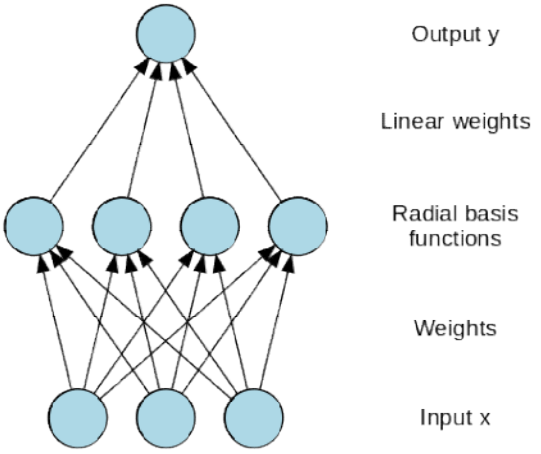
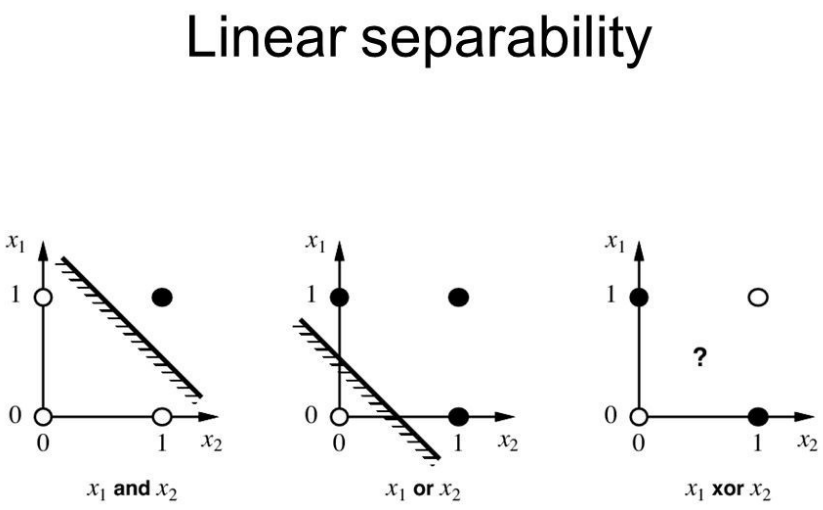
Radial Basis Functions Neural Networks



- In **Single Perceptron** / **Multi-layer Perceptron(MLP)**, we only have linear separability because they are composed of input and output layers(some hidden layers in MLP)

- For example, AND, OR functions are **linearly**-separable & XOR function is **not** linearly separable.

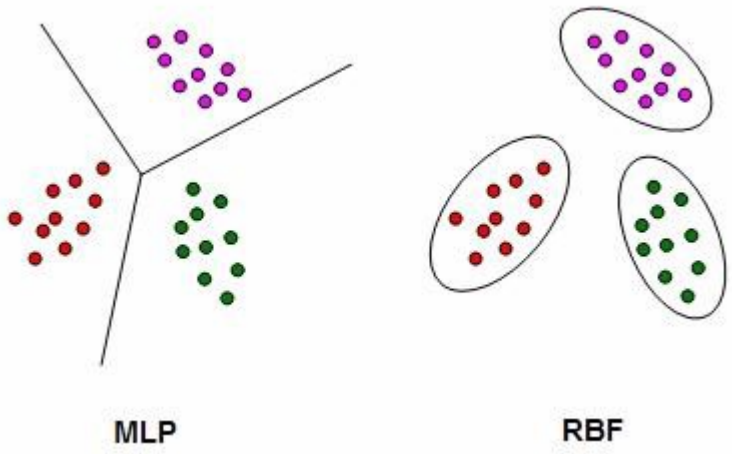


Linear-separability of AND, OR, XOR functions

- We atleast need **one hidden layer** to derive a non-linearity **separation**.

- Our RBNN what it does is, it transforms the input signal into another form, which can be then **feed** into the network to **get linear separability.**

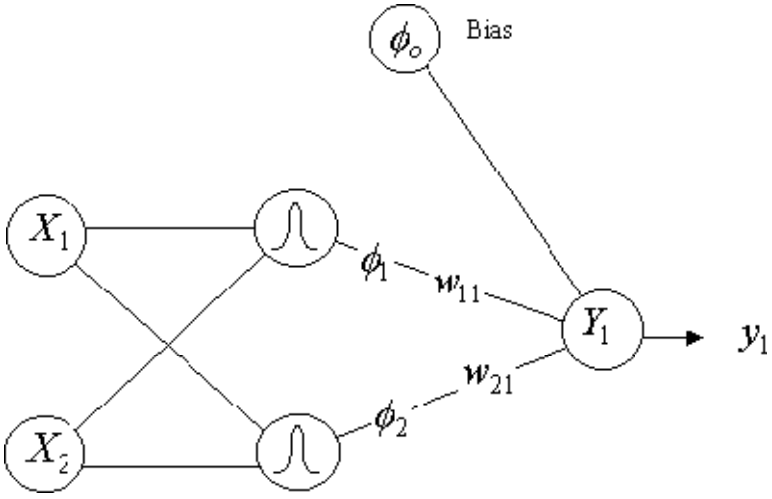
- RBNN is **structurally same** as perceptron(MLP).



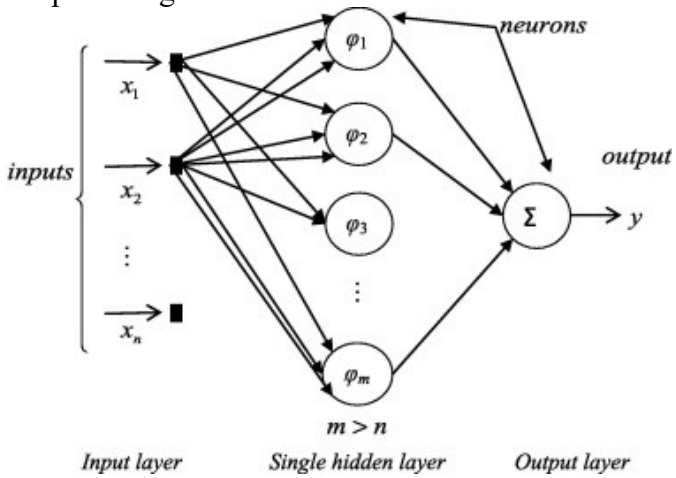
Distinction between MLP and RBF

- RBNN is composed of **input, hidden,** and **output** layer. RBNN is **strictly limited** to have exactly **one hidden layer**. We call this hidden layer as **feature vector.**

- RBNN **increases dimenion** of feature vector.



Simplest diagram shows the architecture of RBNN



Extended diagram shows the architecture of RBNN with hidden functions.

- We apply **non-linear transfer function** to the feature vector before we go for **classification problem.**

- When we increase the dimension of the feature vector, the linear separability of feature vector increases.

A non-linearity separable problem(pattern classification problem) is highly separable in high dimensional space than it is in low dimensional space.

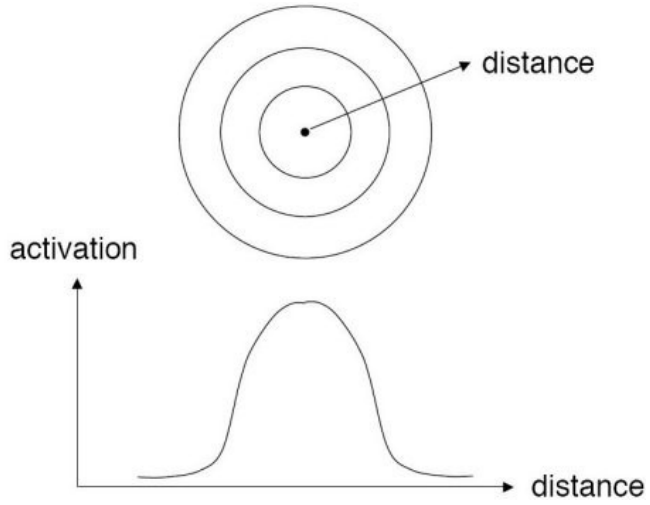
[***Cover's Theorem***]

- What is a Radial Basis Function ?

- we define a receptor = t

- we draw confrontal maps around the receptor.

- Gaussian Functions are generally used for Radian Basis Function(confrontal mapping). So we define the radial distance r = ||x- t||.

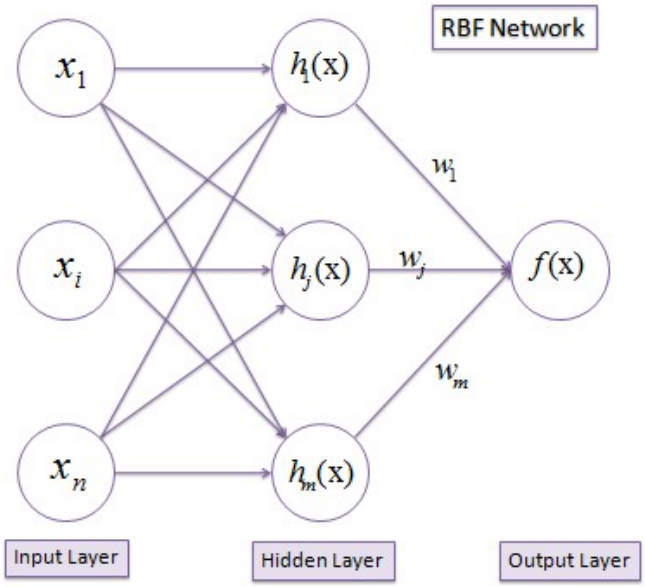


Radial distance and Radial Basis function with confrontal map

Gaussian Radial Function :=

$$\phi(r) = \exp(-r^2/2\sigma^2)$$

where $\sigma > 0$



$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

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

Classification only happens on the second phase, where linear combination of hidden functions are driven to output layer.

- Example. XOR function :-

- I have 4 inputs and I will not increase dimension at the feature vector here. So I will select 2 receptors here. For each transformation function $\phi(x)$, we will have each receptors t.

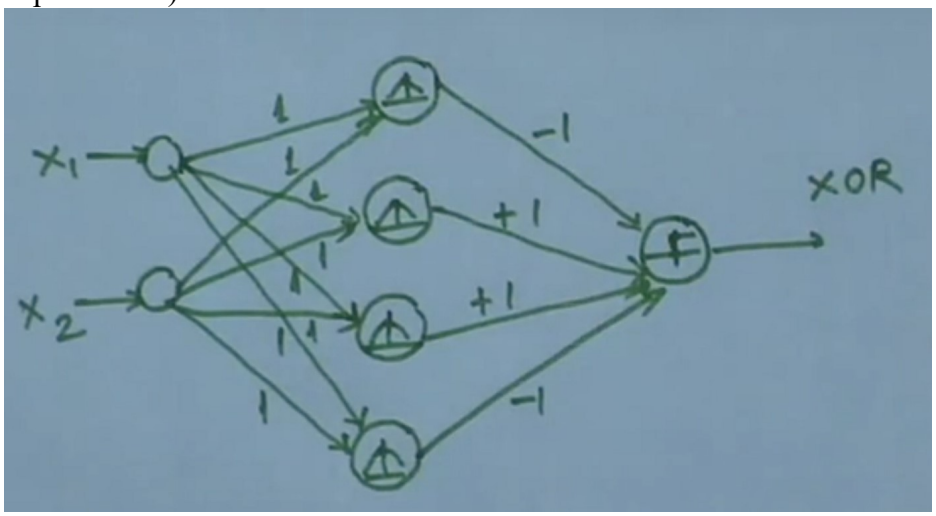
- Now consider the RBNN architecture,

- P := # of input features/ values.

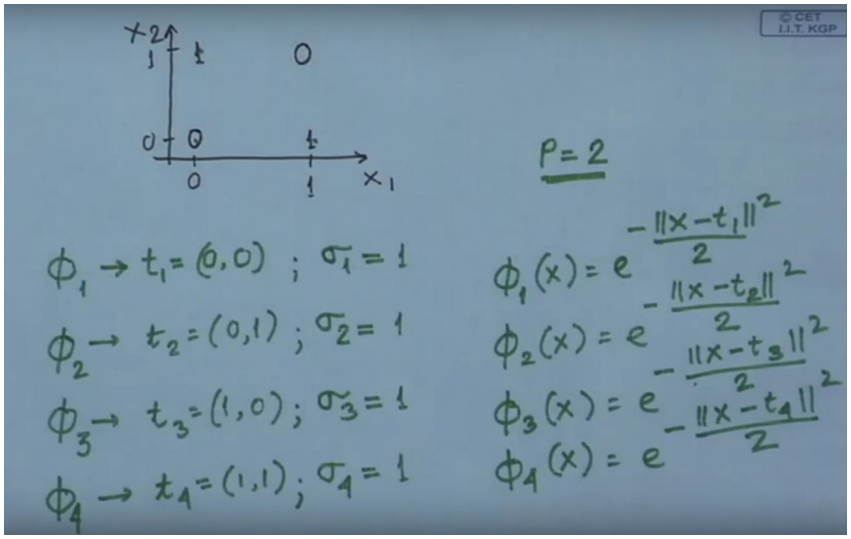
- M = # of transformed vector dimensions (hidden layer width). So M ≥ P usually be.

- Each node in the hidden layer, performs a set of non-linear radian basis function.

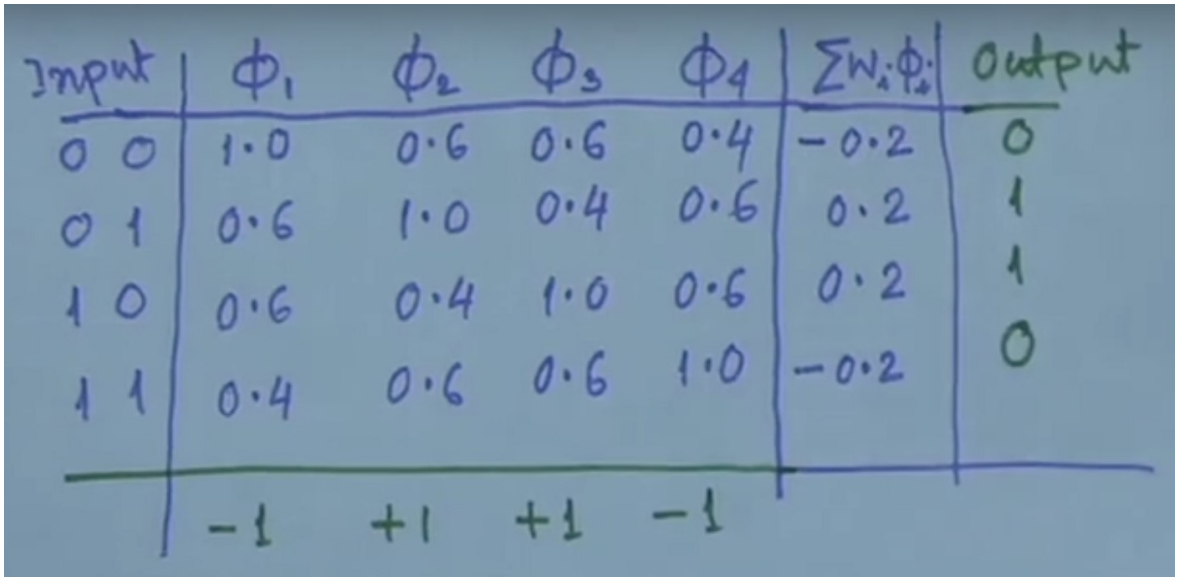
- Output C will remains the same as for the classification problems(certain number of class labels as predefined).



Architecture of XOR RBNN



Transformation function with receptors and variances.



Input	ϕ_1	ϕ_2	ϕ_3	ϕ_4	$\sum w_i \phi_i$	Output
0 0	1.0	0.6	0.6	0.4	−0.2	0
0 1	0.6	1.0	0.4	0.6	0.2	1
1 0	0.6	0.4	1.0	0.6	0.2	1
1 1	0.4	0.6	0.6	1.0	−0.2	0
	−1	+1	+1	−1		

Output → linear combination of transformation function is tabulated.

- Only Nodes in the hidden layer perform the radian basis transformation function.

- Output layer performs the linear combination of the outputs of the hidden layer to give a final probabilistic value at the output layer.

- So the classification is only done only @ ***(hidden layer → output layer)***

Training the RBNN :-

- First, we should train the **hidden layer** using **back propagation.**

- Neural Network training(back propagation) is a **curve fitting method**. It **fits** a **non-linear curve** during the **training** phase. It runs through stochastic approximation, which we call the back propagation.

- For each of the node in the hidden layer, we have to find **t**(receptors) & the variance (**σ**)[variance — the spread of the radial basis function]

- On the **second** training phase, we have to **update** the **weighting vectors** between **hidden layers & output layers.**

- In hidden layers, **each** node represents **each** transformation basis function. **Any** of the function could satisfy the non-linear separability OR even **combination** of set of functions could satisfy the non-linear separability.

- So in our hidden layer transformation, all the non-linearity terms are included. Say like $X^2 + Y^2 + 5XY$; its all included in a hyper-surface equation(X & Y are inputs).

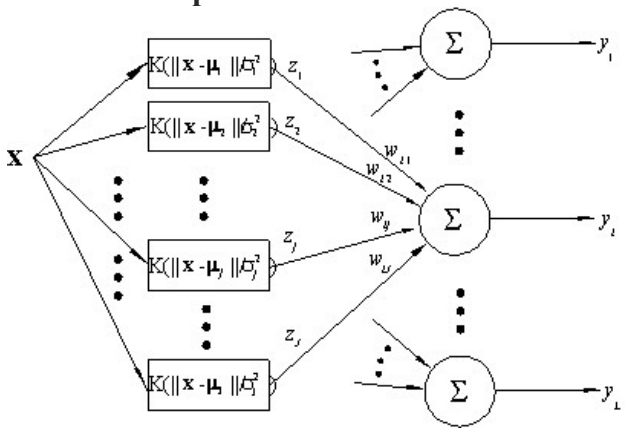
- Therefore, the first stage of training is done by **clustering algorithm.** We define the **number of cluster centers** we need. And by clustering algorithm, we compute the cluster centers, which then is assigned as the **receptors** for each hidden neurons.

- I have to cluster N samples or observations into M clusters (N > M).

- So the output "clusters" are the "receptors".

- for each receptors, I can find the variance as "**the squared sum of the distances between the respective receptor & the each cluster nearest samples**" := $1/N * ||X — t||^2$

- The interpretation of the first training phase is that the **"feature vector is projected onto the transformed space".**



Complex diagram depicting the RBNN

Advantages of using RBNN than the MLP :-

*1. Training in RBNN is **faster** than in Multi-layer Perceptron (MLP) → takes **many interactions** in MLP.*

*2. We can easily interpret what is the meaning / **function of the each node** in hidden layer of the RBNN. This is **difficult** in MLP.*

*3. (what should be the # of nodes in hidden layer **&** the # of hidden layers)*
*this **parameterization** is difficult in MLP. But this is not found in RBNN.*

*4. **Classification** will take more time in RBNN than MLP.*