURAL NETWORK WITH BACKPROPAGATION?

- Backpropagation is a process involved in training a neural network. It involves taking the error rate (i.e. loss) of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights.
- Backpropagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration.) Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.

HOW DOES BACKPROPAGATION WORKS?

- As we all know, training in artificial neural networks happens in stages. These are:
-
- Initialization
- Propagation in the opposite direction
- Error Detection Function
- Backpropagation
- Update on weight
- Iteration

UNIT II

Bias-Variance Trade Off

- The **Bias-Variance Trade-Off** is a key concept in machine learning that deals with the balance between two types of errors that a model can make:
- bias error and variance error.
- Understanding this trade-off helps in building models that generalize well to new, unseen data.

Bias

- **Definition**: Bias is the error due to overly simplistic models that cannot capture the underlying patterns in the data.
- **High Bias**: This leads to underfitting, where the model is too simple to learn from the data effectively. An underfit model makes strong assumptions about the data and ignores the complexities.
- Example: A linear model trying to fit non-linear data.

Variance

- **Definition**: Variance is the error due to a model that is too complex and sensitive to the specific training data.
- **High Variance**: This leads to overfitting, where the model learns not only the underlying patterns but also the noise in the training data. As a result, the model performs well on the training set but poorly on new data.
- **Example:** A decision tree with many branches capturing noise instead of the actual signal.

The Trade-Off

- Increasing model complexity reduces bias (improves accuracy on training data),

  but increases variance (worse generalization on new data).
- Decreasing model complexity reduces variance but increases bias.
- The goal is to find a **sweet spot** where both bias and variance are balanced,

  minimizing the total error (i.e., test error).

Graphical Representation:

In many cases, the relationship between model complexity and error can be visualized as two curves:

- **Bias** decreases as model complexity increases.
- **Variance** increases as model complexity increases.
- The **total error** is the sum of bias and variance, and the minimum of this curve is the optimal model complexity.

- **Low Bias, High Variance**: Overfitting (too complex).
- **High Bias, Low Variance**: Underfitting (too simple).
- The **Bias-Variance Trade-Off** is about finding the right balance to reduce overall prediction error.

Overfitting:

The model performs very well on the training data (low training error) but poorly on the test or unseen data (high test error).

Causes:
- The model is too complex (e.g., too many features, deep trees, or high-degree polynomials).
- Insufficient training data.
- Training the model for too long or without regularization.

Solution:
- Use a simpler model.
- Reduce the number of features.
- Get more training data
  - .

- **Example**:

A decision tree that splits deeply on every minor variation in the data, creating a complex tree that perfectly fits the training data but performs poorly on new, unseen data.

2. Underfitting:

•**Definition**: Underfitting occurs when a model is too simple to capture the underlying patterns in the data. The model fails to learn the important relationships and performs poorly on both the training data and new data.

•**Symptoms**:

- The model has high error on both training data and test data.
- It cannot capture the complexities of the data, leading to poor predictions.

•**Causes**:

- The model is too simple (e.g., linear regression on non-linear data).
- Insufficient features or low model capacity.
- Training the model for too few epochs (in the case of neural networks).

•**Solution**:

•Use a more complex model.

•Add more relevant features to the model.

•Train the model for longer.

•Reduce the amount of regularization if too much has been applied.

•**Example**: Fitting a straight line (linear regression) to data that has a quadratic or non-linear relationship.

Overfitting and Underfitting are common problems in machine learning

that occur when a model does not generalize well to new data.

Penalty-based regularization is a method used to prevent overfitting by adding a penalty to the model’s loss function, discouraging overly complex models.

Types:

1.L1 Regularization (Lasso):

1. Adds the sum of the absolute values of the coefficients.
2. Encourages sparsity, driving some weights to zero (feature selection).

2.L2 Regularization (Ridge):

1. Adds the sum of the squared coefficients.
2. Shrinks coefficients but doesn’t set them to zero, reducing model complexity.

3.Elastic Net:

1. Combines L1 and L2 regularization.
2. Balances between feature selection (L1) and shrinkage (L2).

These techniques help models generalize better by preventing overfitting to training data.

Ensemble methods combine multiple machine learning models to improve overall performance and robustness. The idea is that multiple models working together can produce better predictions than a single model.

Key Types:

1.Bagging (Bootstrap Aggregating):

1. Trains multiple models on different subsets of the data (with replacement) and combines their predictions (e.g., Random Forest).
2. Reduces variance and helps prevent overfitting.

2.Boosting:

1. Sequentially trains models, each focusing on the mistakes of the previous one (e.g., AdaBoost, XGBoost).
2. Reduces bias and improves accuracy.

3.Stacking:

1. Combines predictions from multiple models by training a meta-model to make the final prediction.
2. Leverages the strengths of different models.

Ensemble methods improve accuracy, reduce overfitting, and generalize better to unseen data.

Early stopping is a regularization technique used to prevent overfitting in machine learning, particularly in neural networks. It monitors the model's performance on a validation set during training and stops the training process when the performance stops improving (e.g., when validation loss starts increasing), indicating that the model is beginning to overfit.

Key Points:

- **Prevents overfitting** by halting training before the model starts memorizing the training data.
- **Monitors validation performance** to determine the optimal stopping point.
- Reduces training time while ensuring better generalization on unseen data.

Unsupervised pre-training

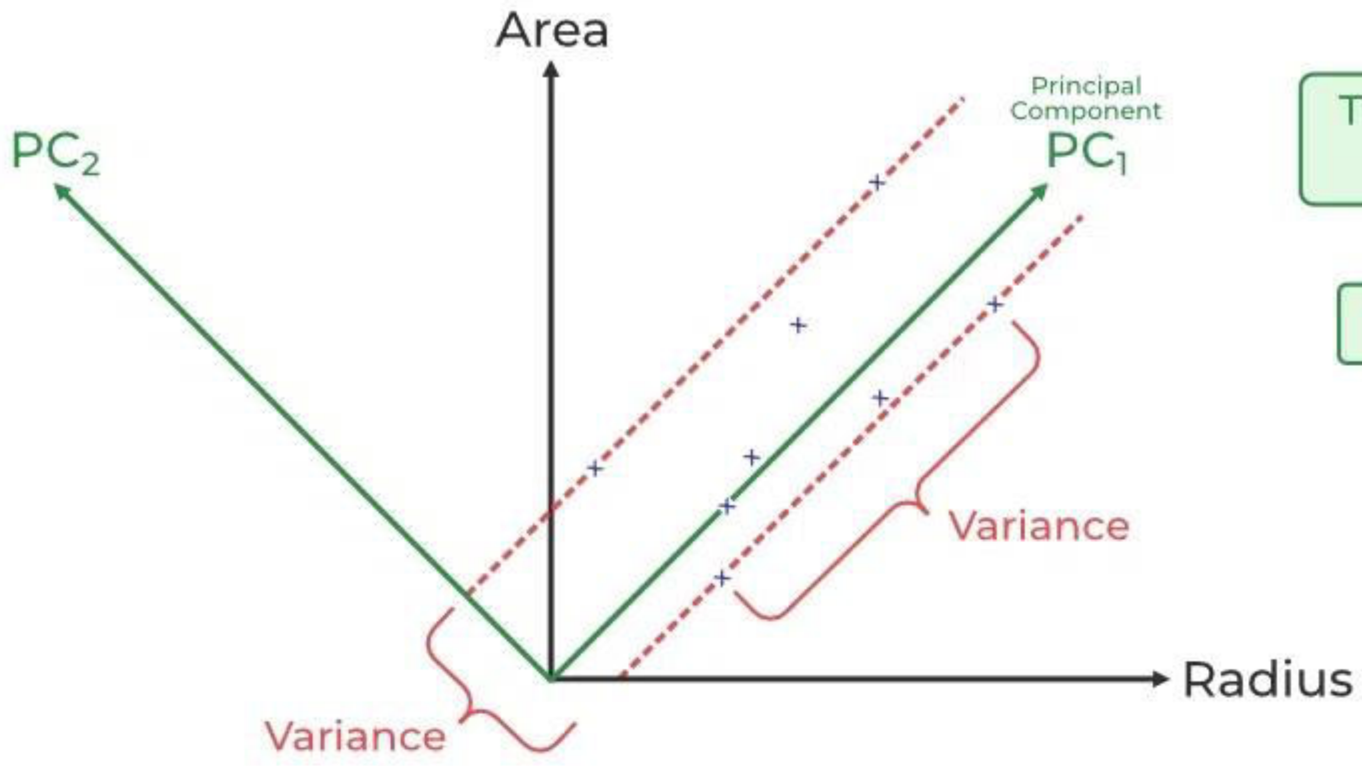
Unsupervised pre-training is a machine learning technique that uses unlabeled data to train a model that can later be refined with a <u>smaller amount of labeled data</u>. This technique is especially useful when labeled data is hard to find or expensive to get.

It's been used in a variety of domains, including natural language processing (NLP) and computer vision.

What Is Principal Component Analysis?

- Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.
- **Principal Component Analysis (PCA)** is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. PCA is the most widely used tool in exploratory data analysis and in machine learning for predictive models. Moreover,
- Principal Component Analysis (PCA) is an unsupervised learning algorithm technique used to examine the interrelations among a set of variables. It is also known as a general factor analysis where regression determines a line of best fit.

- The main goal of Principal Component Analysis (PCA) is to reduce the dimensionality of a dataset while preserving the most important patterns or relationships between the variables without any prior knowledge of the target variables.
- Principal Component Analysis (PCA) is used to reduce the dimensionality of a data set by finding a new set of variables, smaller than the original set of variables, retaining most of the sample's information, and useful for the regression and classification of data.



Transformation
2D → 1D

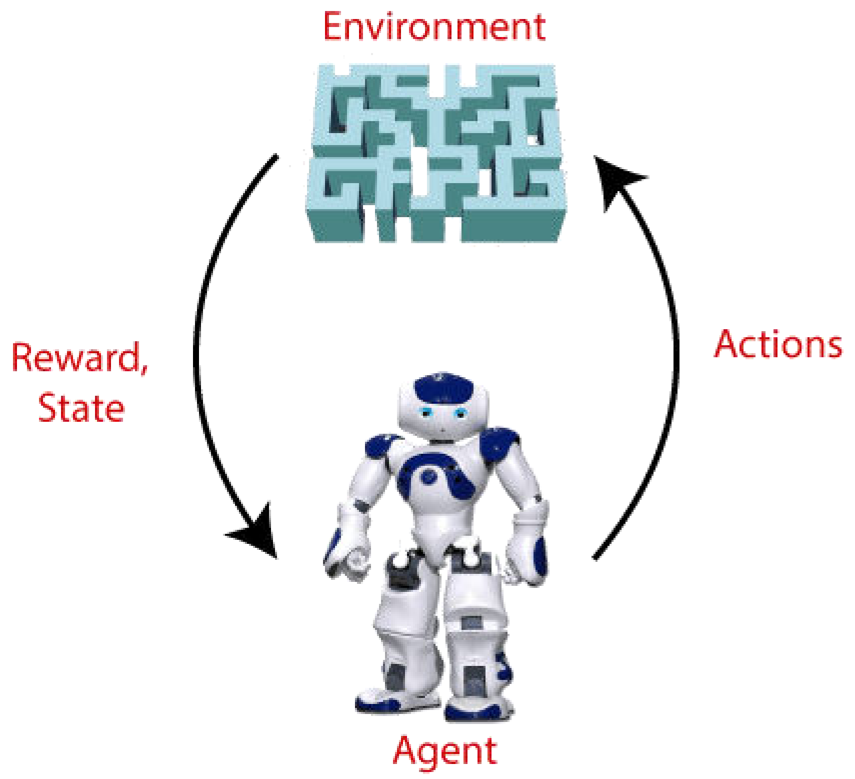
$PC_1 > PC_2$

- Principal Component Analysis (PCA) is a technique for dimensionality reduction that identifies a set of orthogonal axes, called principal components, that capture the maximum variance in the data. The principal components are linear combinations of the original variables in the dataset and are ordered in decreasing order of importance. The total variance captured by all the principal components is equal to the total variance in the original dataset.
- The first principal component captures the most variation in the data, but the second principal component captures the maximum variance that is orthogonal to the first principal component, and so on.
- Principal Component Analysis can be used for a variety of purposes, including data visualization, feature selection, and data compression. In data visualization, PCA can be used to plot high-dimensional data in two or three dimensions, making it easier to interpret. In feature selection, PCA can be used to identify the most important variables in a dataset. In data compression, PCA can be used to reduce the size of a dataset without losing important information.
- In Principal Component Analysis, it is assumed that the information is carried in the variance of the features, that is, the higher the variation in a feature, the more information that features carries.

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.

- ***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.***
- 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.

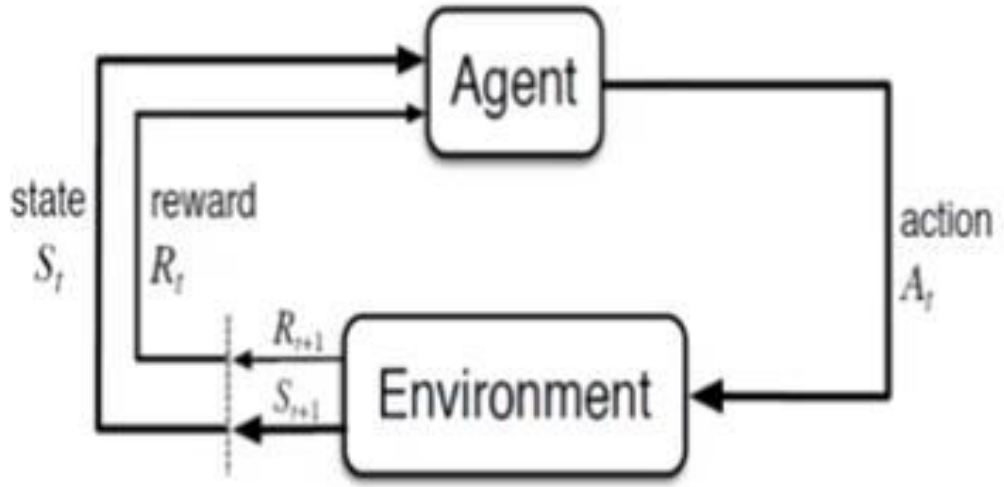


Markov decision process (MDP)

- The Markov decision process (MDP) is a mathematical tool used for decision-making problems where the outcomes are partially random and partially controllable. It’s a framework that can address most reinforcement learning (RL) problems.

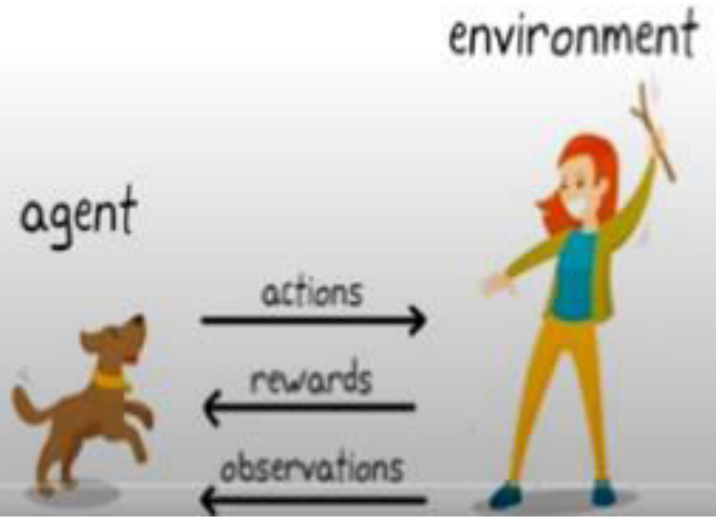
What is Markov Decision Process?

Markov Decision Process (MDP) is a mathematical framework to describe an environment in reinforcement learning.



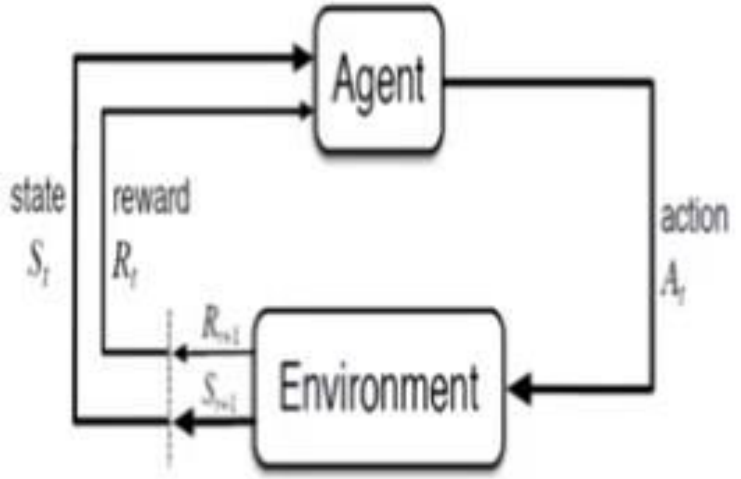
What is Reinforcement Learning?

Reinforcement Learning (RL) is a learning technique by which the learner learns to behave in an interactive environment using its own actions and rewards for its actions. The learner, often called, agent, discovers which actions give the maximum reward by exploiting and exploring them.



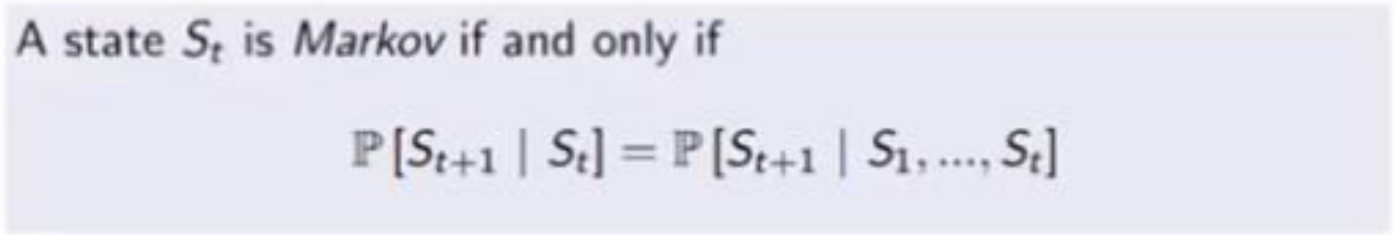
The Components of Reinforcement Learning

- **Environment**: The outside world with which the agent interacts
- **State**: Current situation of the agent
- **Action:** The choice that the agent makes at the current time step (For instance, Move left, Move right).
- **Reward**: Numerical feedback signal from the environment
- **Policy**: Method to map the agent's state to actions. A policy is used to select an action at a given state
- **Value**: Future reward (delayed reward) that an agent would receive by taking an action in a given state



Back to Markov Decision Process(MDP)...

- We need to understand MDP, we need to understand, Markov Property.



A state S_t is *Markov* if and only if

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

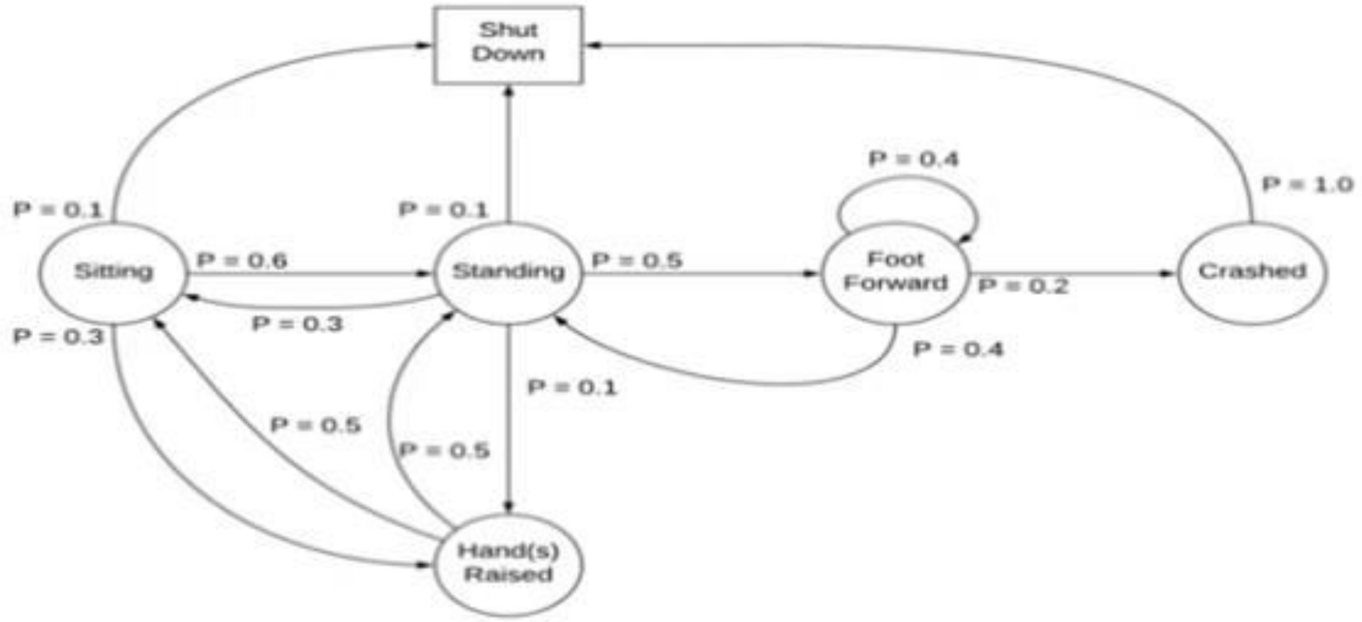
- Let us assume that a Robot was seated on a chair, it stood up and picked up the object.
- Current state= picking up object.
- Next state would depend only on the probability of this current state and not on the previous states.

Markov Process or Markov Chain

- The state transition probability is the probability of jumping to a state ***s'*** from the current state ***s***.

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

- A Markov Process is defined by *(S, P)* where *S* are the states, and *P* is the state-transition probability. It consists of a sequence of random states S_1, S_2, ... where all the states obey the Markov Property.



The Difference between the Markov process and the Markov Decision Process

Markov Decision Process: $\Pr(\mathbf{s'} \mid \mathbf{s}, \mathbf{a})$

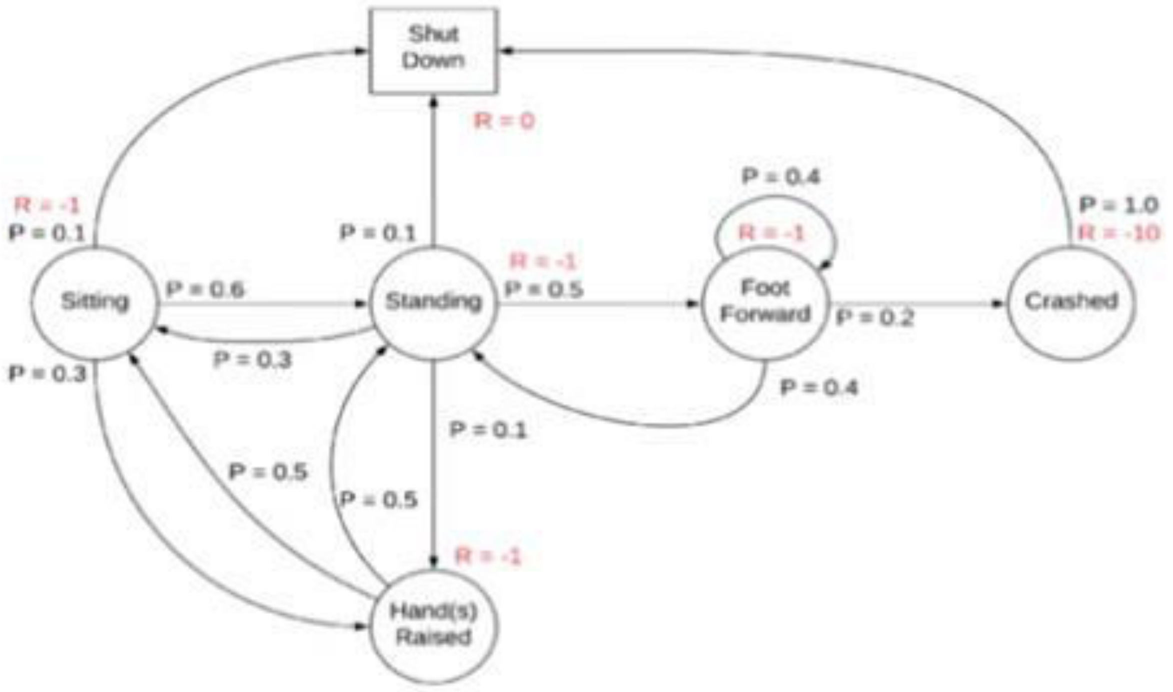
Markov Process $\Pr(\mathbf{s'} \mid \mathbf{s})$

Markov Reward Process

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

$$\mathcal{R}_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$$

- An MRP is defined by ***(S, P, R, γ)***, where ***S*** are the states, ***P*** is the state-transition probability, ***R*** is the reward, and ***γ*** is the discount factor.
  - The state reward ***R*** is the **expected reward** over all the possible states that one can transition to from state ***s***.
  - This reward is received for being at the state ***S***. **By convention**, it is said to be received after the agent leaves the state and hence, regarded as ***R(t+1)***.



Markov Decision Process

- An MDP is defined by ***(S, A, P, R, γ)***, where ***A*** is the set of actions. It is essentially **MRP with actions.**
- The rewards and the next state also depend on what action the agent picks.

$$\mathcal{P}^{a}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s, A_t = a\right]$$

$$\mathcal{R}^{a}_{s} = \mathbb{E}\left[R_{t+1} \mid S_t = s, A_t = a\right]$$

Policy (π)

- A policy defines the thought behind making a decision (picking an action). It defines the behavior of an RL agent.
- A policy is a **probability distribution** over the set of actions ***a***, given the current state ***s*** i.e., it gives the probability of picking an action ***a*** at state ***s***.

A *policy* π is a distribution over actions given states,

$$\pi(a|s) = \mathbb{P}\left[A_t = a \mid S_t = s\right]$$

Utility Function

- The reward function just captures the immediate or short-term consequences of executing actions.
- What we need instead is a function that captures the long-term consequences. Such a function is called a **utility function**.
- Intuitively, the utility of taking an action in some state is the expected immediate reward for that action plus the sum of the long-term rewards over the rest of the agent's lifetime, assuming it acts using the best policy.

$$U^{\pi}(s) = \mathrm{E}_{\Pr([s_0, s_1, \ldots] | s_0 = s, \pi)}\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

$$\pi^*(s) = \arg\max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' | s, a) U(s')$$

Bellman equation