UNIT-IV: Probabilistic Neural Networks

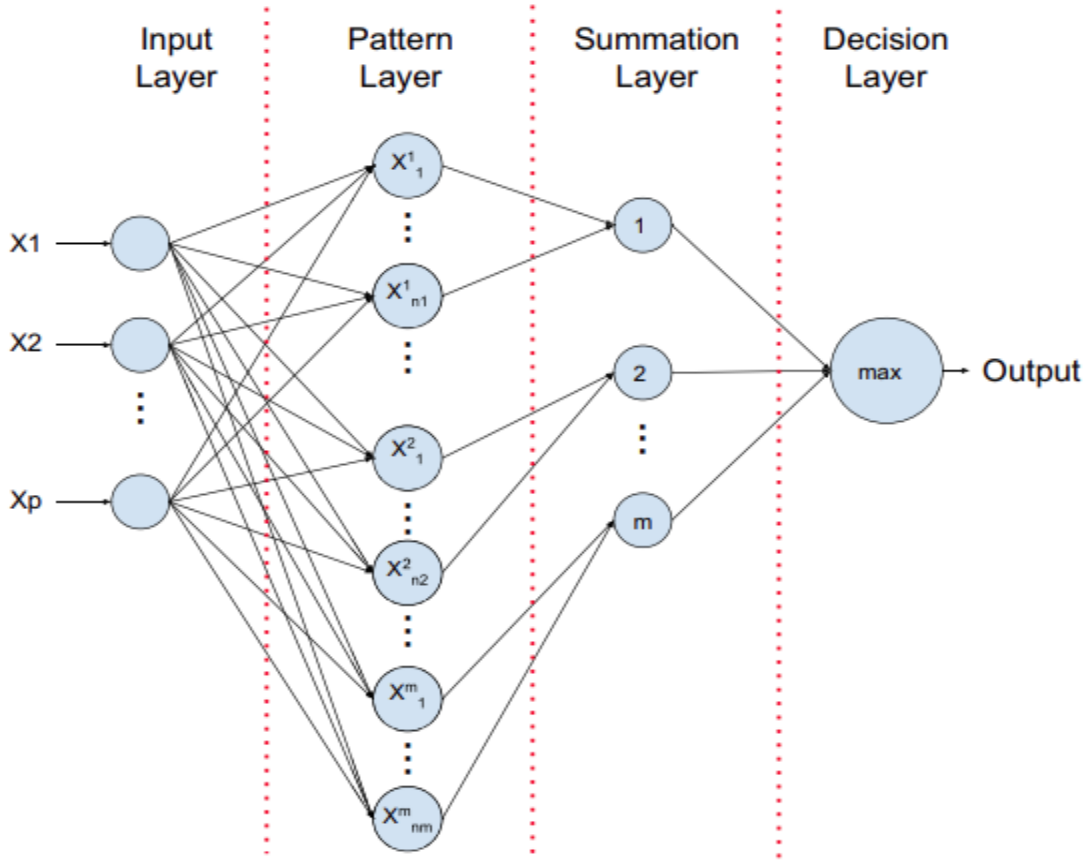
Syllabus: Hopfield Net, Boltzmann machine, RBMs, Sigmoid net, Auto encoders.

Probabilistic Neural Networks

- A probabilistic neural network (PNN) is a sort of feedforward neural network used to handle classification and pattern recognition problems.
- In the PNN technique, the parent probability distribution function (PDF) of each class is approximated using a **Parzen window** and a **non-parametric function.**
- The PDF of each class is then used to estimate the class probability of input data, **and Bayes' rule** is used to allocate the class with the highest posterior probability to new input data.

Parzen Window

- The Parzen-window method is a **non-parametric method** for estimating a probability density function p(x) for a specific point p(x) from a sample $p(x_n)$ that does not require any prior knowledge or assumptions about the underlying distribution.



Pattern Layer

- **A hidden neuron calculates the Euclidean distance between the test case and the neuron's center point, then uses the values to apply the kernel function.**

Summation Layer

- **Each category of the target variable has one pattern neuron in PNN.**
- **Each hidden neuron stores the actual target category of each training event; the weighted value output by a hidden neuron is only supplied to the pattern neuron that corresponds to the hidden neuron's category.**
- **The values for the class that the pattern neurons represent are added together.**

Decision Layer

- **The output layer compares the weighted votes accumulated in the pattern layer for each target category and utilizes the largest vote to predict the target category.**

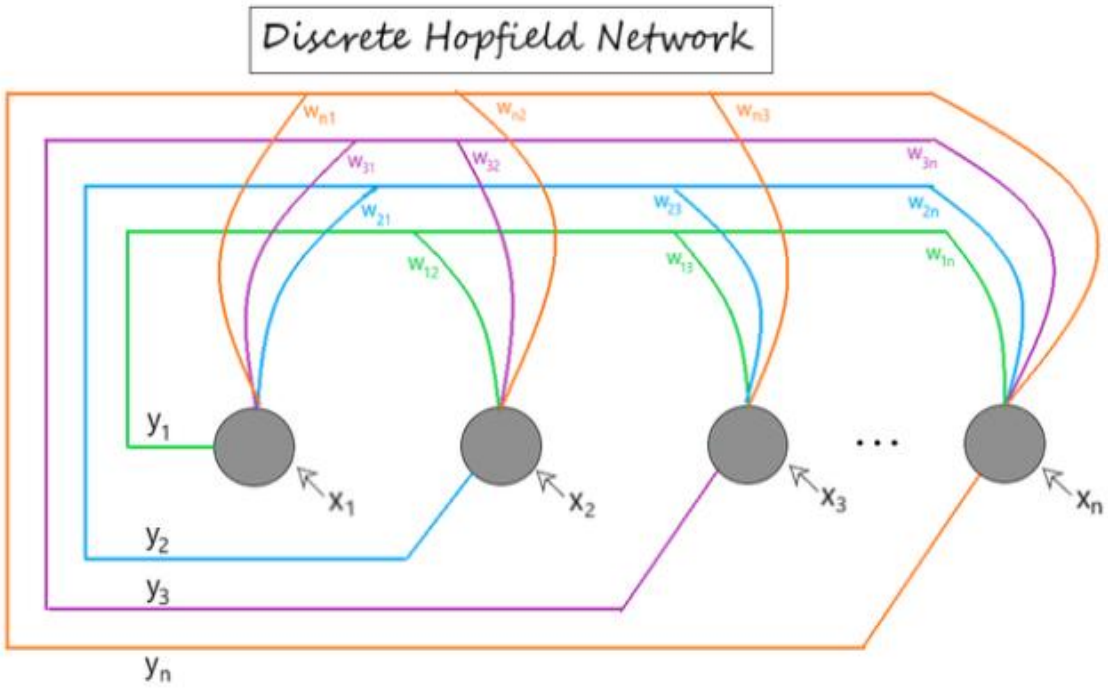
Hopfield Net

- **It consists of a single layer which contains one or more fully connected recurrent neurons.**
- **The Hopfield network is commonly used for optimization tasks.**
- **it generates a different response than normal neural nets.**
- **It is a fully interconnected neural network where each unit is connected to every other unit.**
- **It behaves in a discrete manner, i.e. it gives finite distinct output, generally of two types:**

  Binary (0/1) and Bipolar (-1/1)
- **The weights associated with this network are symmetric in nature and have the following properties.**

$$1.\ w_{ij} = w_{ji}$$
$$2.\ w_{ii} = 0$$

- **Structure & Architecture of Hopfield Network**
- **Each neuron has an inverting and a non-inverting output.**
- **Being fully connected, the output of each neuron is an input to all other neurons but not the self.**



- **Steps Involved in the training of a Hopfield Network are as mapped below:**

- **Initialize weights (w_{ij}) to store patterns (using training algorithm).**
- The algorithm iterates over each input vector yiy_iyi, initializing the network's activations based on each pattern to be memorized and adjusting the weights accordingly, enabling the network to store multiple patterns and retrieve them later from partial or noisy versions.
- Make the initial activators of the network equal to the external input vector x.

$$y_i = x_i : (for\ i = 1\ to\ n)$$

- **storing or testing that specific pattern in the network.**
- **Calculate the total input of the network y_{in} using the equation given below.**

$$y_{in_i} = x_i + \sum_j [y_j w_{ji}]$$

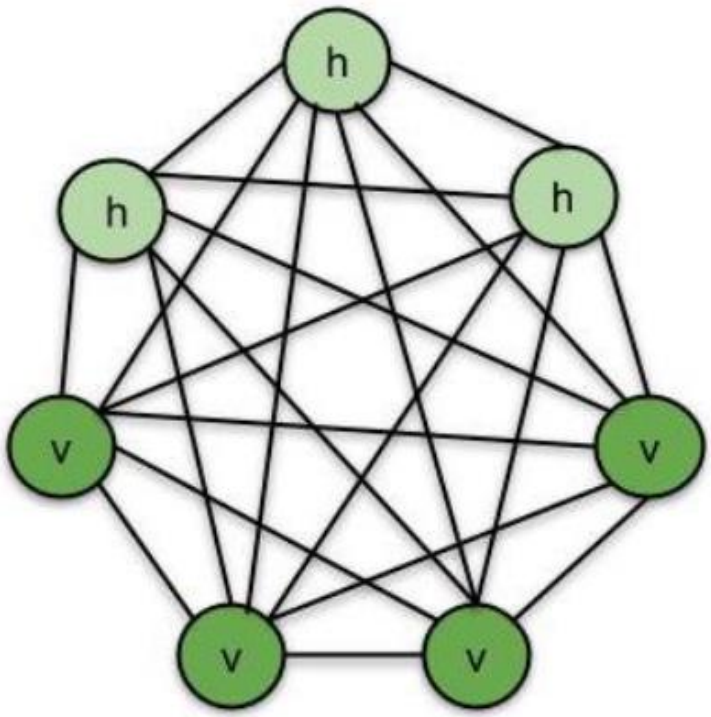
- **Apply activation over the total input to calculate the output as per the equation given below:**

- $$y_i = \begin{cases} 1 \text{ if } y_{in} > \theta_i \\ y_i \text{ if } y_{in} = \theta_i \\ 0 \text{ if } y_{in} < \theta_i \end{cases}$$
- **where θ_i (threshold) and is normally taken as 0)**
- **Now feedback the obtained output y_i to all other units. Thus, the activation vectors are updated.**
- **Test the network for convergence.**

Boltzmann Machines

- **Boltzmann Machines is an unsupervised DL model in which every node is connected to every other node.**
- **the Boltzmann Machines are undirected (or the connections are bidirectional).**
- **Boltzmann Machine is not a deterministic DL model but a stochastic or generative DL model.**
- ***Boltzmann* Machine has an input layer (also referred to as the *visible layer*) and one or several hidden layers (also referred to as the *hidden layer*).**

Boltzmann Machines



v - visible nodes, h - hidden nodes

- **They use recurrent structure.**
- **They consist of stochastic neurons, which have one of the two possible states, either 1 or 0.**
- **Some of the neurons in this are adaptive freestate and some are clamped frozenstate.**
  Training Algorithm
- **Boltzmann machines have fixed weights, hence there will be no training algorithm as we do not need to update the weights in the network.**
- **However, to test the network we have to set the weights as well as to find the consensus function CF.**
- **Boltzmann machine has a set of units U_i and U_j and has bi-directional connections on them.**
- **We are considering the fixed weight say w_{ij}.**
- **$w_{ij} \neq 0$ if U_i and U_j are connected.**
- **There also exists a symmetry in weighted interconnection, i.e. $w_{ij} = w_{ji}$.**
- **w_{ii} also exists, i.e. there would be the self-connection between units.**
- **For any unit U_i, its state u_i would be either 1 or 0.**
- **The main objective of Boltzmann Machine is to maximize the Consensus Function CF which can be given by**

$$CF = \sum_{i} \sum_{j \leqslant i} w_{ij} u_i u_j$$

- **Now, when the state changes from either 1 to 0 or from 0 to 1, then the change in consensus can be given by**

$$\Delta CF = (1 - 2u_i)(w_{ij} + \sum_{j \neq i} u_i w_{ij})$$

Here u_i is the current state of U_i.

- **Probability of the network to accept the change in the state of the unit is given by**

$$AF(i, T) = \frac{1}{1 + exp[-\frac{\Delta CF(i)}{T}]}$$

- **Here, T is the controlling parameter. It will decrease as CF reaches the maximum value.**

Testing Algorithm:

- **Step 1 – Initialize the following to start the training**
  - **Weights representing the constraint of the problem**
    Control Parameter T
- **Step 2 – Continue steps 3-8, when the stopping condition is not true.**
- **Step 3 – Perform steps 4-7.**
- **Step 4 – Assume that one of the state has changed the weight and choose the integer I, J as random values between 1 and n.**
- **Step 5 – Calculate the change in consensus as follows –**

$$\Delta CF = (1 - 2u_i)(w_{ij} + \sum_{j \neq i} u_i w_{ij})$$

- **Step 6 – Calculate the probability that this network would accept the change in state**

$$AF(i,T) = \frac{1}{1 + exp[-\frac{\Delta CF(i)}{T}]}$$

- **Step 7 – Accept or reject this change as follows –**

  Case I – if R < AF, accept the change.

  Case II – if R ≥ AF, reject the change.

  Here, R is the random number between 0 and 1.

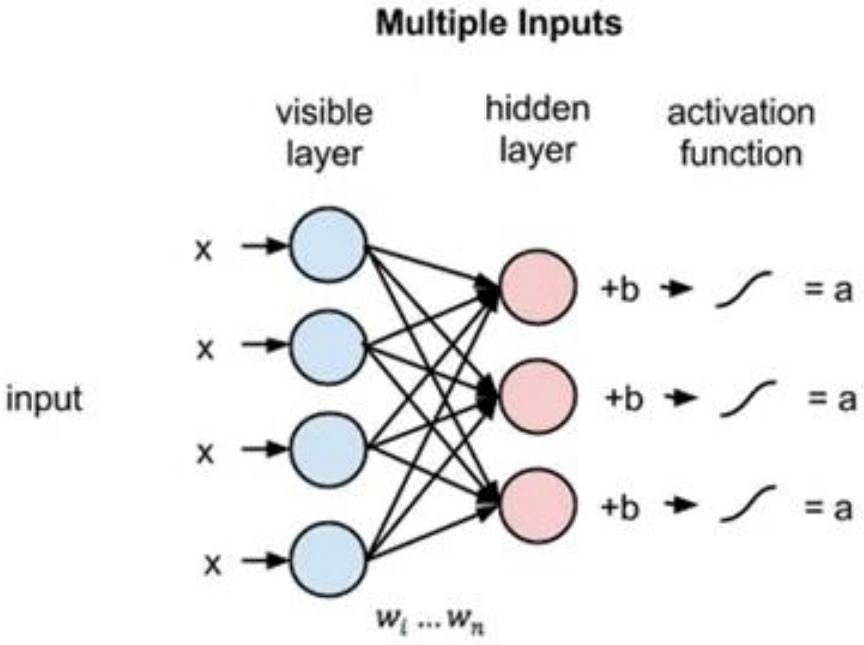
- **Step 8 – Reduce the control parameter temperature as follows –**
  - **Tnew = 0.95T_{old}**
- **Step 9 – Test for the stopping conditions which may be as follows –**

  Temperature reaches a specified value
- **There is no change in state for a specified number of iterations**

Restricted Boltzmann Machine:

- Restricted Boltzmann Machine (RBM) is a type of artificial neural network that is used for unsupervised learning.
- It is a type of generative model that is capable of learning a probability distribution over a set of input data.
- It is a type of neural network that consists of two layers of neurons – a **visible layer** and a **hidden layer.**
- The visible layer represents the input data, while the hidden layer represents a **set of features that are learned** by the network.
- The RBM is called "**restricted**" because each neuron in the visible layer is only connected to neurons in the hidden layer, and vice versa.
- This allows the RBM to learn a compressed representation of the input data by reducing the dimensionality of the input.
- RBM is a Stochastic Neural Network which means that each neuron will have some random behavior when activated.
- There are two other layers of bias units (hidden bias and visible bias) in an RBM.

- The RBM is trained using a process called **contrastive divergence,** which is a variant of the stochastic gradient descent algorithm.
- Contrastive divergence is an alternative training technique to approximate the graphical slope representing the relationship between a network's weights and its error, called the gradient.
- The algorithm most often used to train RBMs, that is, to optimize the weight matrix. , is the contrastive divergence (CD) algorithm
- During training, the network adjusts the weights of the connections between the neurons in order to maximize the likelihood of the training data.
- Once the RBM is trained, it can be used to generate new samples from the learned probability distribution.
- The inputs are multiplied by the weights and then added to the bias.
- The result is then passed through a sigmoid activation function and the output determines if the hidden state gets activated or not.

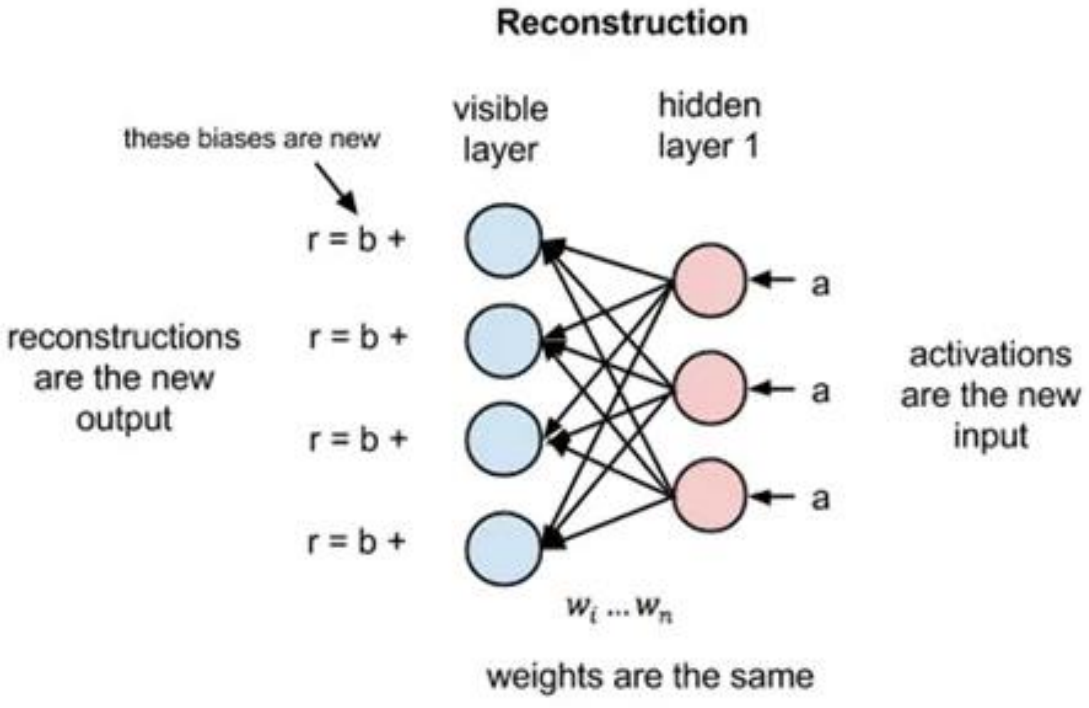


- a sigmoid function is
- So the equation that we get in this step would be

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{1 + e^{x}}$$

$$\mathbf{h}^{(1)} = S(\mathbf{v}^{(0)T} W + \mathbf{a})$$

- where **h(1)** and **v(0)** are the corresponding vectors (column matrices) for the hidden and the visible layers with the superscript as the iteration v(0) means the input that we provide to the network) and **a** is the hidden layer bias vector.



- The equation comes out to be:

$$\mathbf{v}^{(1)} = S(\mathbf{h}^{(1)} W^T + \mathbf{b})$$

- where **v(1)** and **h(1)** are the corresponding vectors (column matrices) for the visible and the hidden layers with the superscript as the iteration and **b** is the visible layer bias vector.

The learning process

- Now, the difference **v(0)–v(1)** can be considered as the reconstruction error that we need to reduce in subsequent steps of the training process.
- So the weights are adjusted in each iteration so as to minimize this error.
- In the forward pass, we are calculating the probability of output **h(1)** given the input **v(0)** and the weights **W** denoted by:

$$p(\mathbf{h}^{(1)} \mid \mathbf{v}^{(0)}; W)$$

- And in the backward pass, while reconstructing the input, we are calculating the probability of output **v(1)** given the input **h(1)** and the weights **W** denoted by:

$$p(\mathbf{v}^{(1)} \mid \mathbf{h}^{(1)}; W)$$

- The weights used in both the forward and the backward pass are the same. Together, these two conditional probabilities lead us to the joint distribution of inputs and the activations:

$$p(\mathbf{v}, \mathbf{h})$$

This is known as generative learning

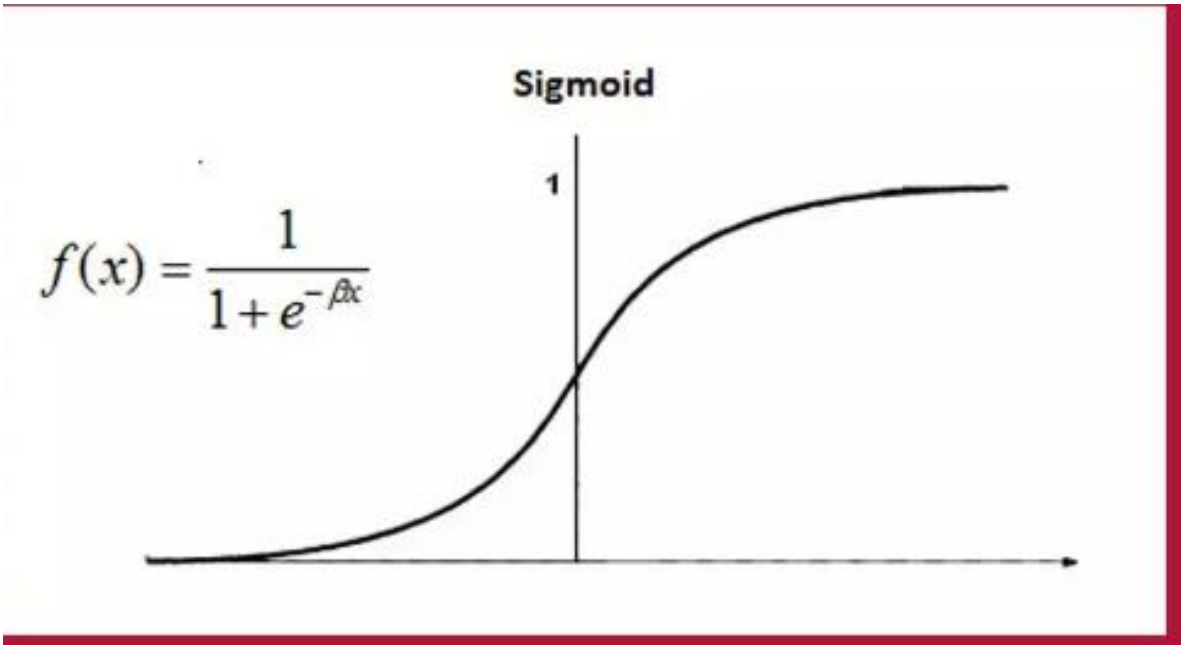
Sigmoid net:

- The sigmoid function is a mathematical function that has a characteristic that can take any real value and map it to between 0 to 1 shaped like the letter “S”.

- The sigmoid function is also known as a logistic function.

$$Y = 1 / 1+e^{-z}$$

Sigmoid graph:



- If the value of z goes up to positive infinity, then the predicted value of y will become 1.
- But if the value of z goes down to negative infinity, then the predicted value of y will become 0.
- If the outcome of the sigmoid function is greater than 0.5 then you would classify that label to be class 1 or positive class and if it is less than 0.5 then you would classify it to be a negative class or label it as class 0.
- The Sigmoid function performs the role of an activation function in machine learning which is used to add non-linearity in a machine learning model.
- Basically, the function determines which value to pass as output and what not to pass as output.

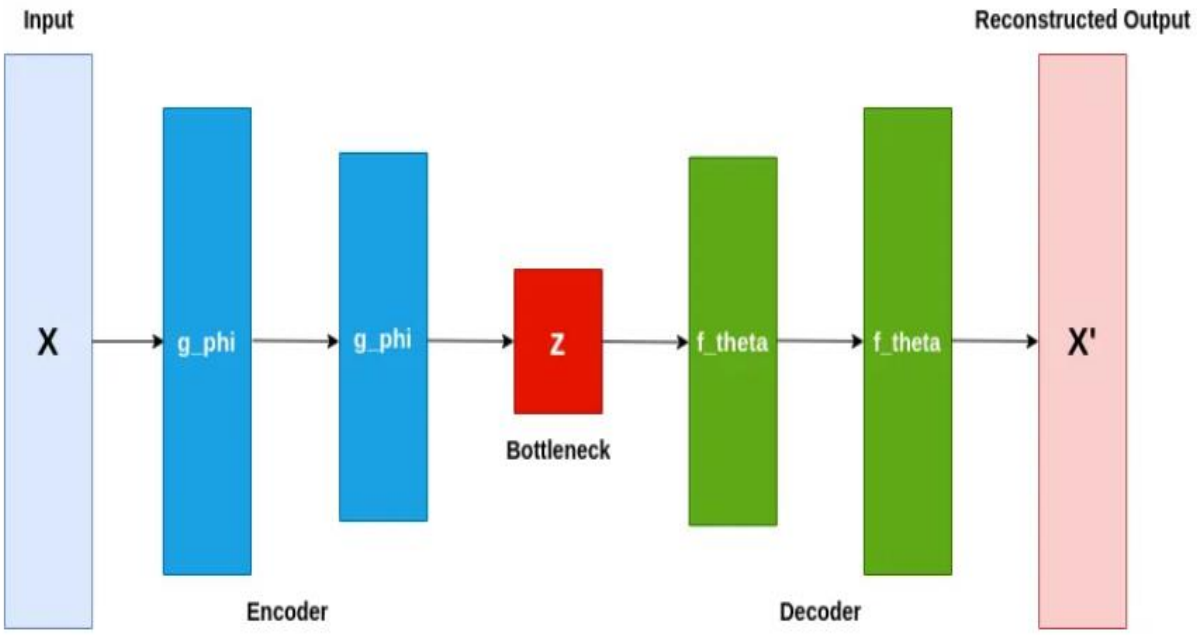
Auto encoders:

- Autoencoders are neural network-based models that are used for unsupervised learning purposes to discover underlying **correlations among data** and represent data in a smaller dimension.
- The autoencoders frame unsupervised learning problems as supervised learning problems to train a neural network model.
- The input only is passed a the output.

- The input is squeezed down a **lower encoded representation** using an **encoder network**, then a decoder network **decodes** the encoding to recreate back the input.
- The encoding produced by the encoder layer has a lower-dimensional representation of the data and shows several interesting complex relationships among data.

An Autoencoder has the following parts:

- **Encoder:** The encoder is the part of the network which takes in the input and produces a lower Dimensional encoding
- **Bottleneck:** It is the lower dimensional hidden layer where the **encoding is produced.**
- The bottleneck layer has a lower number of nodes and the number of nodes in the bottleneck layer also gives the dimension of the encoding of the input.
- **Decoder:** The decoder takes in the encoding and recreates back the input.



Auto encoders

- The bottleneck layer is the lower dimension layer.
- In the diagram, we have the neural networks encoder and decoder.
- **Phi** and **Theta** are the representing parameters of the encoder and decoder respectively.
- The target of this model is such that the Input is equivalent to the Reconstructed Output.
- To achieve this we minimize a loss function named **Reconstruction Loss.**

- Basically, Reconstruction Loss is given by the error between the input and the reconstructed output.
- It is usually given by the **Mean Square error** or **Binary Crossentropy** between the input and reconstructed output.
- **Autoencoders are used largely for anomaly detection:** As we know, autoencoders **create encodings** that basically captures the relationship among data.
- Now, if we train our autoencoder on a particular dataset, the encoder and decoder parameters will be trained to represent the relationships on the datasets.
- **Autoencoders are used for Noise Removal:** If we can pass the **noisy data as input** and **clean data** as output and train an autoencoder on such given data pairs, trained Autoencoders can be highly useful for noise removal.
- This is because noise points usually do not have any **correlations**.
- Now, as the autoencoders need to represent the data in the **lowest dimensions**, the encodings usually have only the **important relations.**
- Autoencoders as Generative models
- Autoencoders used for collaborative filtering.
- Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users.