3.1 Radial Basis Function Networks

Non Linear Problem

- **MLPs are highly non-linear in the parameter space ⇒ gradient descent ⇒ local minima**
- **RBF networks solve this problem by dividing the learning into two independent processes.**

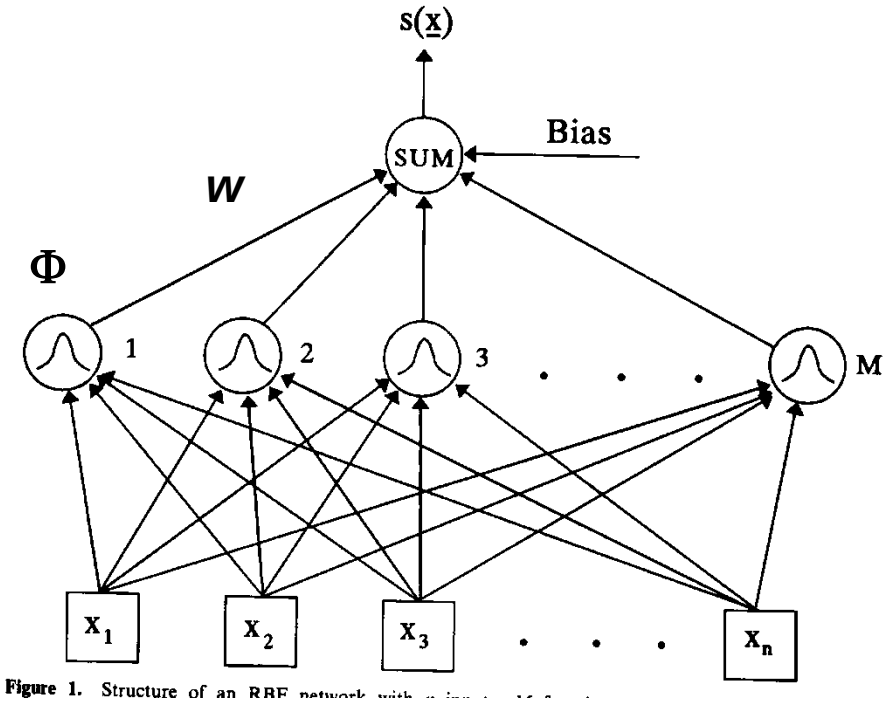
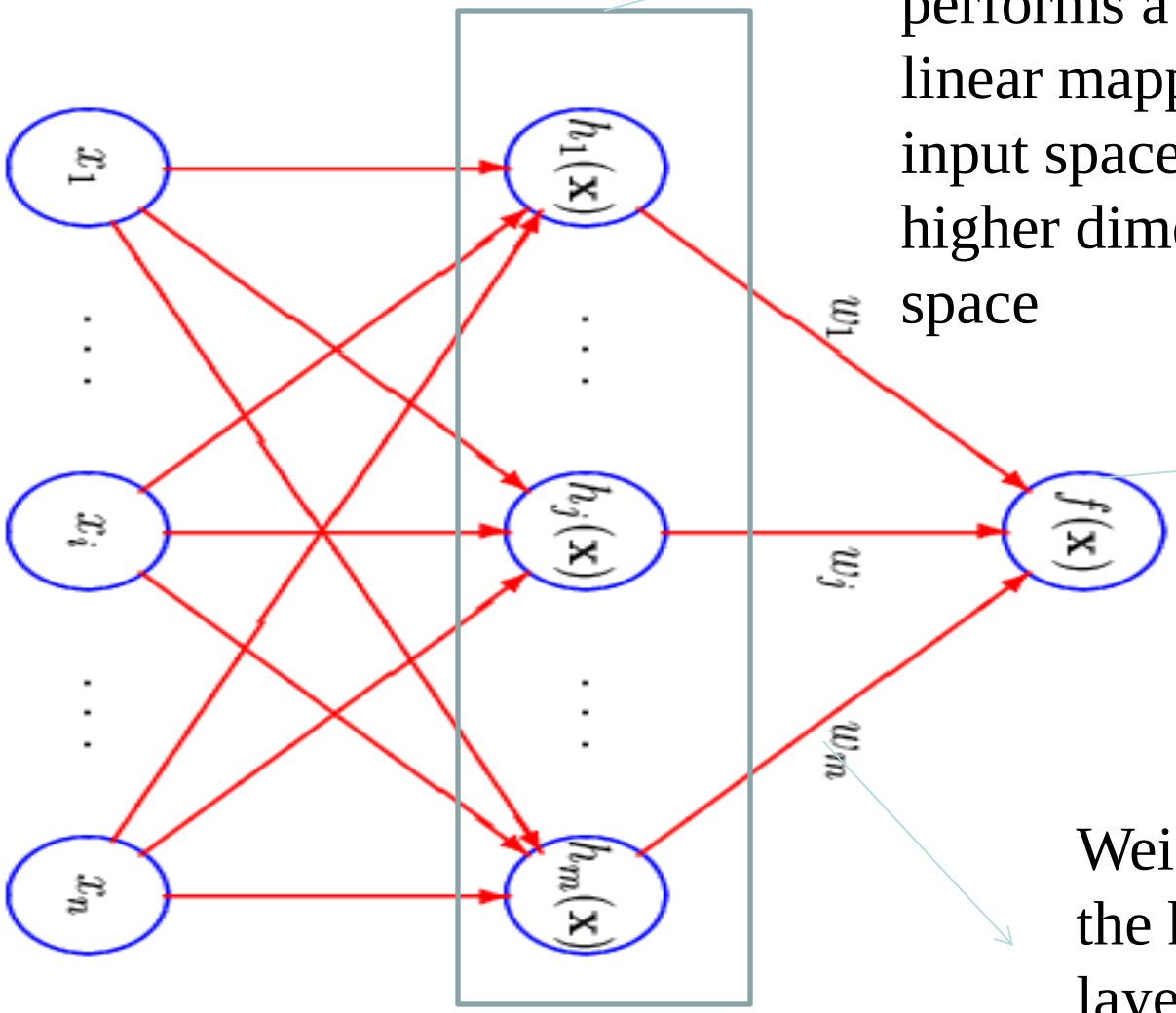


Figure 1. Structure of an RBF network with

Basic Architecture



Hidden layer performs a non-linear mapping from input space into higher dimensional space

Gaussian function

Weights from the hidden layer are cluster centers

Covers Theorem

"A complex pattern-classification problem cast in high-dimensional space nonlinearly is more likely to be linearly separable than in a low dimensional space"

(Cover, 1965).

Covers Theorem

- Let X denote a set of N patterns (points) $x_1, x_2, x_3, \ldots, x_N$
- Each point is assigned to one of two classes: X^+ and X^-
- This dichotomy is separable if there exist a surface that separates these two classes of points.

Covers Theorem

- For each pattern $x \in X$ define the next vector:

$$\varphi(x) = \left[\varphi_1(x), \varphi_2(x), \ldots, \varphi_M(x)\right]^T$$

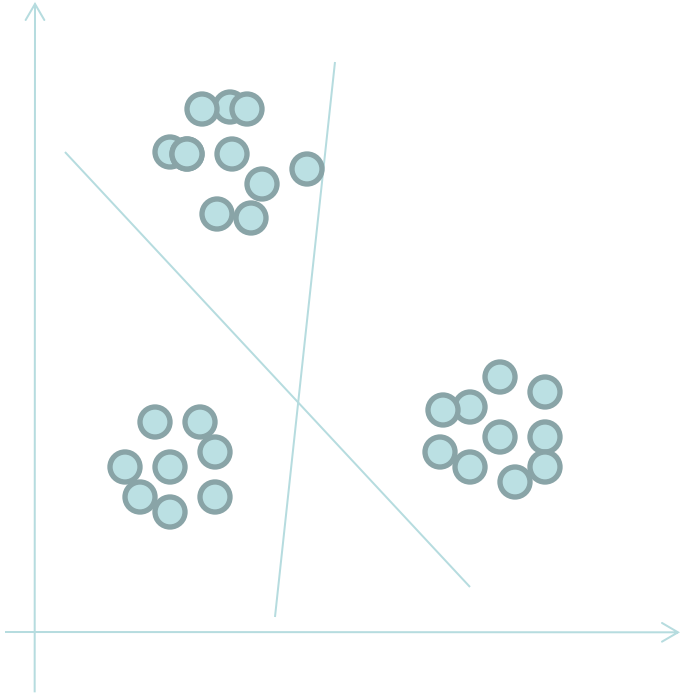
- The vector $\varphi(x)$ maps points in a p-dimensional input space into corresponding points in a new space of dimension m.
- Each $\varphi_i(x)$ is a hidden function, i.e., a hidden unit

Covers Theorem

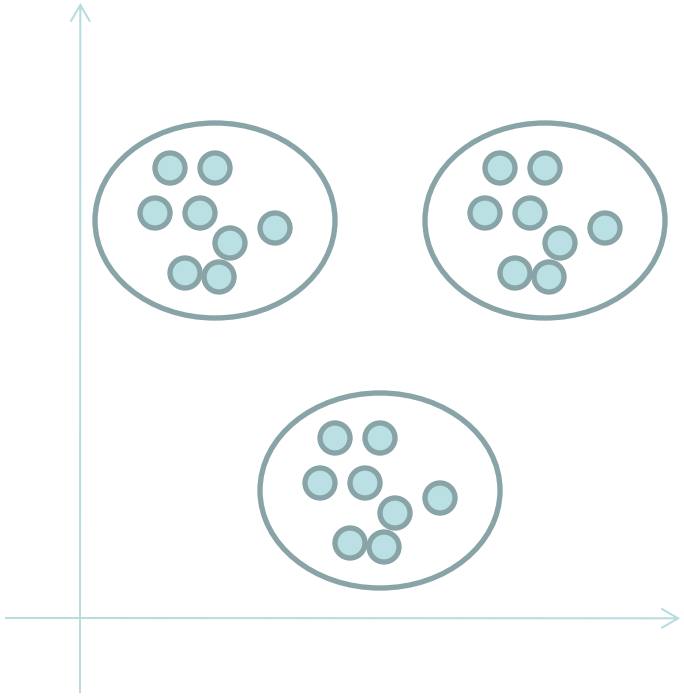
- A dichotomy $\{X^+, X^-\}$ is said to be φ-separable if there exist a m-dimensional vector w such that we may write
- (Cover, 1965):
  - $w^T \varphi(x) \geq 0, x \in X^+$
  - $w^T \varphi(x) < 0, x \in X^-$
  - The hyperplane defined by $w^T \varphi(x) = 0$, is the separating surface between the two classes.

RBF Networks for classification

- MLP



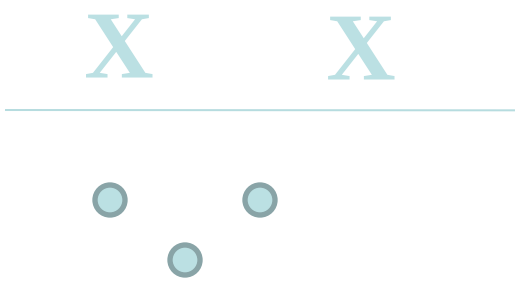
- RBF



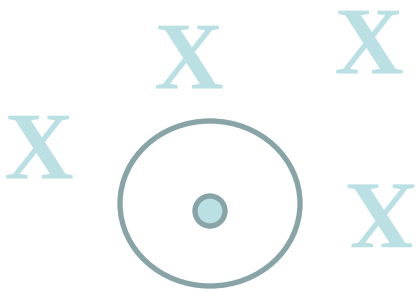
RBF Networks for classification Contd…

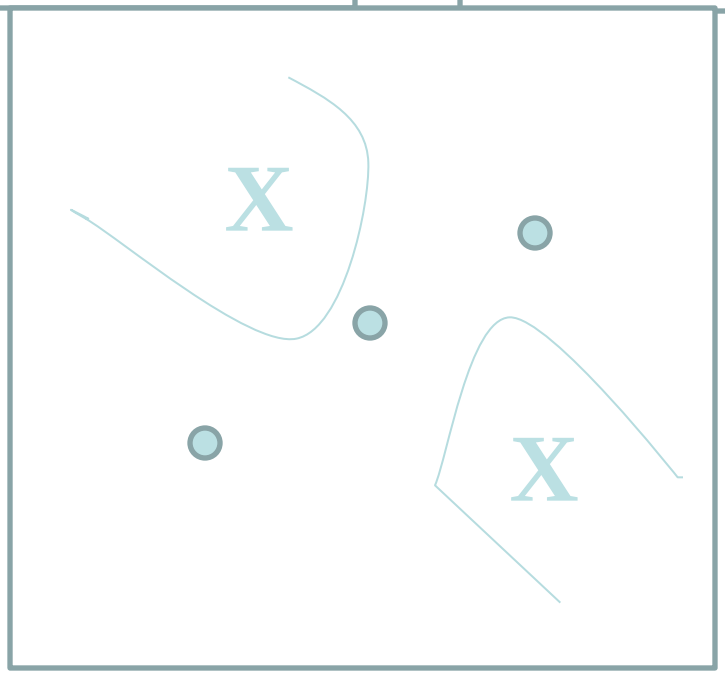
- An MLP naturally separates the classes with hyperplanes in the Input space
- RBF would be to separate class distributions by localizing radial basis functions
- Types of separating surfaces are
  - Hyperplane-linearly separable
  - Spherically separable-Hypersphere
  - Quadratically separable-Quadrics

Hyperplane-linearly separable



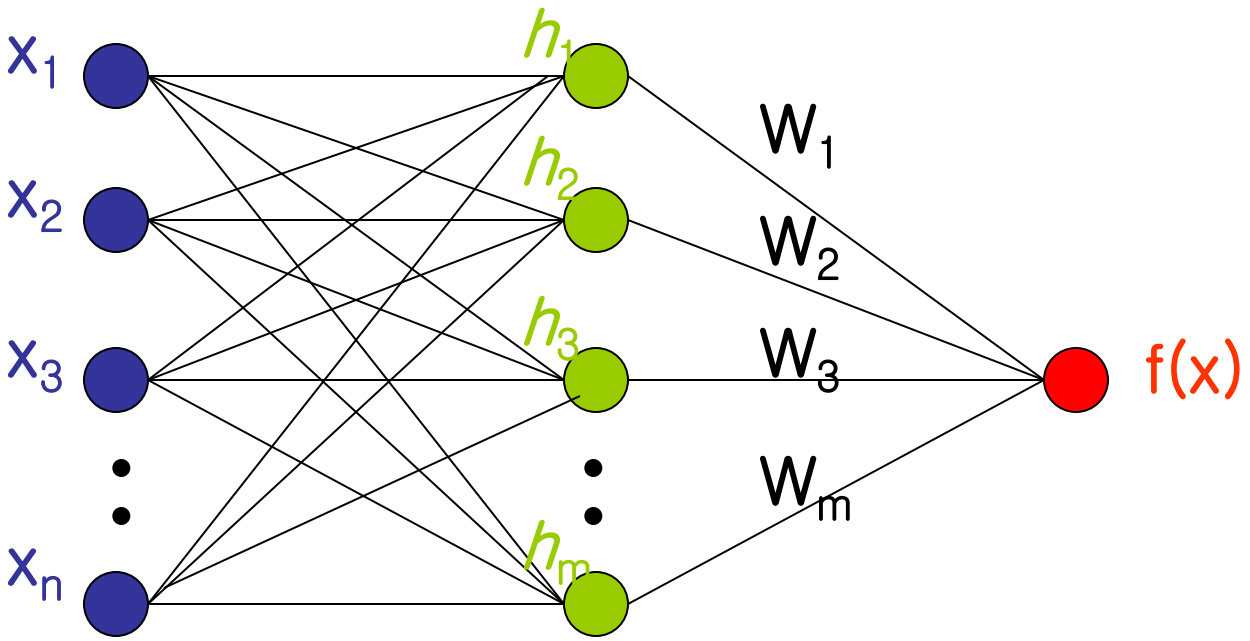
Hypersphere-spherically separable





Quadratically separable- Quadrics

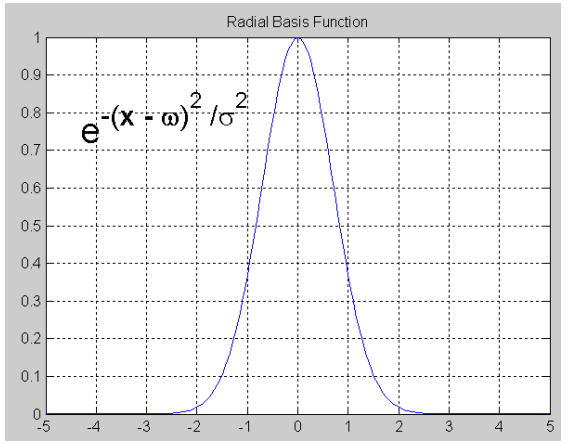
RBF Architecture



Three layers

- Input layer
  - Source nodes that connect to the network to its environment
- Hidden layer
  - Hidden units provide a set of basis function
  - High dimensionality
- Output layer
  - Linear combination of hidden functions

RBF Networks



d input nodes

H hidden layer RBFs (receptive fields)

c output nodes

x_1

x_2

$x_{(d-1)}$

x_d

U_{ji}

W_{kj}

φ_1

φ_j

φ_H

y_j

net_k

z_1

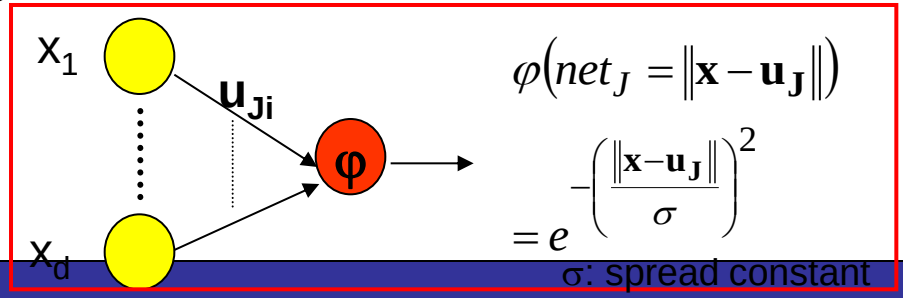
z_k

z_c

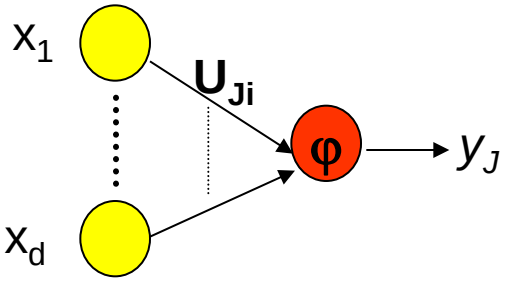
$$z_k = f(net_k) = f\left(\sum_{j=1}^{H} w_{kj} y_j\right) = \sum_{j=1}^{H} w_{kj} y_j$$

Linear act. function

$$\mathbf{U} = \mathbf{X}^T$$



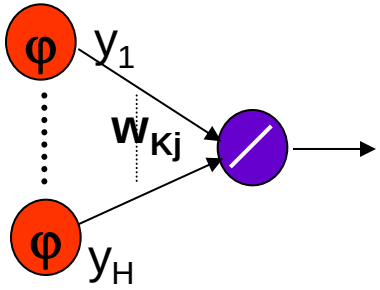
Principle of Operation



$$\varphi\left(net_J = \|\mathbf{x} - \mathbf{u_J}\|\right) = e^{-\left(\frac{\|\mathbf{x}-\mathbf{u_J}\|}{\sigma}\right)^2}$$

Euclidean Norm

σ: spread constant



$$z_k = f\left(net_k\right) = f\left(\sum_{j=1}^{H} w_{kj} y_j\right) = \sum_{j=1}^{H} w_{kj} y_j$$

Unknowns: $\mathbf{u_{ji}}$, $\mathbf{w_{kj}}$, σ

Radial basis function

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

$$h_j(x) = \exp(-(x - c_j)^2 / \sigma_j^2)$$

Where c_j is center of a region,

σ_j is width of the receptive field

designing

- Require
  - Selection of the radial basis function width parameter
  - Number of radial basis neurons

Selection of the RBF width para.

- Not required for an MLP
- smaller width
  - alerting in untrained test data
- Larger width
  - network of smaller size & faster execution

Number of radial basis neurons

- By designer
- Max of neurons = number of input
- Min of neurons = (experimentally determined)
- More neurons
  - More complex, but smaller tolerance

learning strategies

- Two levels of Learning
  - Center and spread learning (or determination)
  - Output layer Weights Learning
- Make # (parameters) small as possible
  - Principles of Dimensionality

Various learning strategies

- how the centers of the radial-basis functions of the network are specified.

1. Fixed centers selected at random
2. Self-organized selection of centers
3. Supervised selection of centers

Fixed centers selected at random(1)

- Fixed RBFs of the hidden units
- The locations of the centers may be chosen randomly from the training data set.
- We can use different values of centers and widths for each radial basis function -> experimentation with training data is needed.

Fixed centers selected at random(1)

- Only output layer weight is need to be learned.
- Obtain the value of the output layer weight by pseudo-inverse method
- Main problem
  - Require a large training set for a satisfactory level of performance

Self-organized selection of centers(2)

- Hybrid learning
  - self-organized learning to estimate the centers of RBFs in hidden layer
  - supervised learning to estimate the linear weights of the output layer
- Self-organized learning of centers by means of clustering.
- Supervised learning of output weights by LMS algorithm.

Self-organized selection of centers(2)

- k-means clustering
  1. Initialization
  2. Sampling
  3. Similarity matching
  4. Updating
  5. Continuation

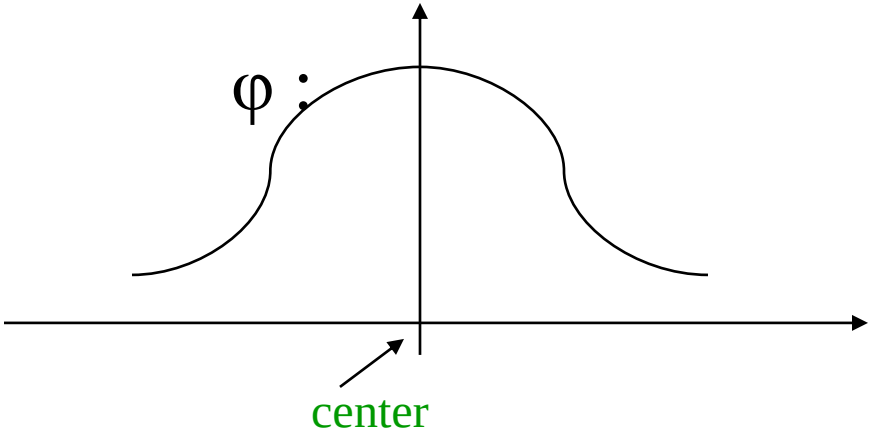
Supervised selection of centers(3)

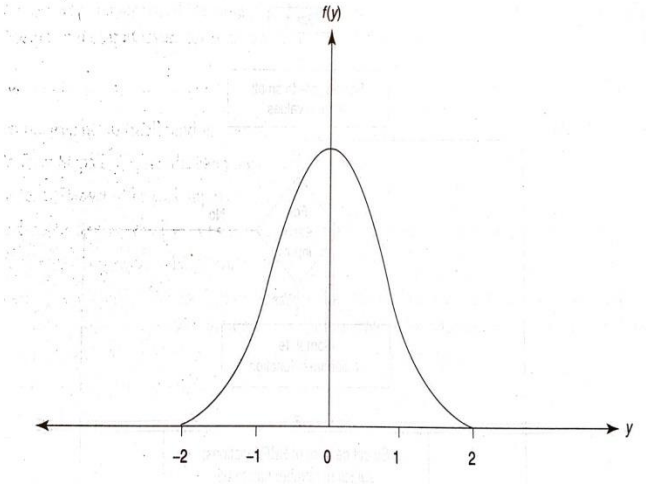
- All free parameters of the network are changed by supervised learning process.
- Error-correction learning using LMS algorithm.

What happens in Hidden layer?

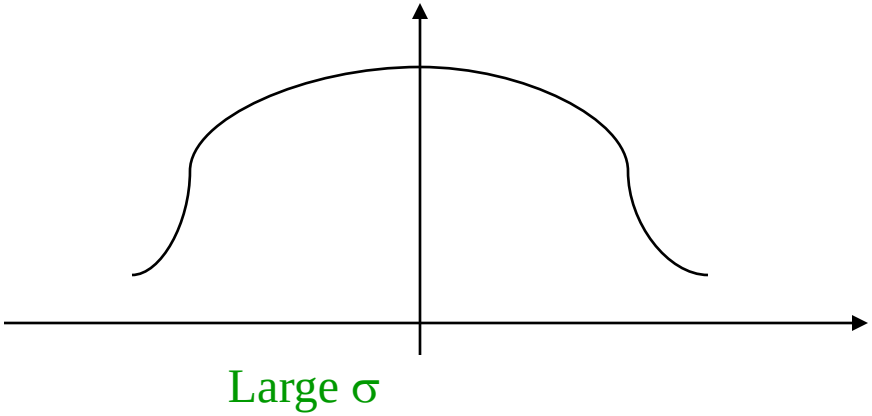
- The patterns in the input space form clusters
- If the centers of these clusters are known then the distance from the cluster center can be measured
- The most commonly used radial basis function is a Gaussian function
- In a RBF network r is the distance from the cluster centre

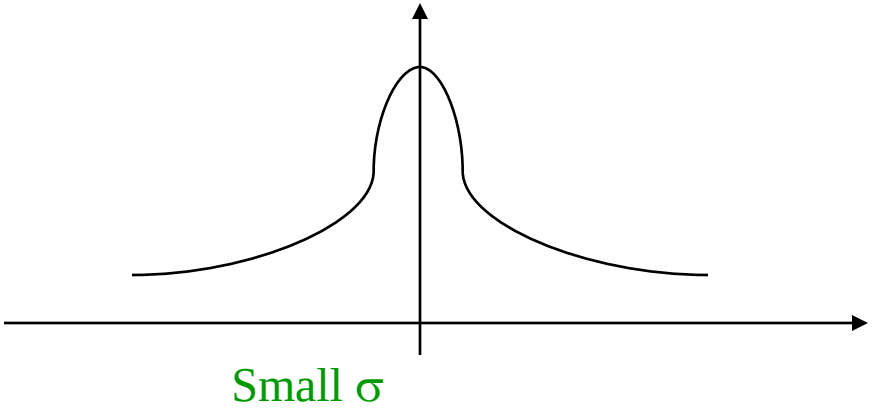
Gaussian RBF φ





σ is a measure of how spread the curve is:





Distance measure

- The distance measured from the cluster center is usually the Euclidean distance
- For each neuron in the hidden layer, the weights represent the co-ordinates from the center of the cluster
- When the neuron receives an input pattern X, the distance is found using the equation

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

Width of hidden unit

$$\phi j = \exp\left(-\frac{\sum_{i=1}^{n}(xi-\mu j)^2}{2\sigma^2}\right) \qquad 1$$

where $\sigma = \frac{d_{\max}}{\sqrt{2M}}$ 2

σ Is the width or radius of the bell shape and has to be determined empirically

M=no. of basis function μj =basis function centre

D_{max}=distance between them

$$\phi j = \exp\left(-\frac{M}{d^2_{\max}}\frac{\sum_{i=1}^{n}(xi-\mu j)^2}{}\right) \qquad 3$$

Training of the hidden layer

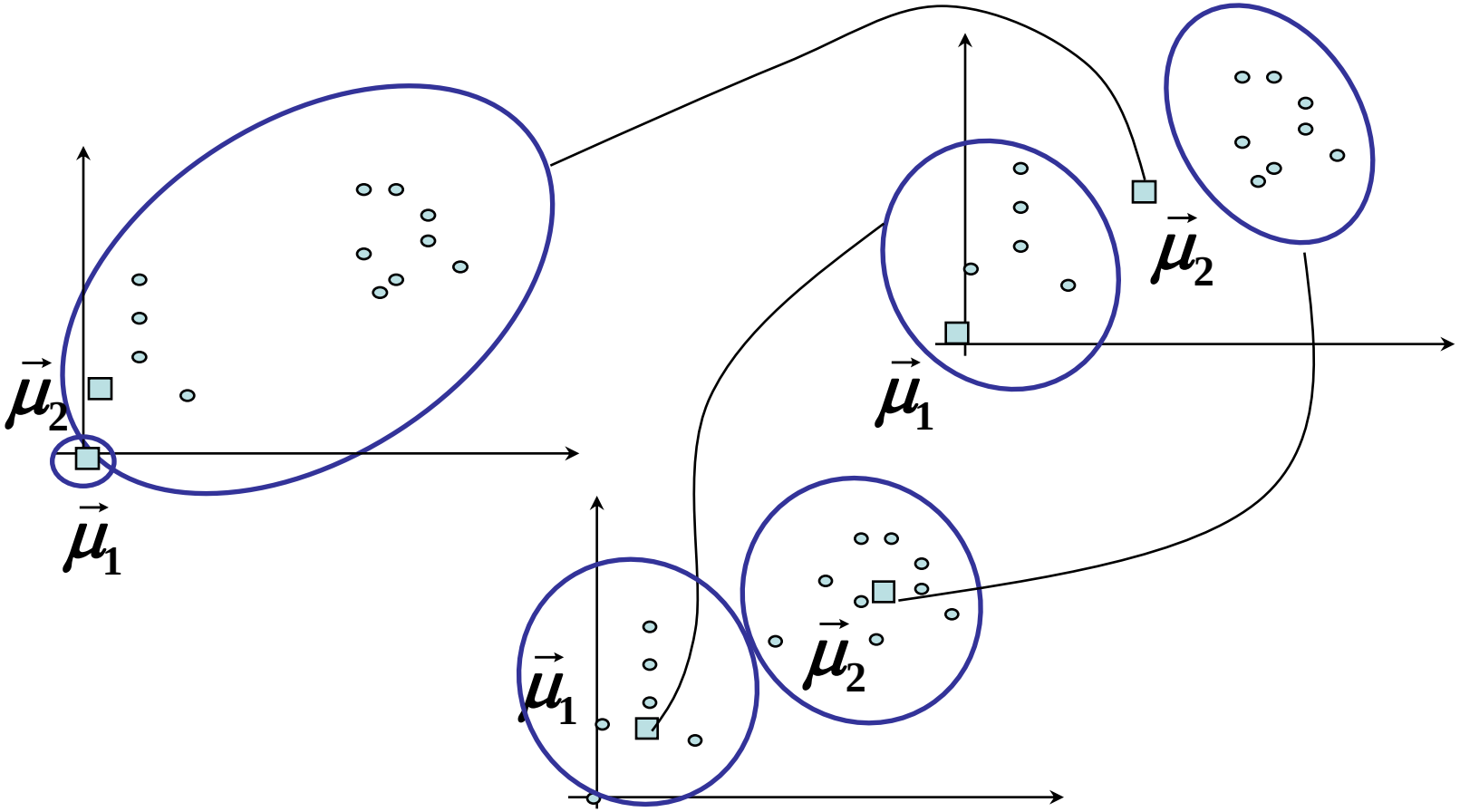
- The hidden layer in a RBF network has units which have weights corresponding to the vector representation of the center of the cluster
- These weights are found either by k-means clustering algo or kohonen's algorithm
- Training is unsupervised but the no. of clusters is set in advance. The algorithms finds the best fit to these clusters.

K-means algorithm

- Initially ‘k’ points in the pattern space are randomly set
- Then for each item of data in the training set, the distances are found from all of the ‘k’ centers
- The closest centre is chosen for each item of data. This is the initial classification, so all items of data will be assigned a class from 1 to ‘k’
- Then for all data which has been found to be in class 1, the average or mean values are found for each of the co-ordinates
- These become the new values for the centre corresponding to class 1
- This is repeated till class k-which generates k-new centers
- This process is repeated until there is no further change

2. Finding the RBF Parameters

- Use the *K*-mean algorithm to find c_i



K-mean Algorithm

step1: K initial clusters are chosen randomly from the samples to form K groups.

step2: Each new sample is added to the group whose mean is the closest to this sample.

step3: Adjust the mean of the group to take account of the new points.

step4: Repeat step2 until the distance between the old means and the new means of all clusters is smaller than a predefined tolerance.

Outcome: There are K clusters with means representing the centroid of each clusters.

Advantages: (1) A fast and simple algorithm.

(2) Reduce the effects of noisy samples.

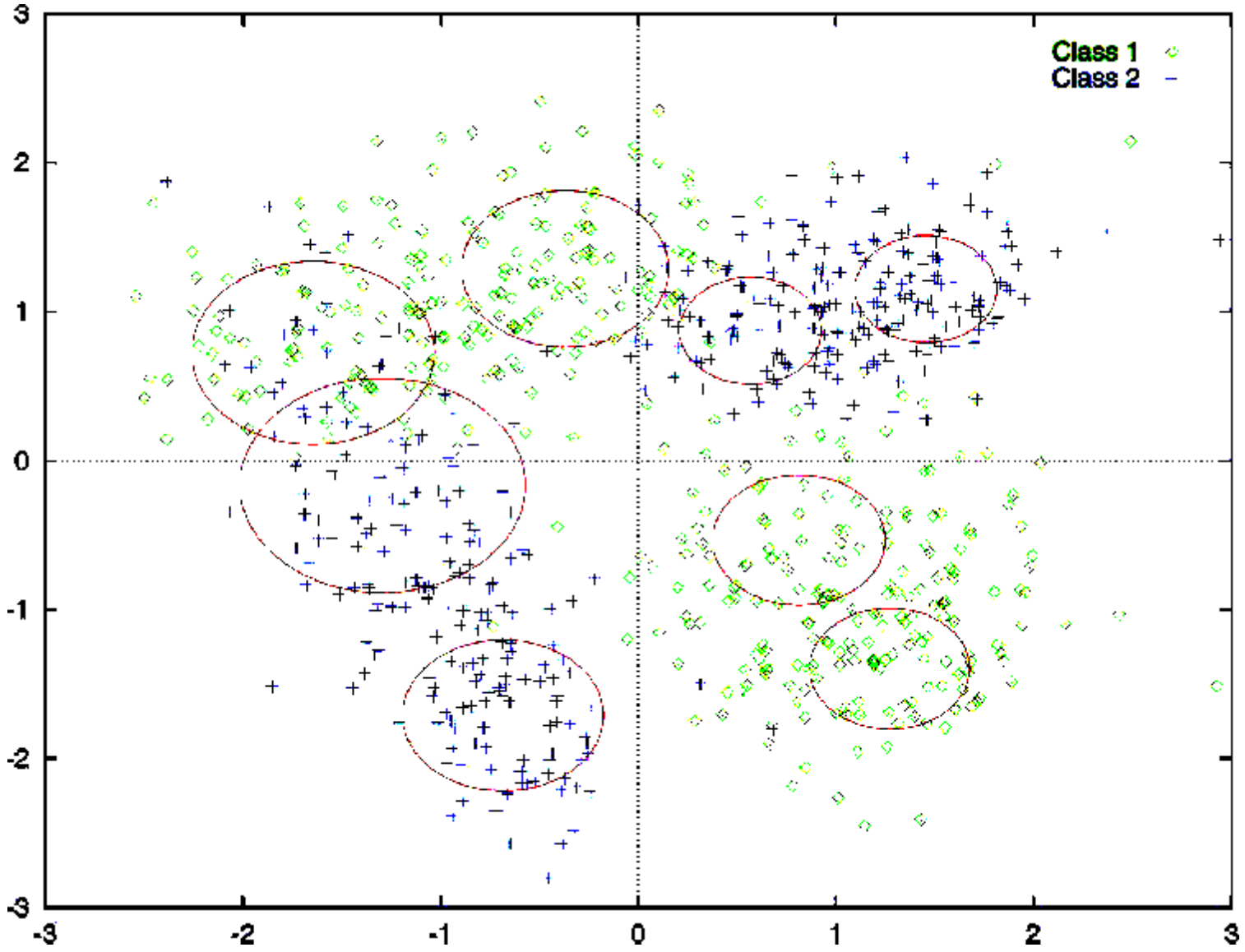
- **Use *K* nearest neighbor rule to find the function width σ**

$$\sigma_i = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left\|\vec{c}_k - \vec{c}_i\right\|^2}$$

***k*-th nearest neighbor of c_i**

- **The objective is to cover the training points so that a smooth fit of the training samples can be achieved**

Centers and widths found by K-means and K-NN



Adaptive k-means algorithm

- Similar to kohenen learning.
- Input patterns are presented to all of the cluster centers one at a time and the cluster centers adjusted after each one
- Cluster center that is nearest to the input data wins, and is shifted slightly towards the new data
- Online training can be done using kohenen algo.

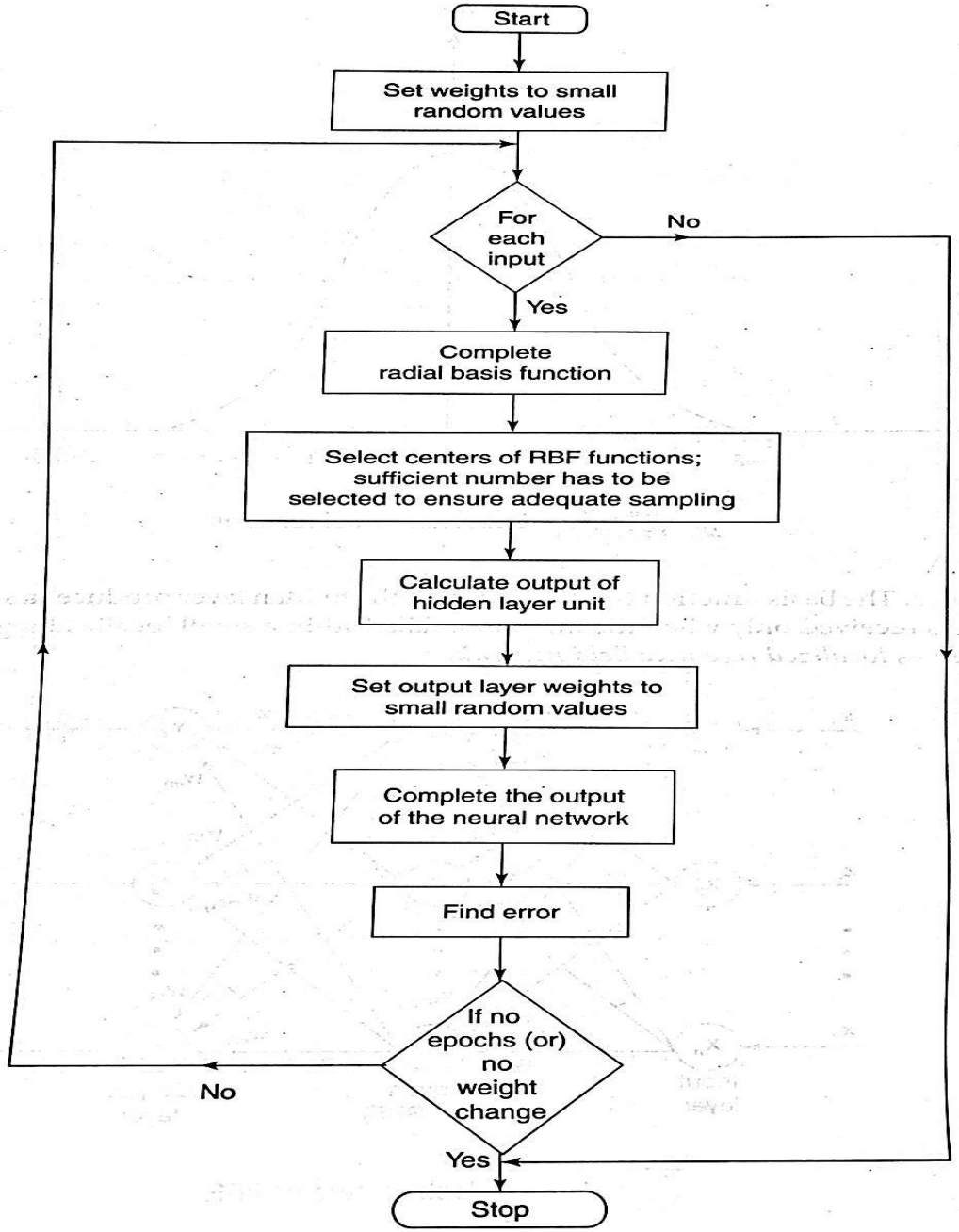
Training the output layer

- The output layer is trained using the least mean square algorithm, which is a gradient descent technique
- Given input signal vector x(n) and desired response d(n)
- Set initial weights w(x)=0
- For n=1,2,……..

  Compute

  $e(n)=error=d - w^t x$

  $w(n+1)=w(n)+c.x(n).e(n)$


Start
Set weights to small random values
For each input
No
Yes
Complete radial basis function
Select centers of RBF functions; sufficient number has to be selected to ensure adequate sampling
Calculate output of hidden layer unit
Set output layer weights to small random values
Complete the output of the neural network
Find error
If no epochs (or) no weight change
No
Yes
Stop

Step 0: Set the weights to small random values.

Step 1: Perform Steps 2–8 when the stopping condition is false.

Step 2: Perform Steps 3–7 for each input.

Step 3: Each input unit (x_i for all $i = 1$ to n) receives input signals and transmits to the next hidden layer unit.

Step 4: Calculate the radial basis function.

Step 5: Select the centers for the radial basis function. The centers are selected from the set of input vectors. It should be noted that a sufficient number of centers have to be selected to ensure adequate sampling of the input vector space.

Step 6: Calculate the output from the hidden layer unit:

$$v_i(x_i) = \frac{\exp\left[-\sum_{j=1}^{r}(x_{ji} - \hat{x}_{ji})^2\right]}{\sigma_i^2}$$

where $\hat{x}_{ji}$ is the center of the RBF unit for input variables; σ_i the width of ith RBF unit; x_{ji} the jth variable of input pattern.

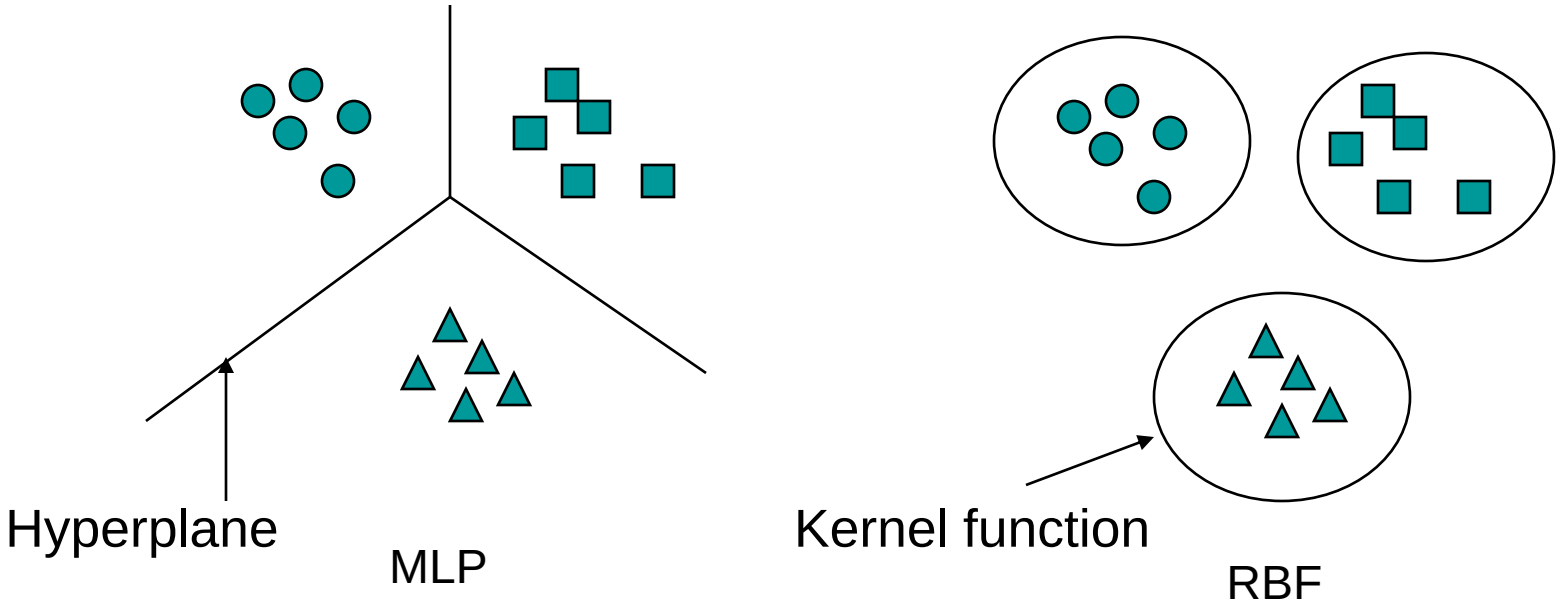
Step 7: Calculate the output of the neural network:

$$y_{net} = \sum_{i=1}^{k} w_{im} v_i(x_i) + w_0$$

where k is the number of hidden layer nodes (RBF function); y_{net} the output value of mth node in output layer for the nth incoming pattern; w_{im} the weight between ith RBF unit and mth output node; w_0 the biasing term at nth output node.

Step 8: Calculate the error and test for the stopping condition. The stopping condition may be number of epochs or to a certain extent weight change.

RBFN for Pattern Classification



The probability density function (also called conditional density function or likelihood) of the k-th class is defined as

$$p(\vec{x} \mid C_k)$$

MLP

1. The activation function can be any non linear function which can serve the purpose.
2. The final layer in MLP also uses the activation function before linearly combining it.
3. There can be more than one hidden layer in MLP.

RBF

1. The activation function is a function of the euclidean distance of input vector and a certain vector.
2. The final layer of RBF don’t use activation function, it rather linearly combines the output of the previous neuron.
3. There is only one hidden layer in RBF and one output layer.
4. The final layer have only one neuron.

- Typically RBF and MLP share a common neuron model.

Similarities between RBF and MLP

- Both are feedforward
- Both are universal approximators
- Both are used in similar application areas

Comparison with multilayer NN

RBF-Networks are used for regression and for performing complex (non-linear) pattern classification tasks.

Comparison between RBF networks and FFNN:

- Both are examples of *non-linear layered feed-forward* networks.
- Both are *universal approximators*.

Comparison with multilayer NN

- Architecture:
  - RBF networks have one *single* hidden layer.
  - FFNN networks may have *more* hidden layers.
- Neuron Model:
  - In RBF the neuron model of the hidden neurons is *different* from the one of the output nodes.
  - Typically in FFNN hidden and output neurons share a *common neuron model*.
  - The hidden layer of RBF is *non-linear*, the output layer of RBF is *linear*.
  - Hidden and output layers of FFNN are usually *non-linear*.

Comparison with multilayer NN

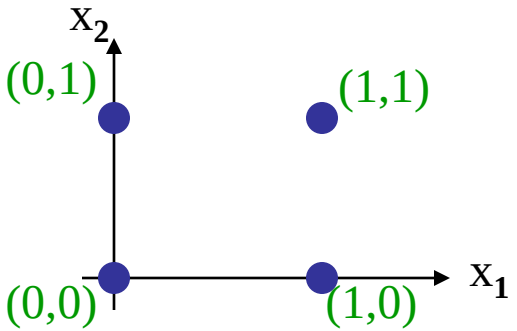
- Activation functions:
  - The argument of activation function of each hidden neuron in a RBF NN computes the *Euclidean distance* between input vector and the center of that unit.
  - The argument of the activation function of each hidden neuron in a FFNN computes the *inner product* of input vector and the synaptic weight vector of that neuron.
- Approximation:
  - RBF NN using Gaussian functions construct *local* approximations to non-linear I/O mapping.
  - FF NN construct *global* approximations to non-linear I/O mapping.

Differences between MLP and RBF

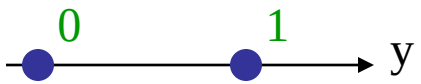
Parameter	MLP	RBF
Hidden layer	Can have any number of hidden layer	Can have only one hidden layer
Connectivity	Can be fully or partially connected	Has to be mandatorily completely connected
Processing Nodes	Processing nodes in different layers shares a common neural model	Hidden nodes operate very differently and have a different purpose
Activation Function	Argument of hidden function activation function is the inner product of the inputs and the weights	The argument of each hidden unit activation function is the distance between the input and the weights
Local/Global optimization	Trained with a single global supervised algorithm	RBF networks are usually trained one later at a time ,Local
Convergence	Not Guarantee	Almost guarantee
Training speed	Training is slower compared to RBF	Training is comparitely faster than MLP
Response time	After training MLP is much faster than RBF	After training RBF is much slower than MLP
Memory requirement	Very Small	Very large
Generalization	Usually poor	Usually better

Example: XOR problem

- Input space:



- Output space:



- Construct an RBF pattern classifier such that:
  (0,0) and (1,1) are mapped to 0, class C1
  (1,0) and (0,1) are mapped to 1, class C2

Example: XOR problem

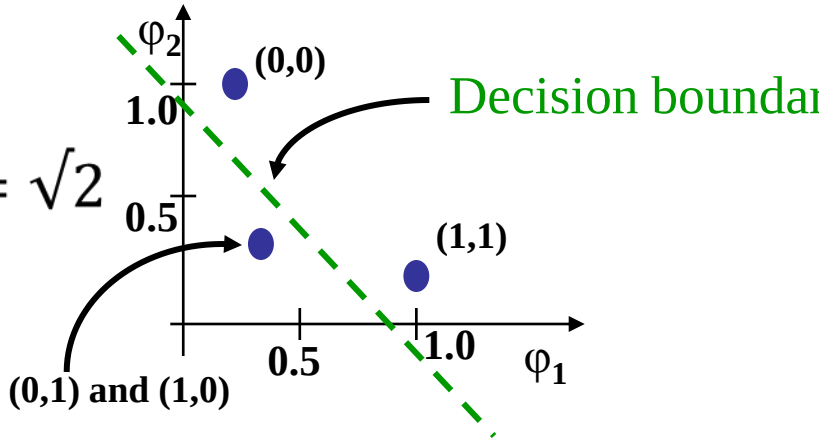
- In the feature (hidden layer) space:

$$\text{no.of patterns}=4$$

$M=2$

$\mu_1=(0,0);\ \mu_2=(1,1)$

$$d_{\max} = \sqrt{((0-1)^2 + (0-1)^2)} = \sqrt{2}$$



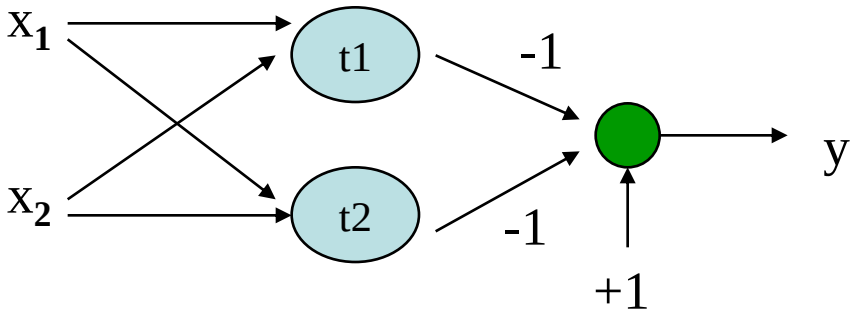
- When mapped into the feature space < φ_1 , φ_2 > (hidden layer), C1 and C2 become *linearly separable.* So a linear classifier with $\varphi_1(x)$ and $\varphi_2(x)$ as inputs can be used to solve the XOR problem.

Example: XOR problem

$$\varphi_1(x) = e^{-\|x-\mu_1\|^2}$$

$$\varphi_2(x) = e^{-\|x-\mu_2\|^2}$$

with $\mu_1 = (0,0)$ and $\mu_2 = (1,1)$

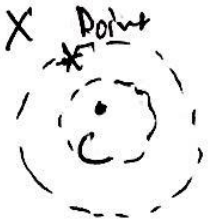


Pattern	X1	X2	φ1	φ2
1	0	0		
2	0	1		
3	1	0		
4	1	1		

RBF XOR

ϕ_1, ϕ_2 ... Radial Basis function

What is RBF < Receptor (Centroid) has more radially from this Spread



ϕ is constant

Multiquadric

$$\phi(r) = (r^2 + c^2)^{1/2} \quad c > 0$$

Inverse multiquadrics

$$\phi(r) = \frac{1}{(r^2 + c^2)^{1/2}} \quad \text{for } c > 0$$

or Gaussian function:

$$\phi(r) = \exp\left[-\frac{r^2}{2\sigma^2}\right] \quad \text{for } \sigma > 0$$

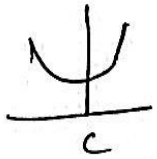
r is the measure of distance of point c

at Point x $r = \|x - c\|$

Point → Center

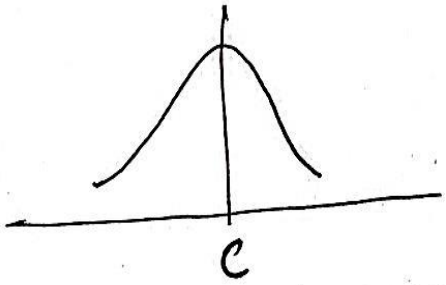
as $r \uparrow$ $\phi(r) \uparrow$

Multi Quadred



Inverse → $r \uparrow$ $\phi(r) \downarrow$

(*) Gaussian



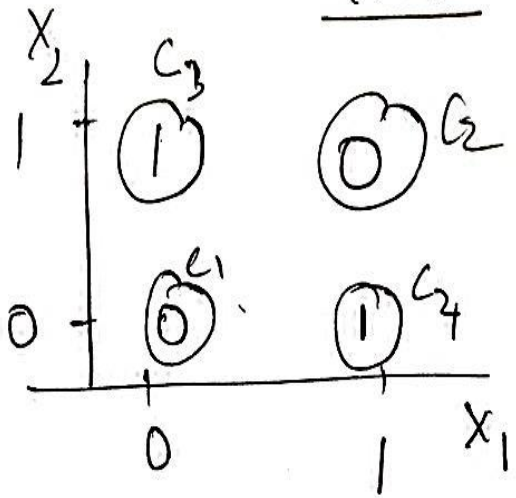
at c ϕ Max.

as none $\phi \downarrow$

Most Commonly used RBF

Non Liner → Linear

XOR



RBF Gaussia

$$\phi(x) = e^{-\|x-c\|^2}$$

↳ Receptor or Center

Consider two c (1 & 1)

Consider two c 1 & 1

Non liner transformation

	O/P (00) ϕ_1	(11) Center ϕ_2
0 0	1	0.1
0 1	0.4	0.4
1 0	0.4	0.4
1 1	0.1	1

00 mapped to 1. 0.1

01 mapped to 0.4 0.4

HP

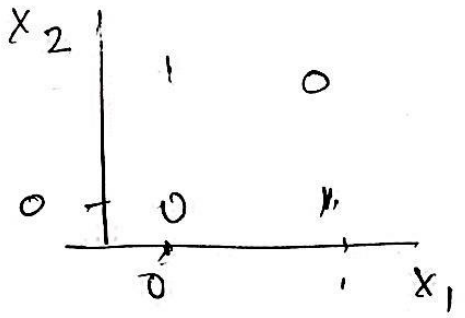
$$\phi_1(x) = e^{-\|x-c_1\|^2}$$

$$\phi_2(x) = e^{-\|x-c_2\|^2}$$

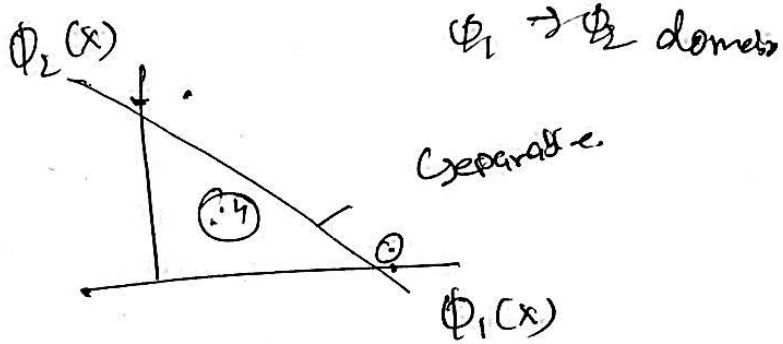
assm $2\sigma^2 = 1$

$$\phi_1(00) = \phi_1 = e^{-0} = 1$$

Plot

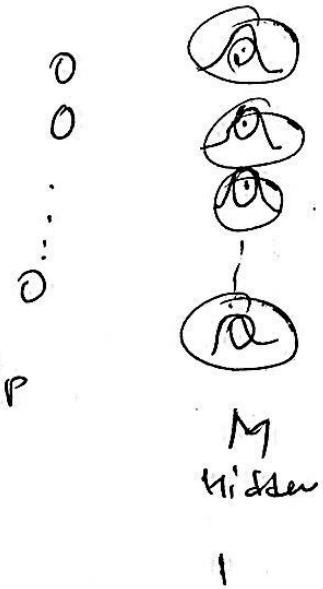


Non Linear.



Linearly Separable.

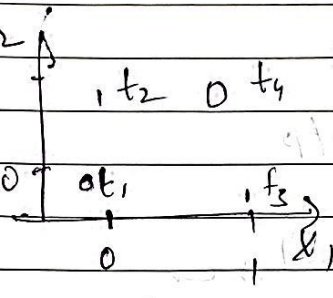
Arch



RBF $\phi_i(x) = e^{\frac{-\|x - t_i\|^2}{2\sigma_i^2}}$

by using this concept we can make a linear classification function

XOR Operation



two dimensional binary feature Vector

↳ Cast Pt to 4 Dimensional RBF

RBF ↳ Receptor

$\phi_1 = t_1 = 0,0$ Receptor of RBF σ_1 Spread

ϕ_2 $t_2 = 0,1$ " " " σ_2

ϕ_3 $t_3 = 1,0$ " " σ_3

ϕ_4 $t_4 = 1,1$ " " σ_4

for each receptor we have to find out K Nearest Neighbour

take $P = 2$ two Nearest Neighbour

$\sigma_1 = 1, \sigma_2 = 1, \sigma_3 = 1, \sigma_4 = 1$

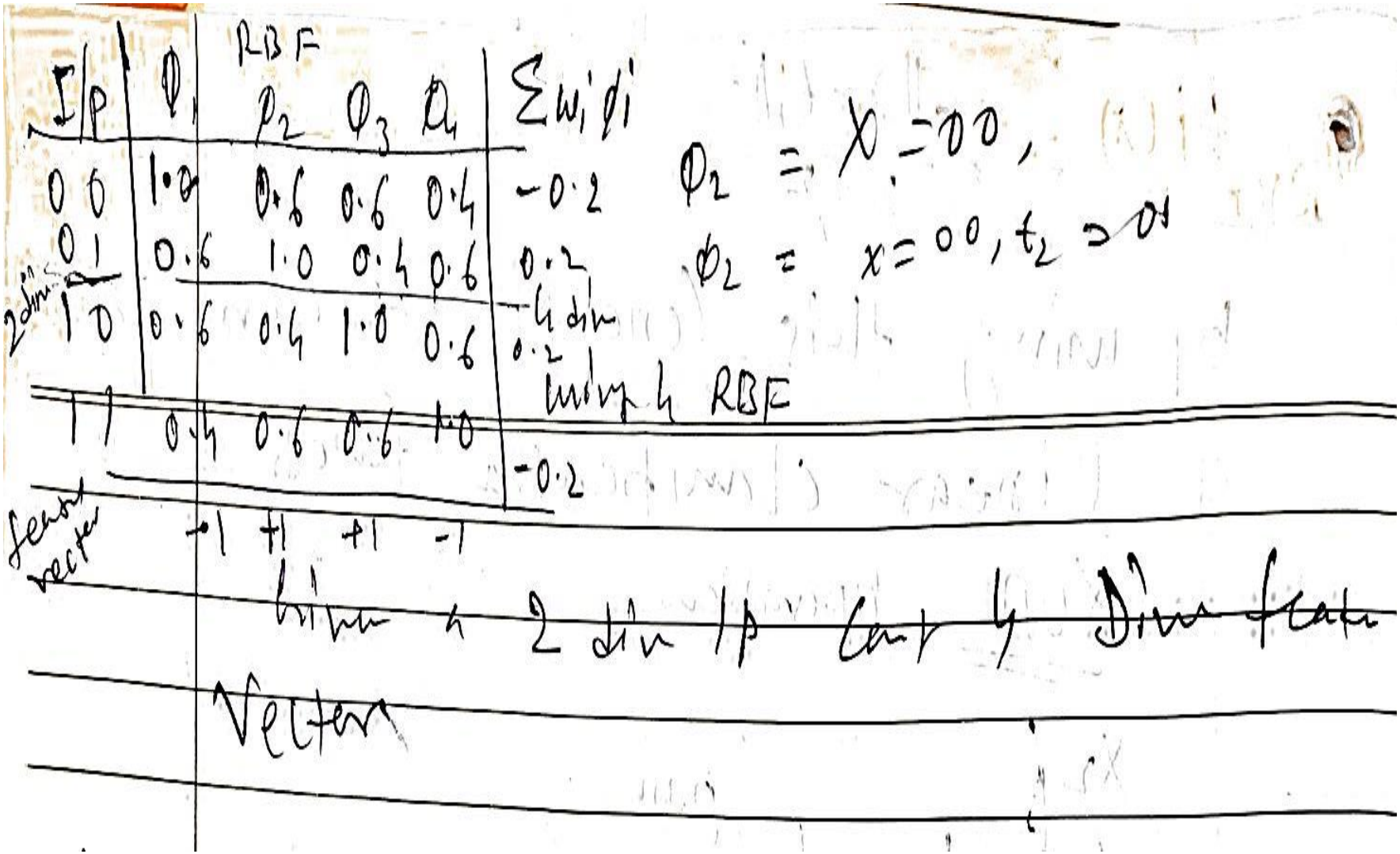
$$\phi_1(X) = e^{\frac{-\|X - t_1\|^2}{2\cdot\sigma_1^2}}$$

$$\phi_2(X) = e^{\frac{-\|X - t_2\|^2}{2}}$$

$$\phi_3(X) = e^{\frac{-\|X - t_3\|^2}{2}}$$

$$\phi_4(X) = e^{\frac{-\|X - t_4\|^2}{2}}$$

2 to 4 Dimn


I/P
RBF
ϕ_1 ϕ_2 ϕ_3 ϕ_4
$\Sigma w_i \phi_i$
0 0 1.0 0.6 0.6 0.4 −0.2
0 1 0.6 1.0 0.4 0.6 0.2
1 0 0.6 0.4 1.0 0.6 0.2
1 1 0.4 0.6 0.6 1.0 −0.2
$\phi_2 = x = 00$,
$\phi_2 = x = 00, t_2 = 01$
2 dim
4 dim
using 4 RBF
Feature vector
+1 +1 +1 −1
Vector
bipolar 4 2 dim I/P Conv 4 Dim feature

linear Combination o/p

−1 +1 +1 −1

Dimension of o/p layer

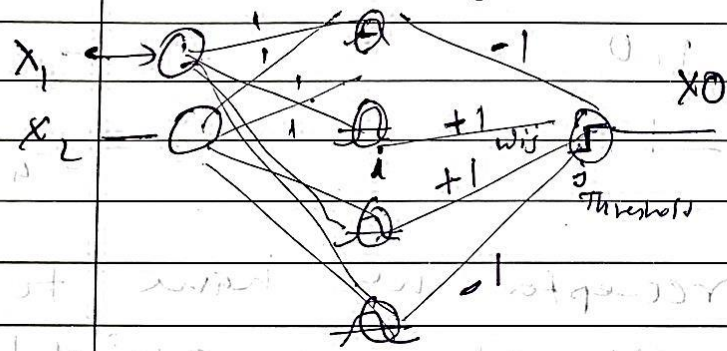
$-0.2 = 0$

$0.2 = 1$

$0.2 = 1$

$-0.2 = 0$

So Architecture of RBF



So By using RBF single layer Perceptron Can Solve the Prob. of XOR Prob.

This can write in Matrix

$$\begin{bmatrix} \phi_{11} & \phi_{21} & \phi_{31} & \cdots & \phi_{M1} \\ \phi_{12} & \phi_{22} & \phi_{32} & \cdots & \phi_{M2} \\ \vdots & & & & \\ \phi_{1M} & \phi_{2M} & \phi_{3M} & & \phi_{MM} \end{bmatrix} \begin{bmatrix} w_{1j} \\ w_{2j} \\ \vdots \\ w_{Mj} \end{bmatrix} = \begin{bmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{Nj} \end{bmatrix}$$

ϕ — w — o/p of the hidden

This indicate Linear equation of H_2O

$b_{ij} = 1$ if $x_i \in w_j$

$= 0$ " " $\notin w_j$

$b_{ij} = 1$ if $x_j \in w_j$

$= 0$ " " $\notin w_j$

This Matrix eqn can be written as

$$\phi w_j = b_j$$

if all w_j got trained

If Not Satisfied

Define error $e = \phi w_j - b_j$

Weight adaptation so that

e will be minimum

Define $J(w_j) = \| \phi w_j - b_i \|^2$ Norm

$\nabla J(w_j) = 2\phi^t (\phi w_j - b_j) = 0$

Gradient

$$w_j = \underbrace{(\phi^t \phi)^{-1} \phi^t}_{\text{Pseudoinverse } \phi^+} b_j$$

Example: XOR problem

- **What do we have to learn for a RBF NN with a given architecture?**
  - The centers of the RBF activation functions
  - the spreads of the Gaussian RBF activation functions
  - the weights from the hidden to the output layer
- Different learning algorithms may be used for learning the RBF network parameters. We describe three possible methods for learning centers, spreads and weights.

Learning Algorithm 1

- **Centers: are selected at random**
  - **centers** are chosen randomly from the training set
- $$\sigma = \frac{\text{Maximum distance between any 2 centers}}{\sqrt{\text{number of centers}}} = \frac{d_{max}}{\sqrt{m_1}}$$

i

- Then the activation function of hidden neuron becomes:

$$\varphi_i(\|x\|) = \exp\left(-\frac{m_1}{d_{max}^2}\|x - \mu_i\|^2\right)$$

Learning Algorithm 1

- **Weights:** are computed by means of the **pseudo-inverse method.**
  - **For an example** (x_i, d_i) **consider the output of the network**

$$y(x_i) = w_1\varphi_1(\| x_i - t_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x_i - t_{m1} \|)$$

  - **We would like** $y(x_i) = d_i$ **for each example, that is**

$$w_1\varphi_1(\| x_i - t_1 \|) + \ldots + w_{m1}\varphi_{m1}(\| x_i - t_{m1} \|) = d_i$$

Learning Algorithm 1

- This can be re-written in matrix form for one example

$$[\varphi_1(\| x_i - t_1 \|) \ldots \varphi_{m1}(\| x_i - t_{m1} \|)][w_1 \ldots w_{m1}]^T = d_i$$

and

$$\begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) \ldots \varphi_{m1}(\| x_1 - t_{m1} \|) \\ \ldots \\ \varphi_1(\| x_N - t_1 \|) \ldots \varphi_{m1}(\| x_N - t_{m1} \|) \end{bmatrix} [w_1 \ldots w_{m1}]^T = [d_1 \ldots d_N]^T$$

for all the examples at the same time

Learning Algorithm 1

let

$$\Phi = \begin{bmatrix} \varphi_1(\| x_1 - t_1 \|) & \dots & \varphi_{m1}(\| x_N - t_{m1} \|) \\ & \dots & \\ \varphi_1(\| x_N - t_1 \|) & \dots & \varphi_{m1}(\| x_N - t_{m1} \|) \end{bmatrix}$$

then we can write

$$\Phi \begin{bmatrix} w_1 \\ \dots \\ w_{m1} \end{bmatrix} = \begin{bmatrix} d_1 \\ \dots \\ d_N \end{bmatrix}$$

If Φ^+ is the pseudo-inverse of the Φ matrix we obtain the weights using the following formula

$$[w_1 \dots w_{m1}]^T = \Phi^+ [d_1 \dots d_N]^T$$

Learning Algorithm 1

1. Choose the centers randomly from the training set.

2. Compute the spread for the RBF function using the normalization method.

3. Find the weights using the pseudo-inverse method.

Exercise

- Check what happens if you choose two different basis function centres

Output weights

- We have one output y(x) with one weight wj to each hidden unit j, and one bias -θ
- $y(x) = w_1 \emptyset_1(x) + w_2 \emptyset_2(x) - \theta$

Based on the values computed for RBF and the patterns we get 4 equations from which the weights w1 and w2 can be computed

Learning Algorithm 2

- ***Hybrid Learning Process:***
  - **Clustering** for finding the centers.
  - Spreads chosen by normalization.
  - **LMS algorithm (Adaline)** for finding the weights.

Application: FACE RECOGNITION

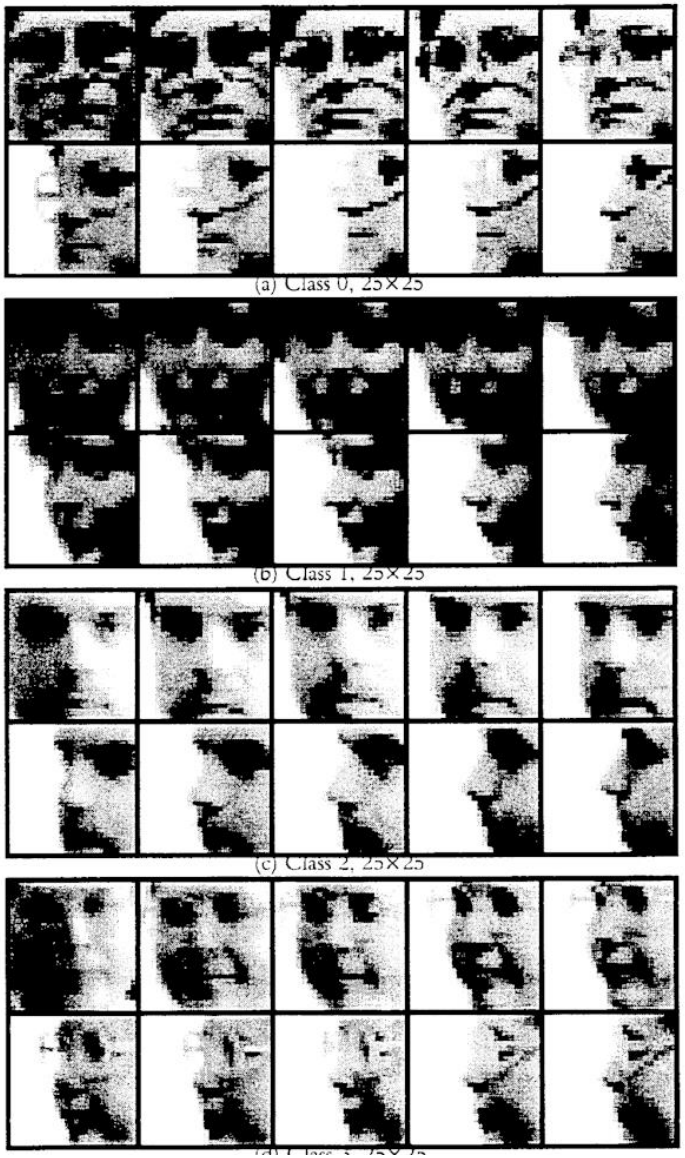
- The problem:
  - Face recognition of persons of a known group in an indoor environment.
- The approach:
  - Learn face classes over a wide range of poses using an RBF network.

Dataset

- **database**
  - **100 images of 10 people** (8-bit grayscale, resolution 384 x 287)
  - for each individual, 10 images of head in different pose **from face-on to profile**
  - Designed to asses performance of **face recognition techniques** when pose variations occur

Datasets

All ten images for classes 0-3 from the Sussex database, nose-centred and subsampled to 25x25 before preprocessing



Approach: Face unit RBF

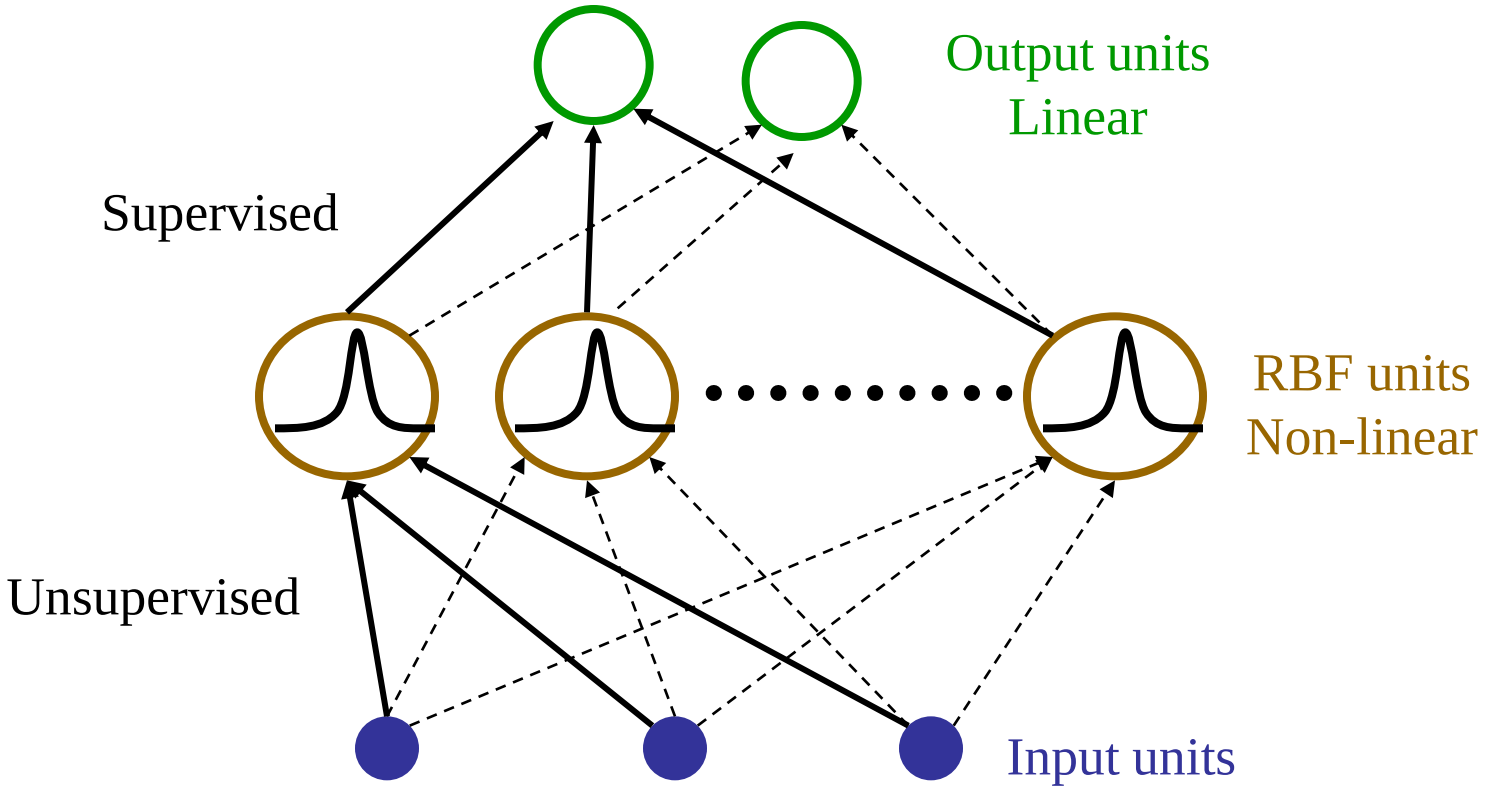
- A face recognition unit RBF neural networks is trained to recognize a single person.
- Training uses examples of images of the person to be recognized as positive evidence, together with selected confusable images of other people as negative evidence.

Network Architecture

- Input layer contains 25*25 inputs which represent the pixel intensities (normalized) of an image.
- Hidden layer contains p+a neurons:
  - **p** hidden pro neurons (receptors for positive evidence)
  - **a** hidden anti neurons (receptors for negative evidence)
- Output layer contains two neurons:
  - One for the particular person.
  - One for all the others.

The output is discarded if the absolute difference of the two output neurons is smaller than a parameter R.

RBF Architecture for one face recognition



Hidden Layer

- Hidden nodes can be:
  - Pro neurons: Evidence for that person.
  - Anti neurons: Negative evidence.
- The number of pro neurons is equal to the positive examples of the training set. For each pro neuron there is either one or two anti neurons.
- Hidden neuron model: Gaussian RBF function.

Training and Testing

- Centers:
  - of a pro neuron: the corresponding positive example
  - of an anti neuron: the negative example which is most similar to the corresponding pro neuron, with respect to the Euclidean distance.
- Spread: average distance of the center from all other centers. So the spread σ_n of a hidden neuron n is

$$\sigma_n = \frac{1}{H\sqrt{2}} \sum_h \| t^n - t^h \|$$

where H is the number of hidden neurons and t^i is the center of neuron i.

- Weights: determined using the pseudo-inverse method.
- A RBF network with 6 pro neurons, 12 anti neurons, and R equal to 0.3, discarded 23 pro cent of the images of the test set and classified correctly 96 pro cent of the non discarded images.

References

- https://www.cc.gatech.edu/~isbell/tutorials/rbf-intro.pdf
- https://www.slideshare.net/sheetalkatkar/radial-basis-function-network
- https://anuradhasrinivas.files.wordpress.com/2012/08/rbf.ppt
- https://slideplayer.com/slide/6641131/
- http://users.rowan.edu/~shreek/spring02/ann/lectures/RBFlecture1.ppt