EE5904/ME5404:
Lecture Five
Radial-Basis Function Networks

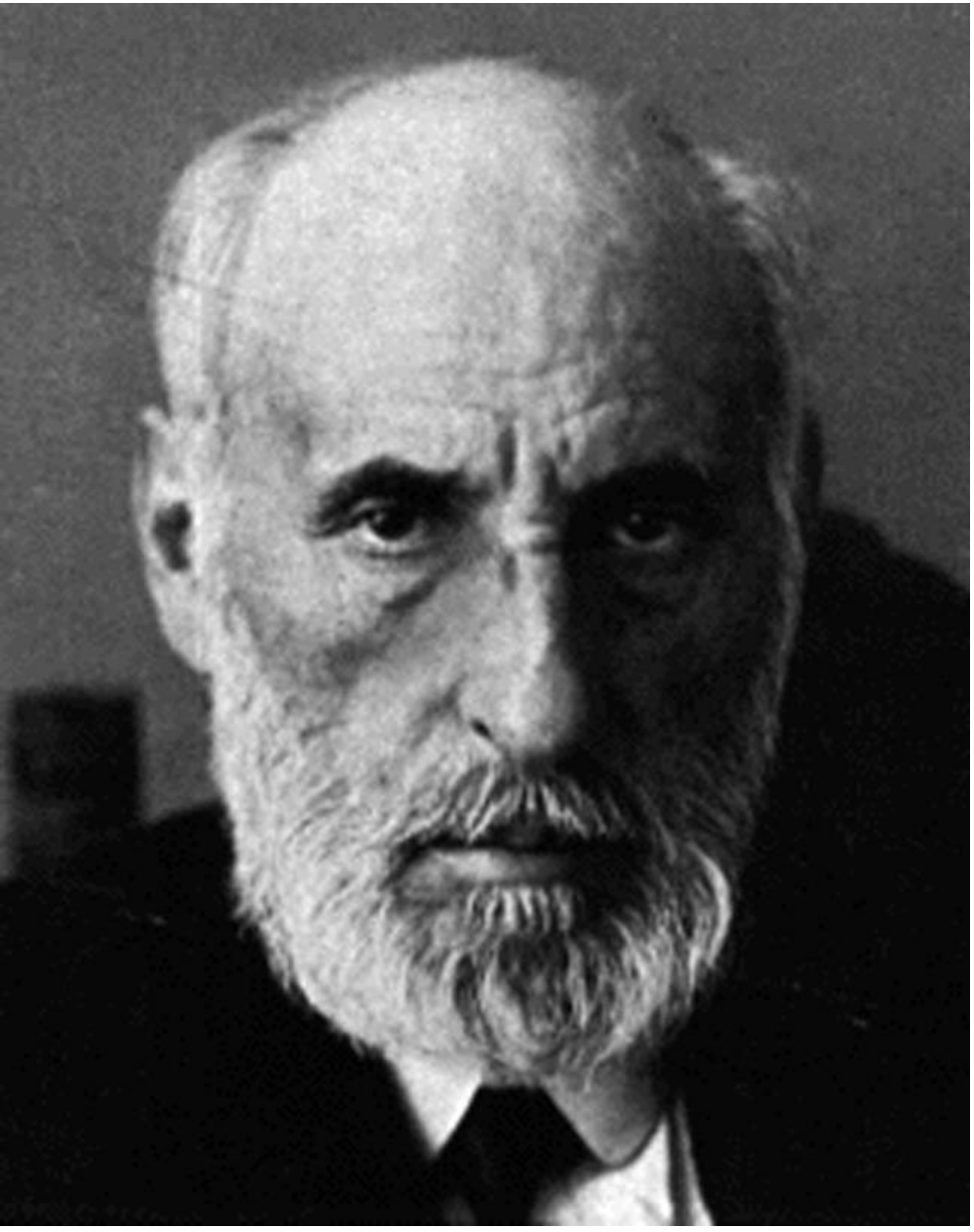
Xiang Cheng

Associate Professor

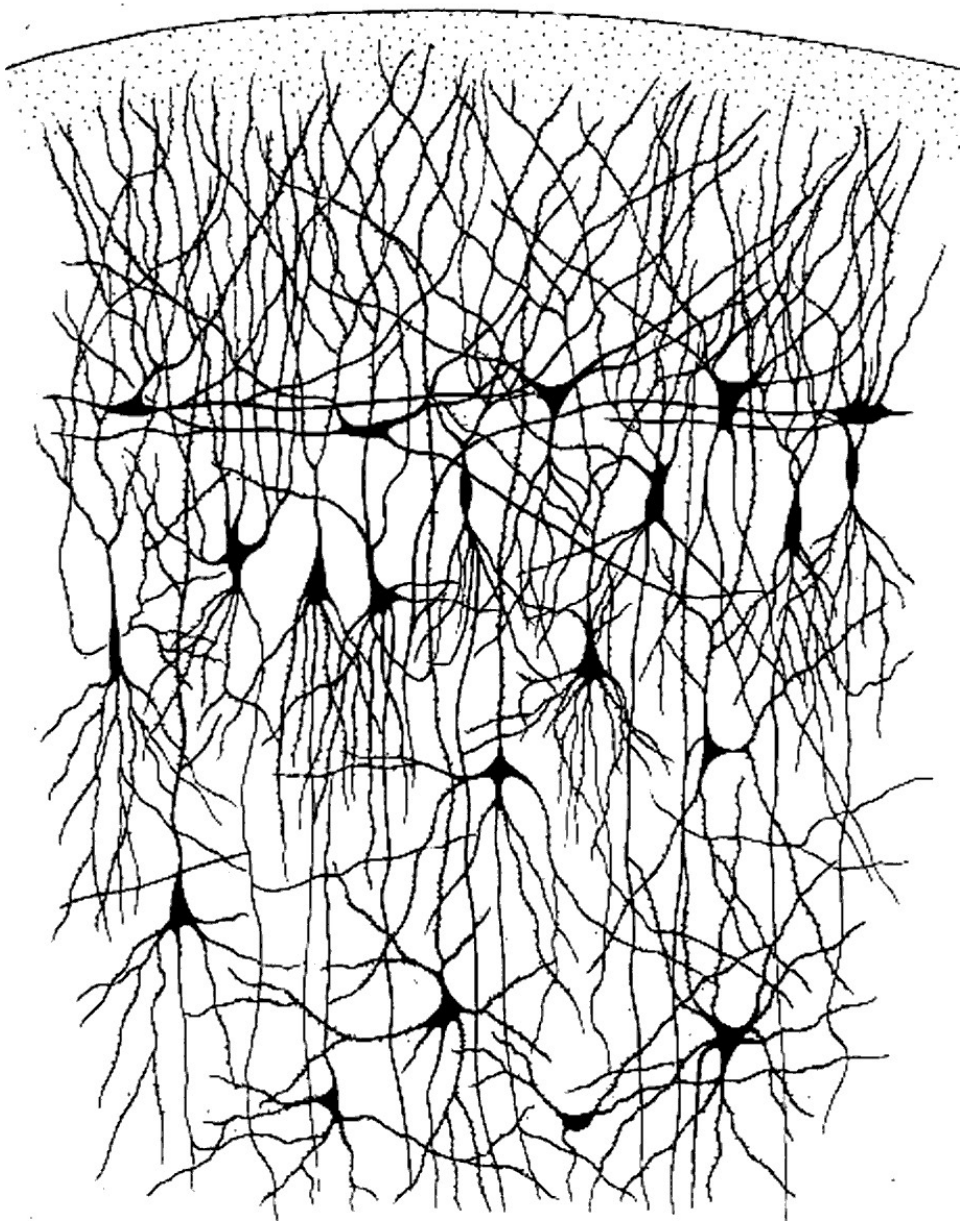
Department of Electrical & Computer Engineering
The National University of Singapore

Phone: 65166210 Office: Block E4-08-07
Email: elexc@nus.edu.sg

The understanding of neuron started more than 100 years ago:

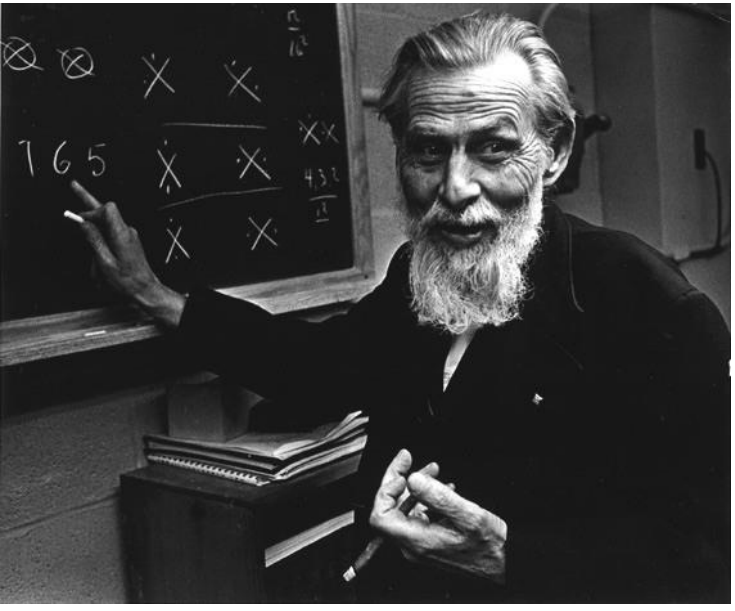


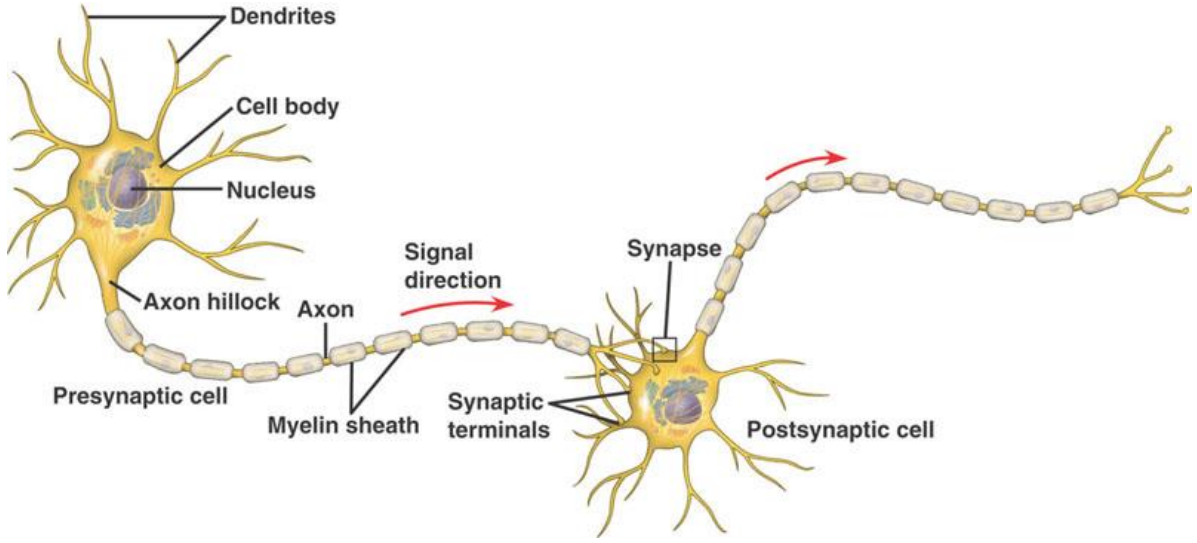
Santiago Ramon y Cajal 1852-1934

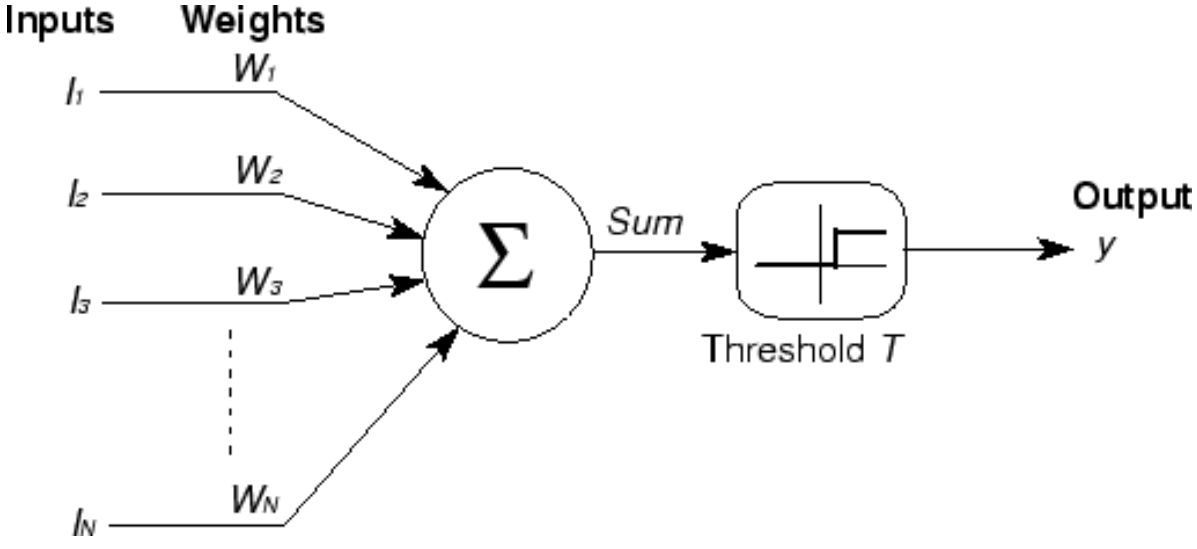


The beginning of the artificial neural networks

McCulloch and Pitts, 1943



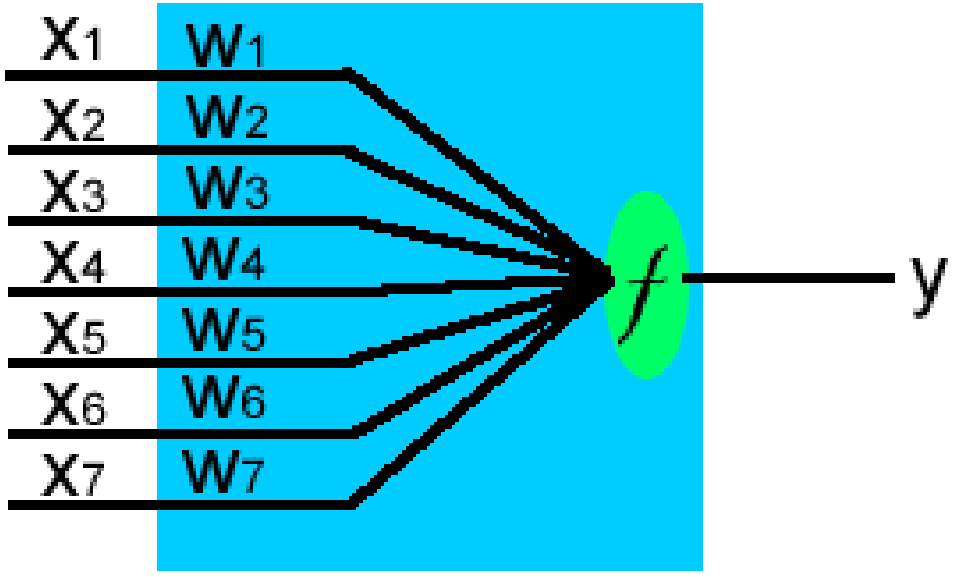




The next major step: Perceptron—single layer neural networks

Frank Rosenblatt, 1958

Frank Rosenblatt
(1928-1969)



Supervised learning:

$$w(n+1) = w(n) + \eta e(n) x(n)$$

$$e(n) = d(n) - y(n)$$

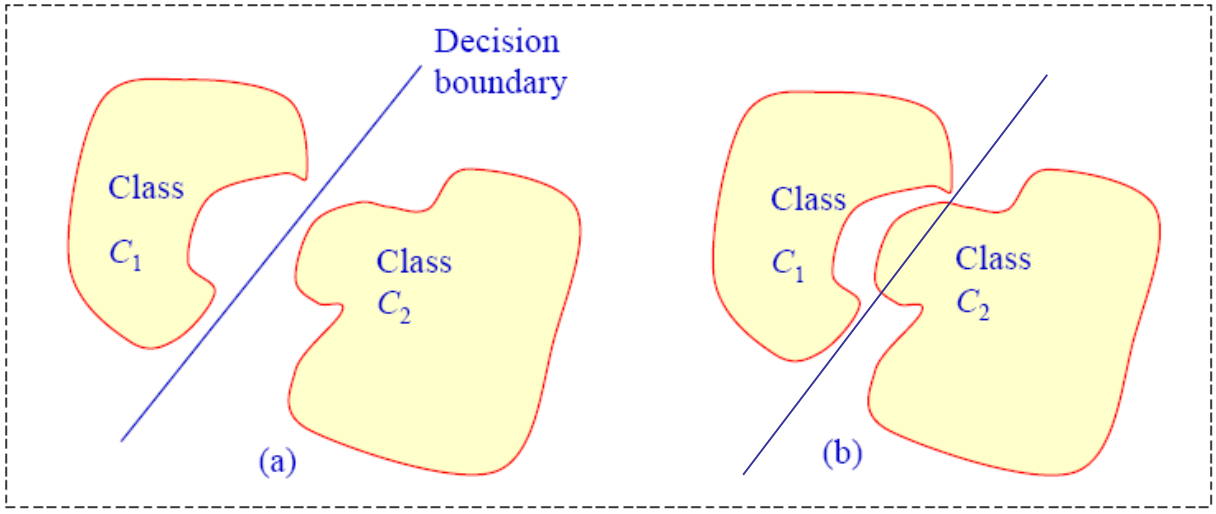
The weights were initially random. Then it could learn to perform certain simple tasks in pattern recognition.

Rosenblatt proved that for a certain class of problems, it could learn to behave correctly!

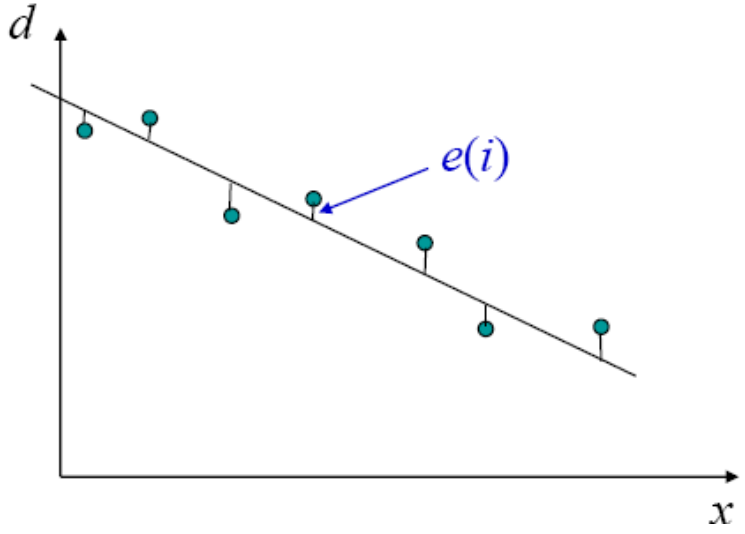
What is the fundamental limit of perceptron?

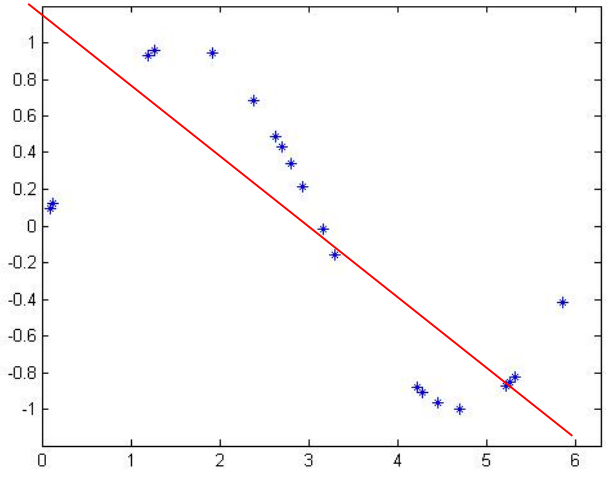
The fundamental limits of Single Layer Perceptrons

For Pattern Recognition Problem: Linearly Separable



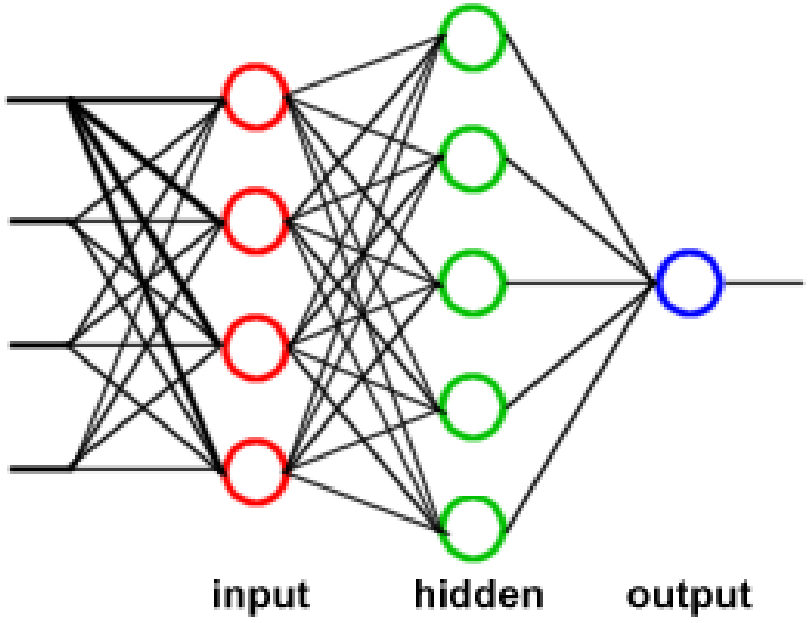
For Regression Problem: The process has to be close to a linear model!





David Rumelhart (1942-2011)

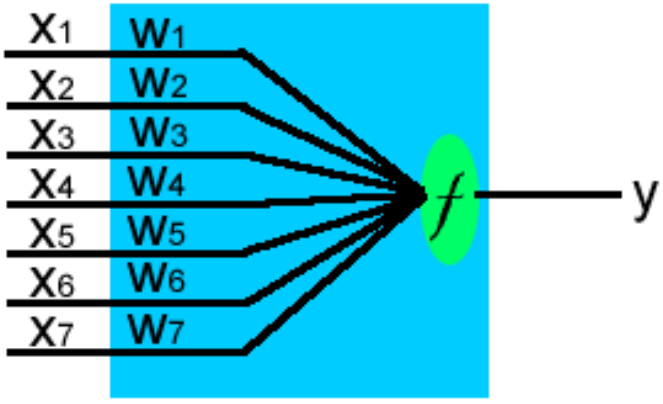
Multilayer Perceptron (MLP) and Back Propagation
David Rumelhart and his colleagues, 1986

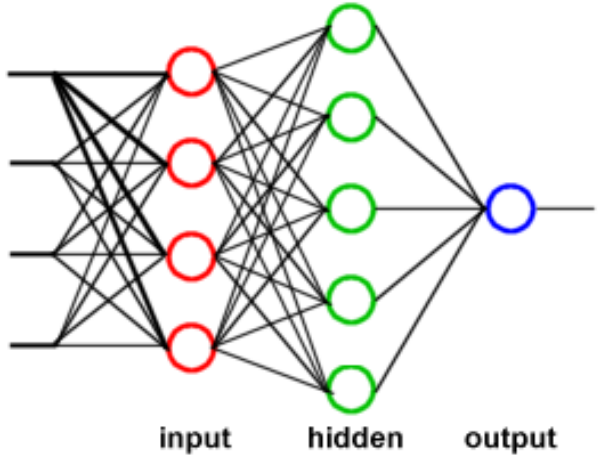


It was proved later that MLP is a universal function approximator, which can approximate any continuous function.

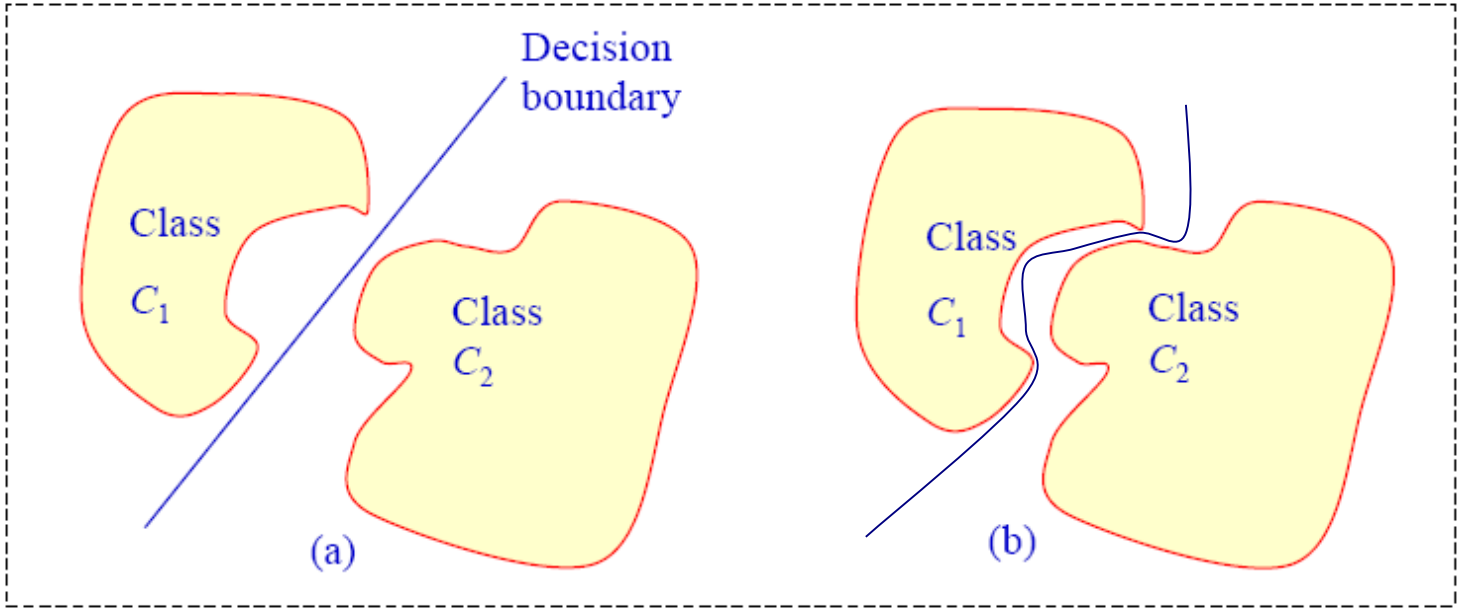
The Back Propagation allgorithm can be easily implemented to adjust the synaptic weights.

Single Layer Perceptron v.s. Multi-layer Perceptrons





Pattern Recognition Problem:

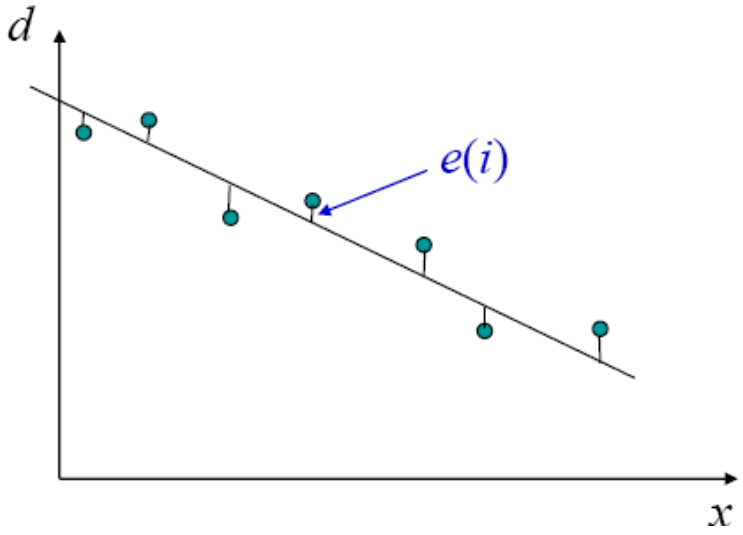


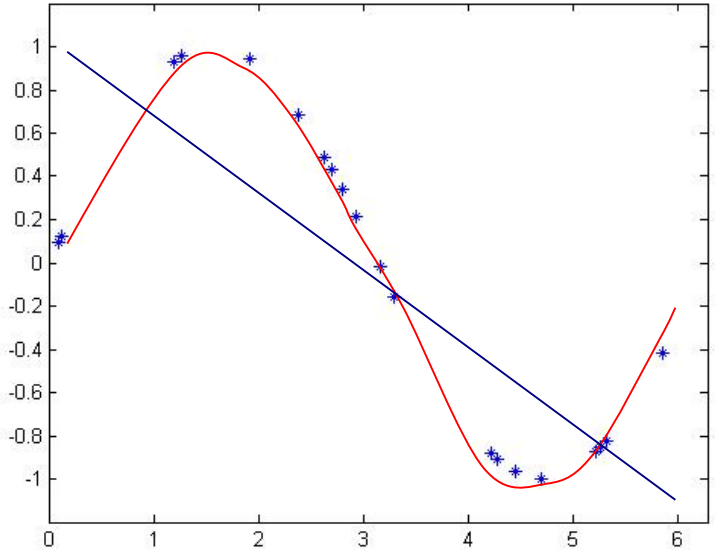
Why can MLP solve nonlinearly separable problem?

The nonlinearly separable problem can be transformed into linearly separable problem in the space produced by the hidden layer!

Single Layer Perceptron v.s. Multi-layer Perceptrons

Regression Problem:





Multi-layer Perceptrons can approximate any bounded continuous functions!

The learning algorithms are based upon the steepest descent method:

$$w_{ij}(k+1) = w_{ij}(k) - \eta \frac{\partial E(w)}{\partial w_{ij}}$$

$$w(n+1) = w(n) + \eta e(n) x(n)$$

Output Error

Input Signal

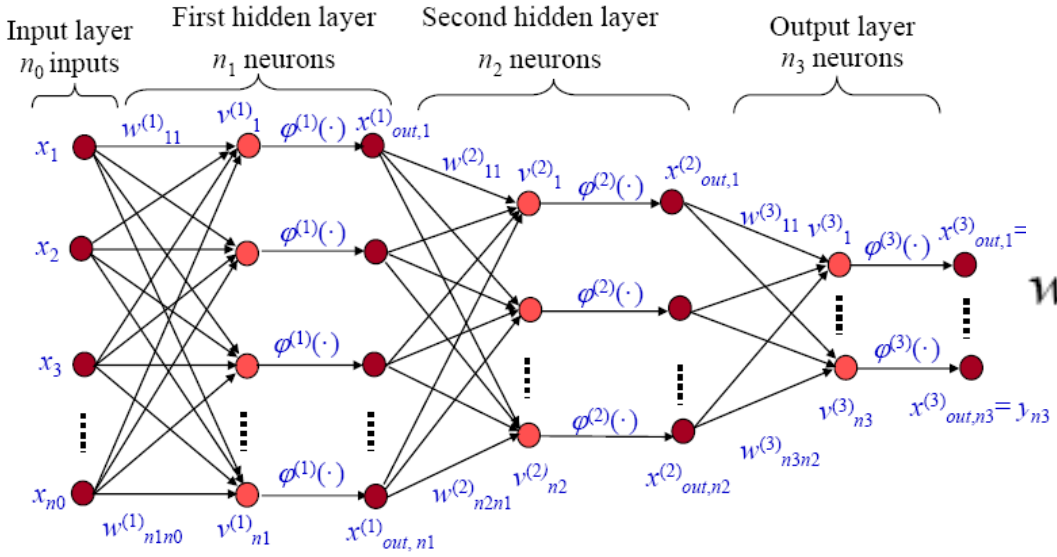
$$w_{ji}^{(s)}(k+1) = w_{ji}^{(s)}(k) + \eta^{(s)} \delta_j^{(s)} x_{out,i}^{(s-1)}$$

$\Delta w_{ji}^{(s)}$

Output Error

Input Signal

The learning of the Multi-layer perceptron

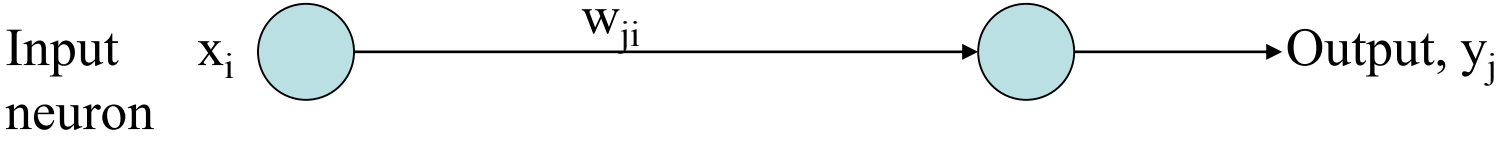


$$w_{ji}^{(s)}(k+1) = w_{ji}^{(s)}(k) + \eta^{(s)}\delta_j^{(s)}x_{out,i}^{(s-1)}$$

$\Delta w_{ji}^{(s)}$

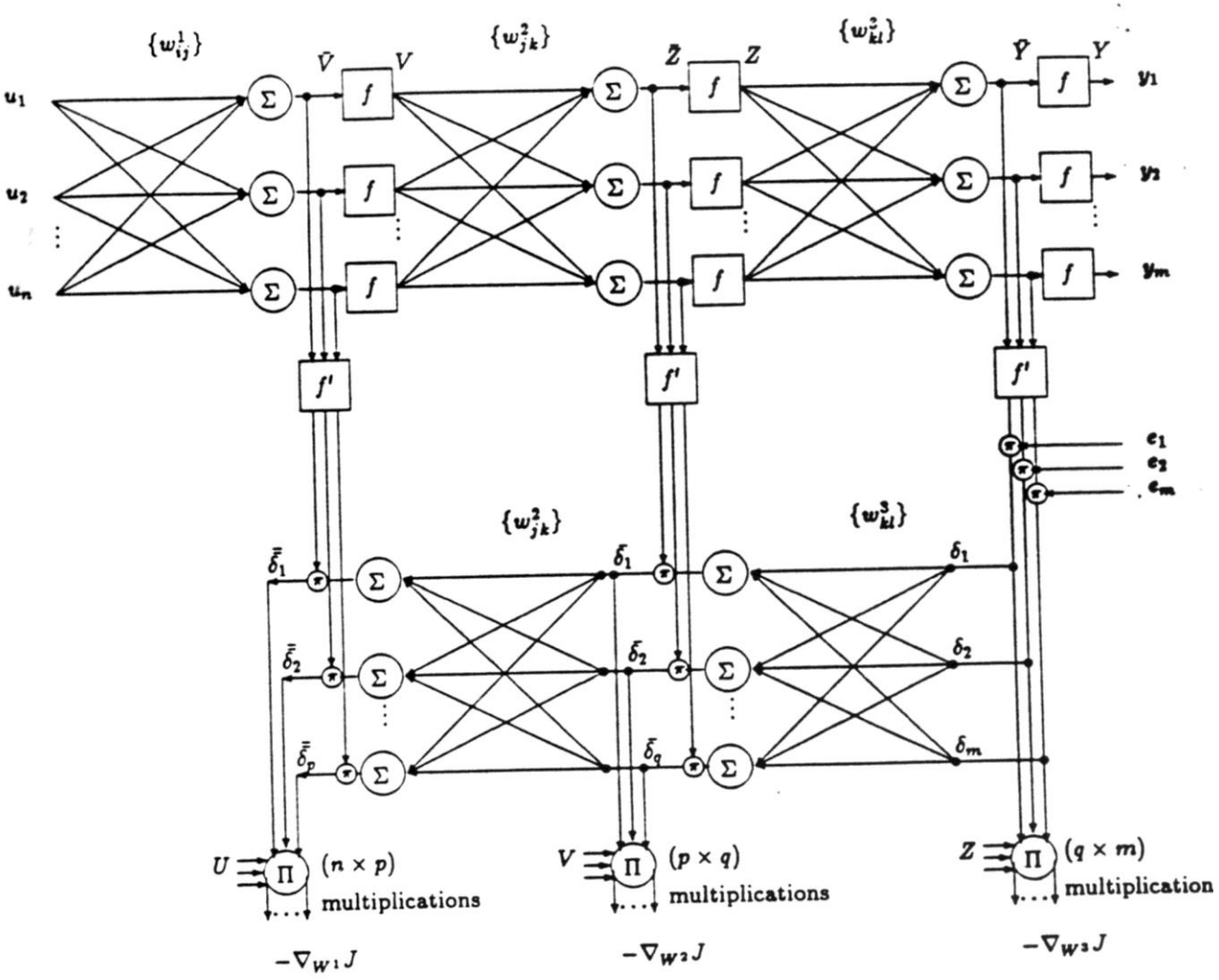
Output Error

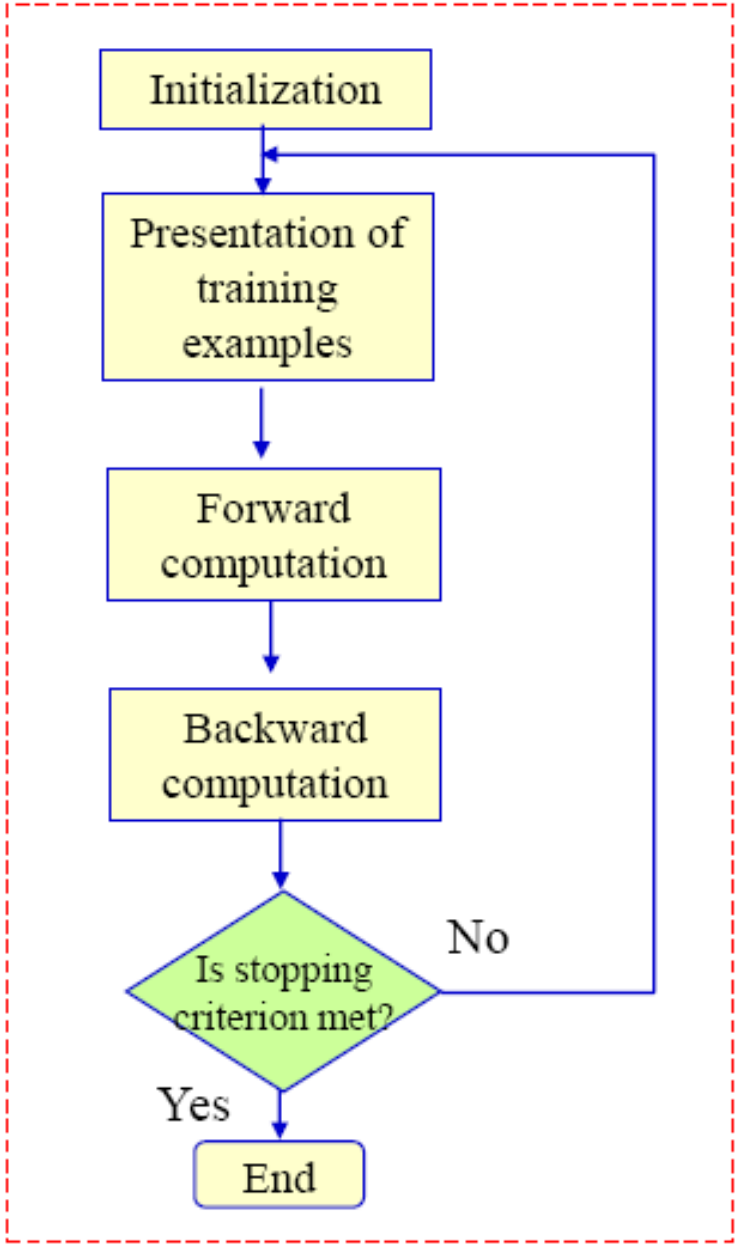
Input Signal



The adjustment of the synaptic weight only depends upon the information of the input neuron and the output neuron, and nothing else.

Signal-flow graphic representation of BP





How to design and train the neural network?

How many hidden layers?

Normally one hidden layer is enough. Two hidden layers may be better if the target function can be clearly decomposed into sub-functions.

How many hidden neurons?

If the geometrical shape can be perceived, then use the minimal number of line segments (or hyper-planes) as the starting point.

For higher dimensional problem, start with a large network, then use SVD to determine the effective number of hidden neurons.

How to choose activation function in the hidden layers?

Hyperbolic tangent (tansig) is the preferred one in all the hidden neurons.

How to choose activation functions in the output neurons?

Logsig for pattern recogntion, purelin for regression problem.

How to pre-process the input data?

Normalize all the input variables to the same range such that the mean is close to zero.

When to use sequential leaning? And when to use batch learning ?

Batch learning with second order algorithm (such as trainlm in MATLAB) is usually faster, but prone to local minima. Sequential learning can produce better solution, in particular for large database with lots of redundant samples.

How to deal with over-fitting?

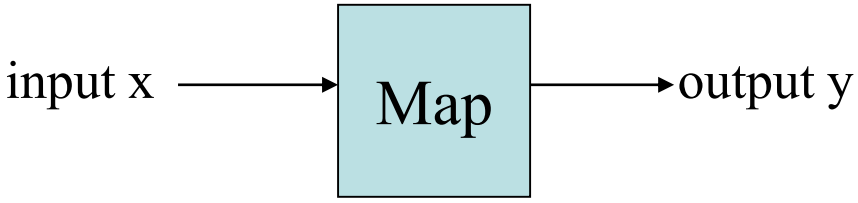
You either identify the minimal structure, or use regularization (trainbr in MATLAB).

What can MLP do?

MLP can solve regression and pattern recognition problems.

They all have the following characteristic:

There exists a function (map) between inputs x and outputs y: y=F(x)



Unfortunately, the mathematical form of this function (map) is unknown!

How to find out this map? Use first principles (physical and chemical laws) to build the model!

But sometimes the model is just too difficult to build.

Or the map is too complicated to be expressed by any known simple functions.

Then we will try to approximate this function instead. **How to approximate a function?**

MLP is an universal approximator! It can approximate any bounded function!

All it needs is a training set. Given a set of observations of input-output data: {(x(1),d(1)),(x(2),d(2)), (x(3),d(3)),…,(x(N), d(N))}

Use this training data to train the MLP such that the difference between the desired outputs and the outputs of the MLP are minimized.

Can we formulate the pattern recognition problem as function approximation? Yes.

Can MLP approximate any bounded function?

Yes. Does it mean that it can do anything? That was the dream of Rosenblatt!

What are the fundamental limits of MLP?

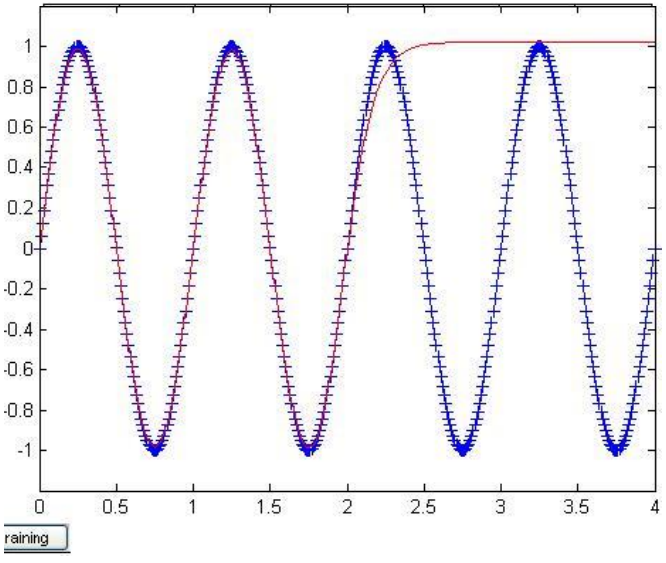
The net is essentially a black box: The MLP is just an approximation of the map.

No matter how good the approximation is, it is NOT the "true" map!

It thus cannot provide an intuitive (e.g., causal) explanation for the computed result.

There is also another fundamental limit:

Consider that the unknown map is a sinusoid: $y = \sin(2\pi x), x \in [0,2]$

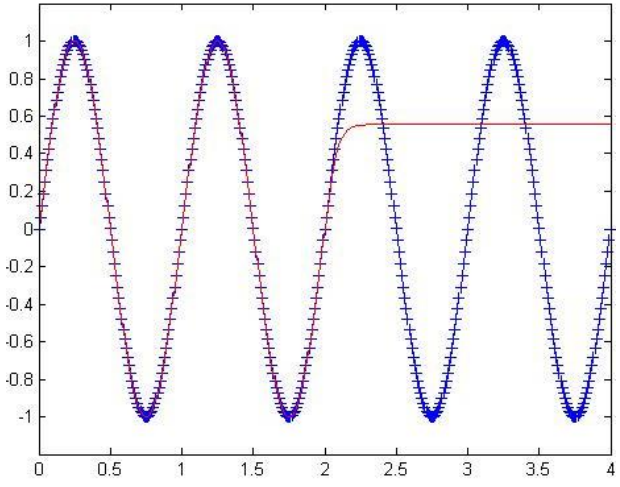


How many hidden neurons?

1-5-1

What do you expect the output for the input in the domain of [2,4]?

5 hidden neurons may not be enough, how about 100 neurons?



1-100-1 will also work with trainbr! But it still does not work well outside the domain of the training data.

The MLP is not smart enough to understand that the function is periodic!

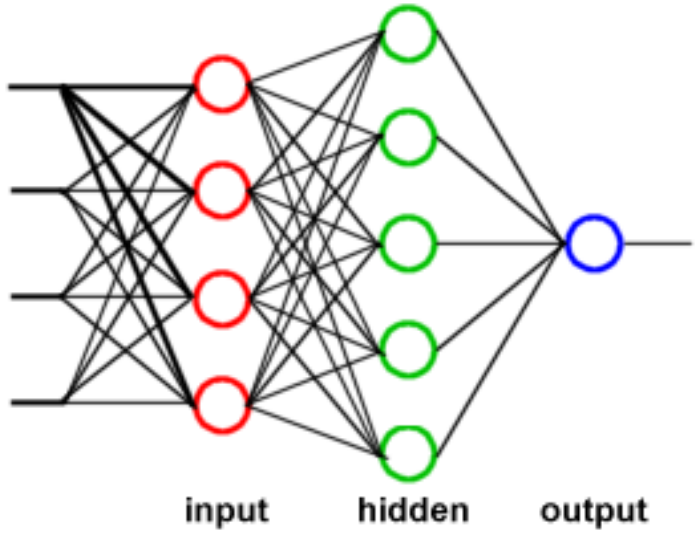
So MLP can learn, but can it really understand and create something new?

What are the other ways to approximate functions?

Fourier Series, polynomials, etc.

Today we are going to introduce another way.

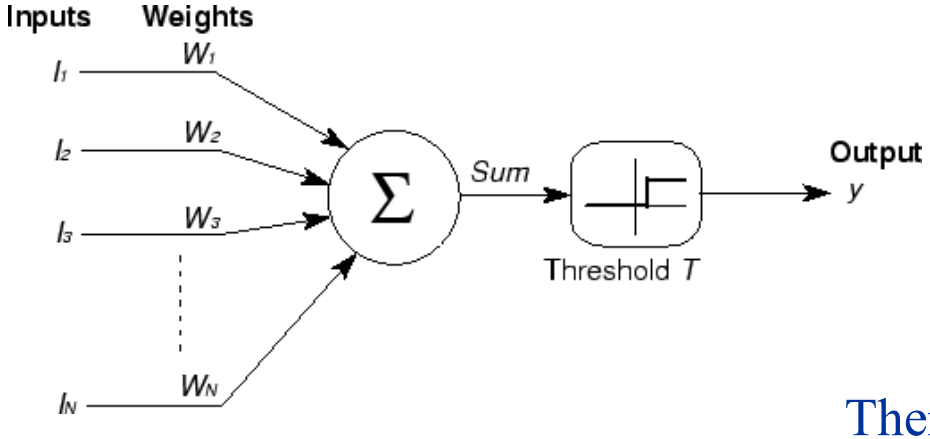
The idea of *Radial Basis Function (RBF) Networks* derives from the theory of function approximation.
RBF: a function which depends only on the *radial* distance from the input to a given center.

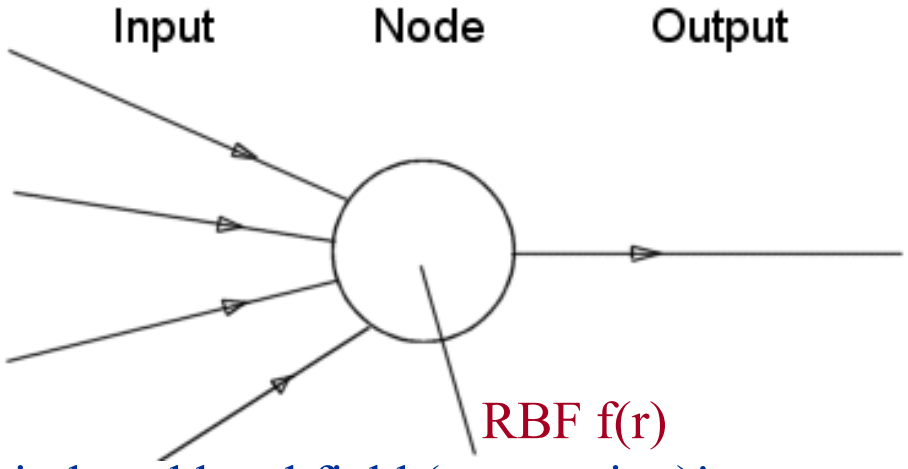


RBFN can also be represented by signal-flow graph.

Radial unit

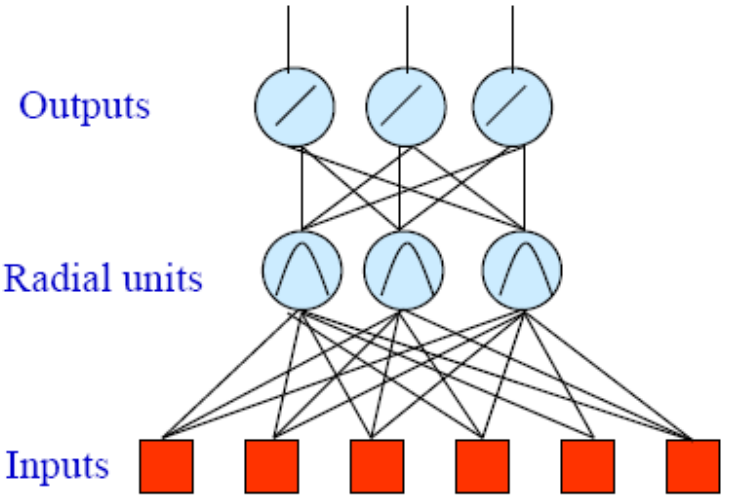
McCulloch and Pitts model





There is no induced local field (summation)!
There are no synaptic weights in the hidden layer.

Radial Basis Functions (RBFs)



The activation of a hidden unit is determined by the distance between the input vector and a prototype vector, the center.

$$\varphi(x) = \varphi(\| x - c \|) = \varphi(r)$$

Are the outputs the same for all the points on the circle?

Yes.

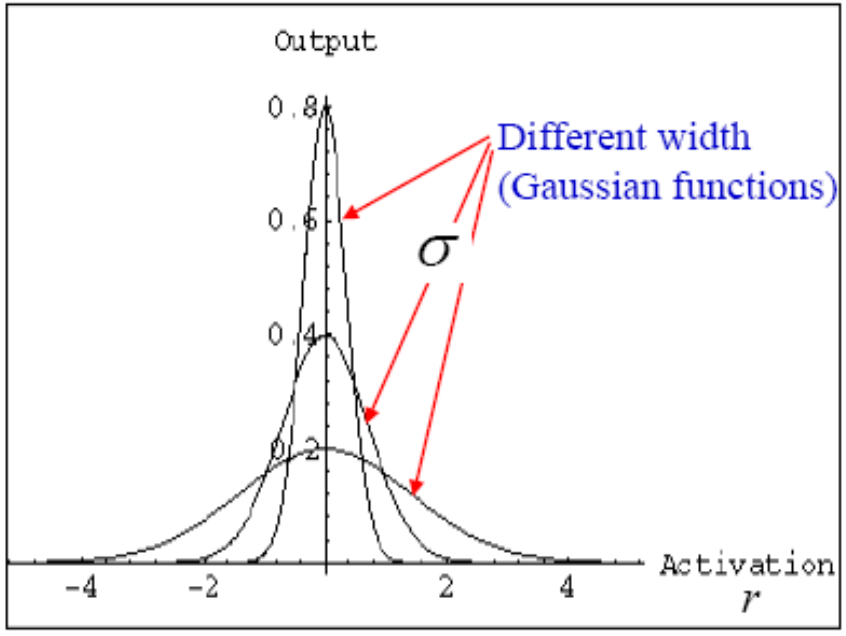
What is the simplest radial basis function you can think of?

$$\varphi(r) = r$$

When r=0, it means that the input is a perfect match of the center. We may want the neuron to respond to this special case instead of "nothing".

In most cases, if the distance is zero, the activation would be maximum.

The activation level will drop off further away from the center.



The study of radial basis functions was motivated by solving the problem of

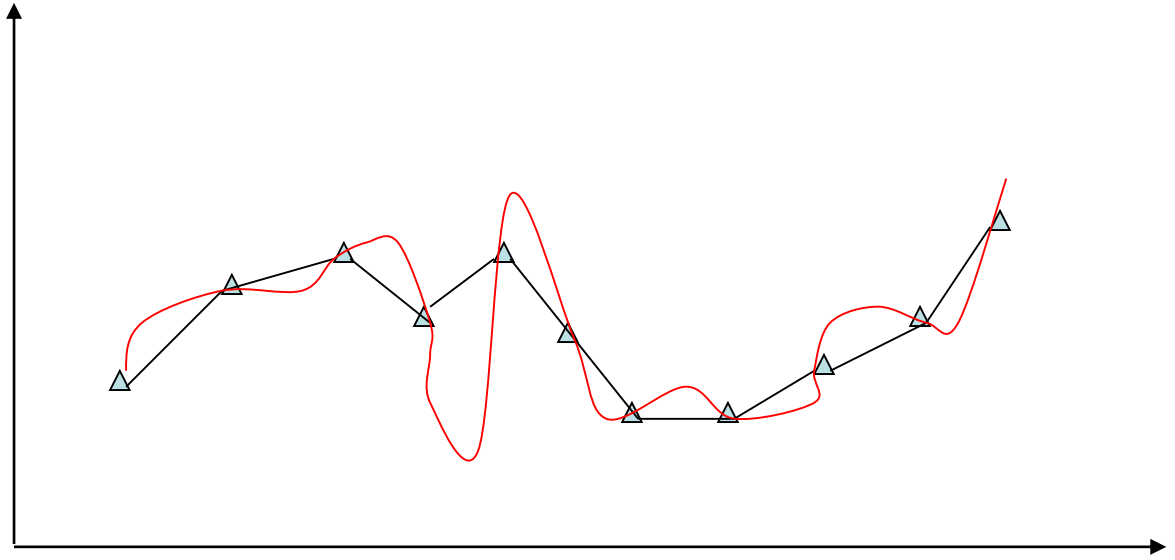
Exact Interpolation (perfect fitting)

Given a set of N data points $\{x_i \in R^{m_0}, i = 1,2,\cdots N\}$

and a set of N target values $\{d_i \in R, i = 1,2,\cdots N\}$

The goal is to find a function $f(\boldsymbol{x})$ that passes through all the training data points,

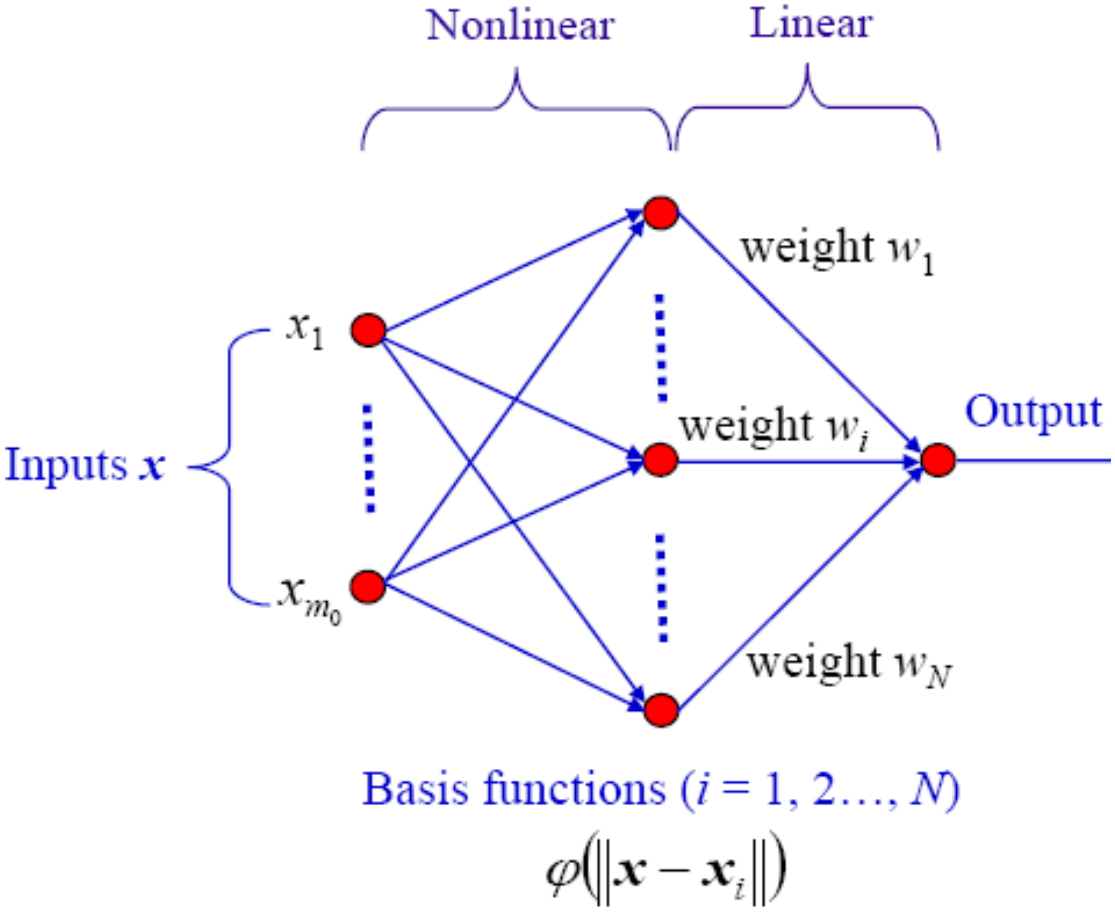
$$f(\boldsymbol{x}_j) = d_j \quad \forall\ j = 1, 2, ..., N$$



How many solutions can you find?

Infinite number of solutions. RBFN is just one solution!

Structure of a RBF network (Exact Interpolation)



$$f(x) = \sum_{i=1}^{N} w_i \varphi(\| x - x_i \|)$$

How many hidden neurons?

Number of hidden units = number of data points.

Where to place the centers?

The data points x_i are the centers of the RBF.

Form of the basis functions are chosen in advance.

How to Determine the Weights?

It is very simple. Just calculate the outputs for each sampling points:

$$f(\boldsymbol{x}_j) = \sum_{i=1}^{N} w_i \varphi\left(\left\|\boldsymbol{x}_j - \boldsymbol{x}_i\right\|\right) = d_j \qquad j = 1, 2, \ldots, N$$

$$\Rightarrow \begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1N} \\ \varphi_{21} & \varphi_{22} & \cdots & \varphi_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_{N1} & \varphi_{N2} & \cdots & \varphi_{NN} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}$$

interpolation matrix weight

where $\varphi_{ji} = \varphi\left(\left\|\boldsymbol{x}_j - \boldsymbol{x}_i\right\|\right), \quad (j,i) = 1,2,\ldots,N$

How many equations? N

How many weights? N

The above equation could be written in matrix form by defining the vectors:

$\boldsymbol{d} = [d_1, d_2, \ldots, d_N]^T$ and $\boldsymbol{w} = [w_1, w_2, \ldots, w_N]^T$, and the matrix $\boldsymbol{\Phi} = \{\varphi_{ji} \mid (j,i) = 1,2,\ldots,N\}$

This simplifies the equation to $\boldsymbol{\Phi w} = \boldsymbol{d}$

What is the condition for existence of unique solution of w? $\boldsymbol{\Phi}$ is nonsingular.

Then provided the inverse of $\boldsymbol{\Phi}$ exists, we can use any standard matrix inversion techniques to give $\boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{d}$

Once we have the weights, we have a function $f(\boldsymbol{x})$ that represents a continuous differentiable surface that passes exactly through each data point.

What is the condition to assure that $\boldsymbol{\Phi}$ is nonsingular?

Micchelli's Theorem (1986):

no matter what values of N or m_0

Let $\{\boldsymbol{x}_i\}_{i=1}^{N}$ be a set of distinct points in R^{m_0}. Then

the N-by-N interpolation matrix $\boldsymbol{\Phi}$, whose ji^{th} element is $\varphi_{ji} = \varphi(\|\boldsymbol{x}_j - \boldsymbol{x}_i\|)$, is nonsingular.

Of course, this theorem only holds for certain classes of radial basis functions.

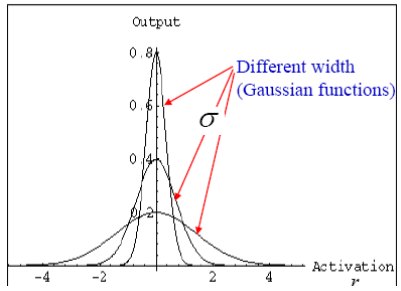
What are the radial basis functions for this theorem to hold?

NUS National University of Singapore

Commonly Used Radial Basis Functions

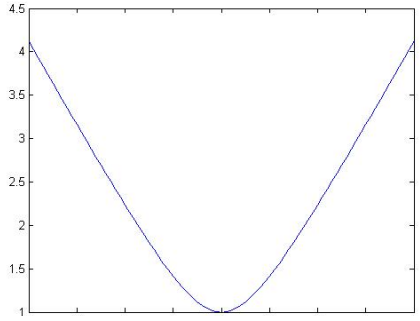
Gaussian Functions:

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \text{width parameter } \sigma > 0$$



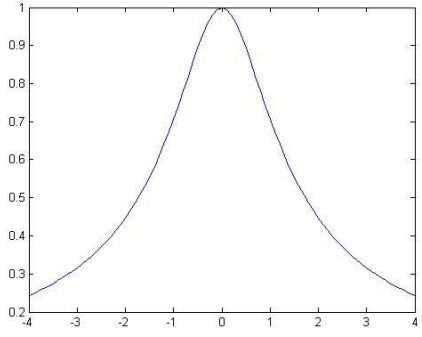
Multi-Quadric Functions:

$$\varphi(r) = \left(r^2 + \sigma^2\right)^{0.5} \quad \text{parameter } \sigma > 0$$



Inverse Multi-Quadric Functions:

$$\varphi(r) = \left(r^2 + \sigma^2\right)^{-0.5} \quad \text{parameter } \sigma > 0$$



The Gaussian and Inverse Multi-Quadric Functions are localized functions in the sense that $\varphi(r)$ is maximum for r=0, and $\varphi(r) \rightarrow 0$ as $|r| \rightarrow \infty$

By contrast, the Multi-Quadric Functions are *nonlocal* in that $\varphi(r)$ becomes unbounded as $|r| \rightarrow \infty$.

NUS National University of Singapore

Exact Interpolation

To summarize:

1. For a given data set containing N points $(\boldsymbol{x}_i, d_i)$, $i = 1, \ldots, N$
2. Choose a RBF function φ
3. Calculate $\varphi_{ji} = \varphi\left(\left\|\boldsymbol{x}_j - \boldsymbol{x}_i\right\|\right), \quad (j,i) = 1, 2, \ldots, N$
4. Obtain the interpolation matrix $\boldsymbol{\Phi}$
5. Solve the linear equation $\boldsymbol{\Phi}\,\boldsymbol{w} = \boldsymbol{d}$
6. Get the unique solution $\boldsymbol{w}$
7. Done

Example: XOR problem

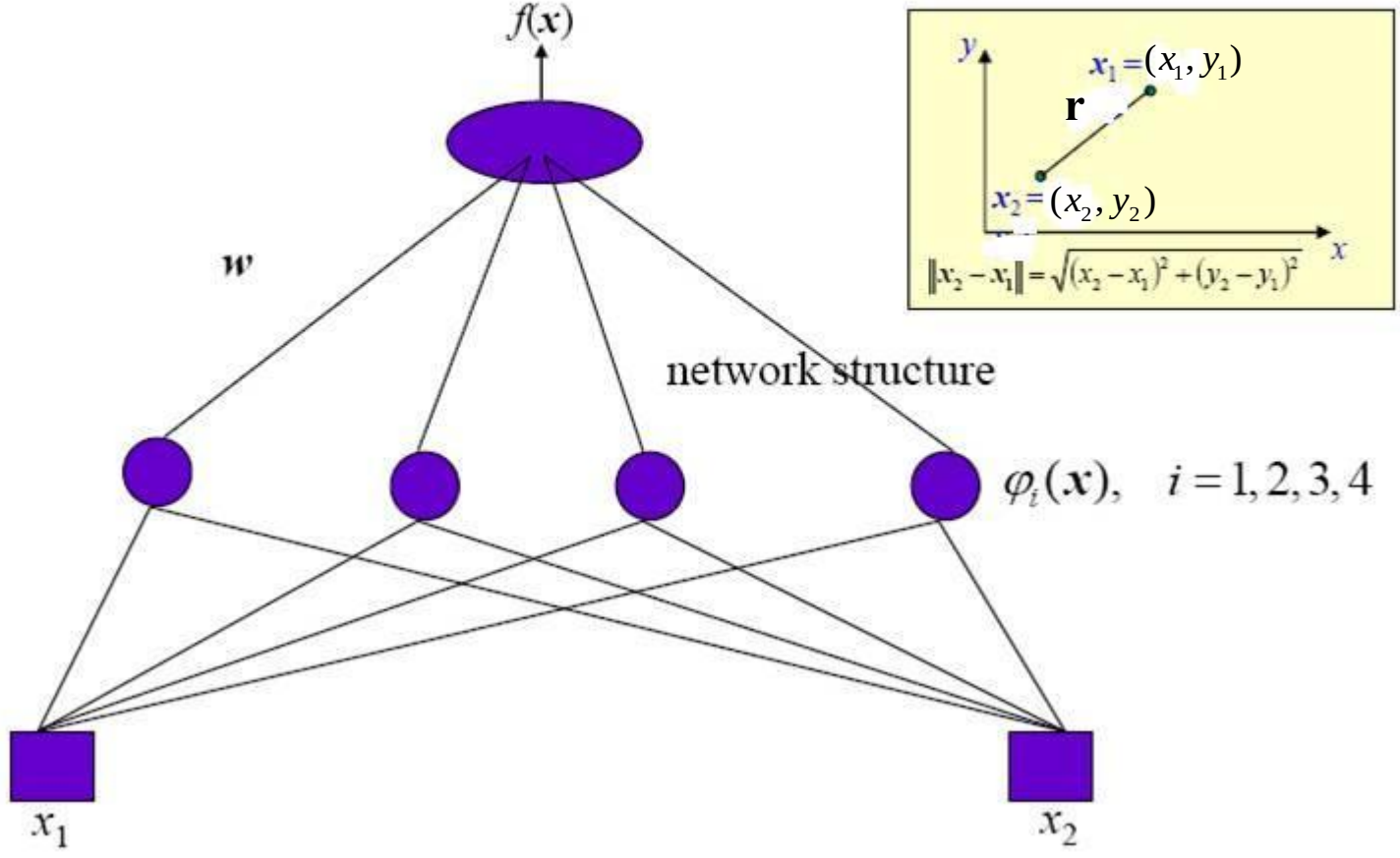
x_i	d_i
(0,0)	0
(1,0)	1
(0,1)	1
(1,1)	0

Exact interpolation: How many hidden units?

4 hidden units in the network, with centers as x_i i=1,…4.

(1) $\varphi_i(\boldsymbol{x}) = \| \boldsymbol{x} - \boldsymbol{x}_i \|$; (2) $\varphi_i(\boldsymbol{x}) = \exp\left(-\frac{\| \boldsymbol{x} - \boldsymbol{x}_i \|^2}{2}\right)$

linear RBF Gaussian RBF with $\sigma = 1$



$$\varphi_{ji}(\boldsymbol{x}_j) = \| \boldsymbol{x}_j - \boldsymbol{x}_i \|$$

$$\underset{\boldsymbol{\Phi}}{\begin{pmatrix} \varphi_{11} & \varphi_{12} & \varphi_{13} & \varphi_{14} \\ 0 & 1 & 1 & \sqrt{2} \\ \varphi_{21} & \varphi_{22} & \varphi_{23} & \varphi_{24} \\ 1 & 0 & \sqrt{2} & 1 \\ \varphi_{31} & \varphi_{32} & \varphi_{33} & \varphi_{34} \\ 1 & \sqrt{2} & 0 & 1 \\ \varphi_{41} & \varphi_{42} & \varphi_{43} & \varphi_{44} \\ \sqrt{2} & 1 & 1 & 0 \end{pmatrix}} \underset{\boldsymbol{w}}{\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix}} = \underset{\boldsymbol{d}}{\begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}}$$

x_i	d_i
(0,0)	0
(1,0)	1
(0,1)	1
(1,1)	0

$$\boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{d}$$

Results

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 1 \\ -\frac{\sqrt{2}}{2} \\ -\frac{\sqrt{2}}{2} \\ 1 \end{pmatrix}$$

check:

$$d_1 = -\frac{\sqrt{2}}{2} - \frac{\sqrt{2}}{2} + \sqrt{2} = 0$$

$$d_2 = 1 - \frac{\sqrt{2}}{2} \cdot \sqrt{2} + 1 = 1$$

etc.

$$\varphi_i(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}{2}\right)$$

$$\begin{pmatrix} \exp(0) & \exp(-.5) & \exp(-.5) & \exp(-1) \\ \exp(-.5) & \exp(0) & \exp(-1) & \exp(-.5) \\ \exp(-.5) & \exp(-1) & \exp(0) & \exp(-.5) \\ \exp(-1) & \exp(-.5) & \exp(-.5) & \exp(0) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

x_i	d_i
(0,0)	0
(1,0)	1
(0,1)	1
(1,1)	0

$$\boldsymbol{w} = \boldsymbol{\Phi}^{-1}\boldsymbol{d}$$

Results

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} -3.0359 \\ 3.4233 \\ 3.4233 \\ -3.0359 \end{pmatrix}$$

check:

$$d_1 = -3.0359 + 3.4233\exp(-.5) + 3.4233\exp(-.5) - 3.0359\exp(-1) = 0$$

etc.

The solutions to XOR problem

(1) $f(x_1, x_2) = \text{sqrt}(x_1^2+x_2^2)$
$- \frac{\sqrt{2}}{2} \text{sqrt}((x_1-1)^2+x_2^2)$
$- \frac{\sqrt{2}}{2} \text{sqrt}(x_1^2+(x_2-1)^2)$
$+ \text{sqrt}((x_1-1)^2+(x_2-1)^2)$

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} w_i \varphi(\|\boldsymbol{x} - \boldsymbol{x}_i\|)$$

$$\varphi_i(\boldsymbol{x}) = \| \boldsymbol{x} - \boldsymbol{x}_i \|$$

$$\|\boldsymbol{x} - \boldsymbol{x}_i\| = \sqrt{(x_1 - x_1(i))^2 + (x_2 - x_2(i))^2}$$

x_i	d_i
(0,0)	0
(1,0)	1
(0,1)	1
(1,1)	0

(2) $f(x_1,x_2) = -3.0359 \exp(-(x_1^2 + x_2^2)/2)$
$+ 3.4233 \exp(-(x_1-1)^2 + x_2^2)/2)$
$+ 3.4233 \exp(-(x_1^2+(x_2-1)^2)/2)$
$-3.0359 \exp(-(x_1-1)^2+(x_2-1)^2)/2)$

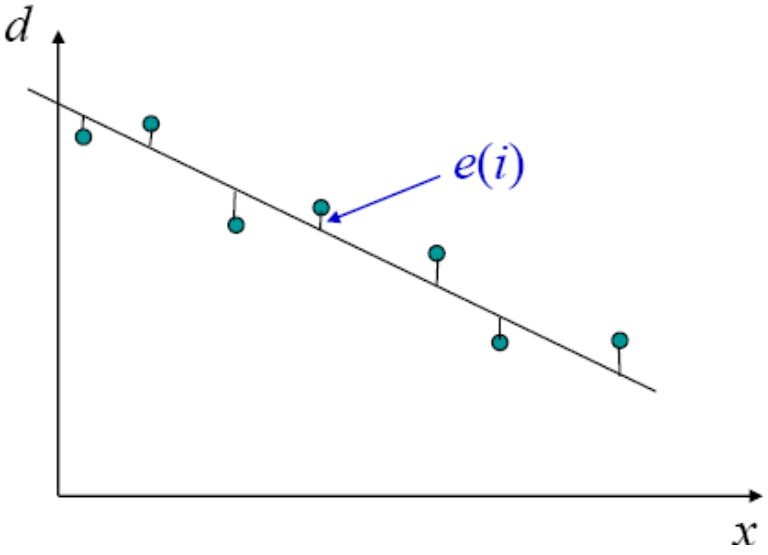
$$\varphi_i(\boldsymbol{x}) = \exp\left(-\frac{\| \boldsymbol{x} - \boldsymbol{x}_i \|^2}{2}\right)$$

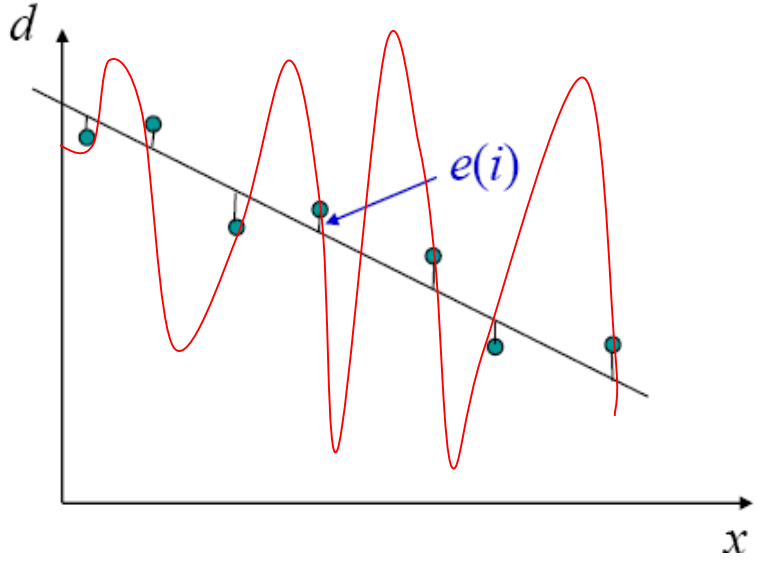
$$\|\boldsymbol{x} - \boldsymbol{x}_i\|^2 = (x_1 - x_1(i))^2 + (x_2 - x_2(i))^2$$

We have seen how we can set up RBF networks that perform exact interpolation, but there are problems with these exact interpolation networks.

Do you really want to fit every sample? What are the obvious problems?

Over-fitting!



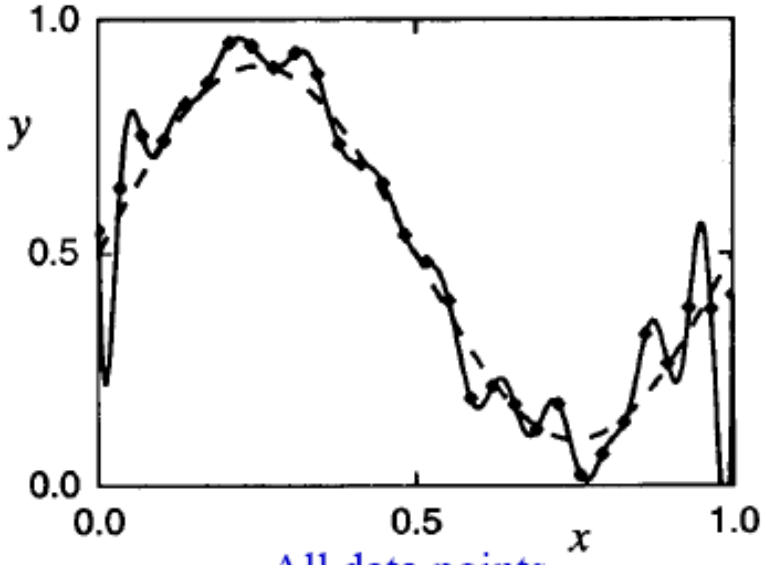


1. They perform poorly with noisy data:

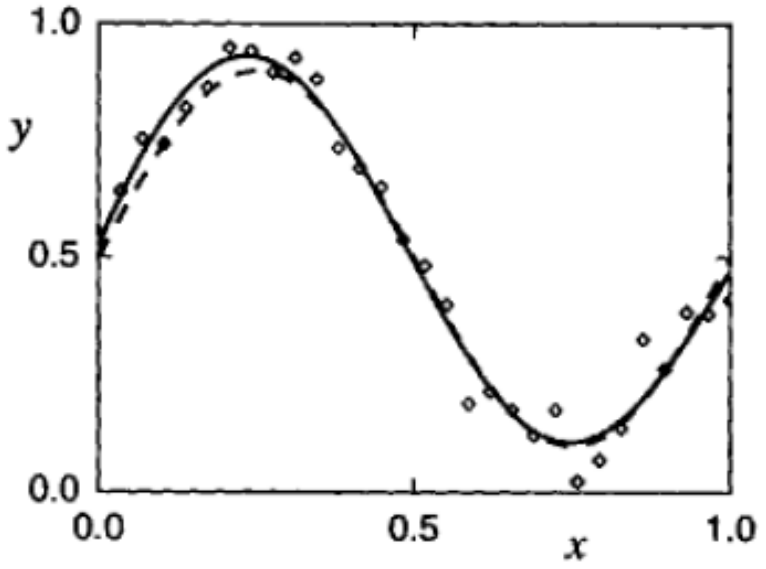
As we have seen for our multi-layer perceptrons, we do not generally want the network's outputs to pass through all the data points when the data are noisy, because that will generally be a highly oscillatory function that will not provide good generalization.

- Bishop (1995) example: Underlying function $f(x) = 0.5 + 0.4\sin(2\pi x)$ sampled randomly for 30 points and added Gaussian noise to each data

How many hidden units are needed for exact interpolation? 30



All data points
(30 Basis functions)



5 Basis functions

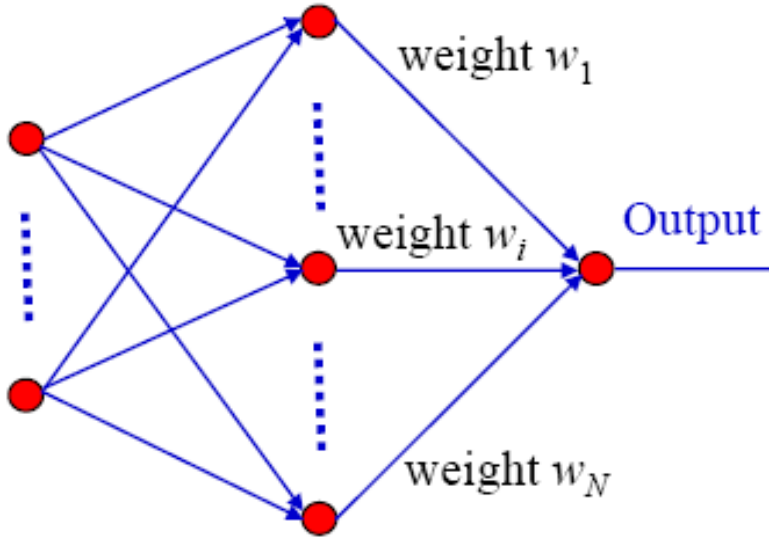
Imagine you are dealing with a large database with 1 million samples, how many hidden units do you need for exact interpolation?

1 Million!

The network requires one hidden unit (i.e. one basis function) for each training data pattern, and so for large data sets the network will become very costly to evaluate.

$$f(x) = \sum_{i=1}^{N} w_i \varphi(\| x - x_i \|)$$

$$\varphi(r) = \exp\left(-\frac{r^2}{2\sigma^2}\right) \quad \text{width parameter } \sigma > 0$$



How to improve the RBF networks?

Do we really want to fix the number of hidden units as the number of the training samples?

In general, it is better to have it much less than *N*.

How about the centers? Is it necessary to use the sampling points as the centers?

The centers do not need to match the sampling points.

How to determine the centers then? They can be learned by training as that for MLP!

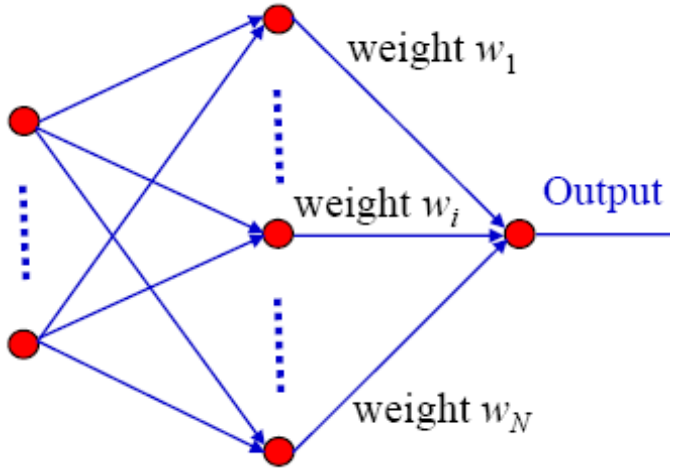
How about the spread parameter σ ? Is it efficient to keep them the same for all the units?

The spread parameters can be different for different units! And they can be learned by training too!

What else can we add to the network?

We can introduce *bias parameters* into the network

$$f(x) = \sum_{i=1}^{M} w_i \varphi_i(\| x - \mu_i \|) + b$$



When these changes are made to the exact interpolation formula, we arrive at an improved RBF network mapping,

$$y(x) = \sum_{i=1}^{M} w_i \varphi_i(\| x - \mu_i \|) + w_0$$

We can simplify it by introducing an extra basis function $\varphi_0 = 1$ to give

$$y(x) = \sum_{i=0}^{M} w_i \varphi_i(\| x - \mu_i \|)$$

For the case of Gaussian basis functions we have,

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

in which we have basis centers $\{\mu_i\}$ and standard deviations or widths $\{\sigma_i\}$

The aim is to develop a process for finding the appropriate values of the weights w_i and the centers and spreads $\{\mu_i\}$, and $\{\sigma_i\}$ for function approximation.

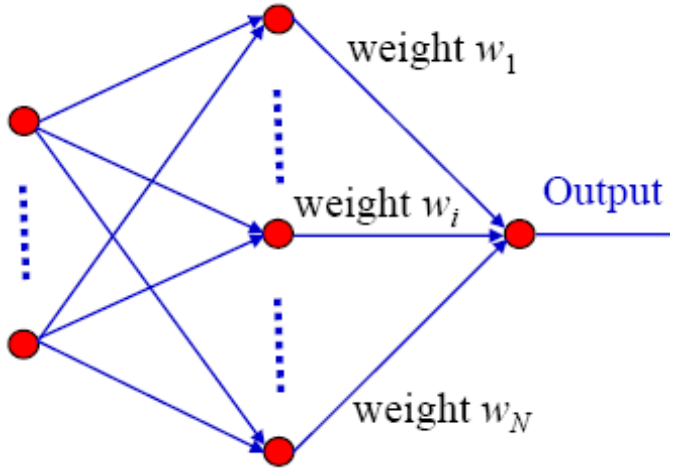
NUS National University of Singapore

Break

- Rise of the machines

$$y(x) = \sum_{i=1}^{M} w_i \varphi_i(\| x - \mu_i \|) + b$$

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



We know that MLP can approximate any bounded continuous function. Can RBFN also do this?

Universal Approximation Theorem (Park and Sandberg, 1991)

RBFN can approximate any bounded continuous function!

It is just an existence result! It does not tell you how to obtain the weights.

Given a set of sampling points and desired outputs {(x(i), d(i)), i=1,…,N}

How to find out the parameters $\{w_i\}$, $\{\mu_i\}$ and $\{\sigma_i\}$ such that the cost function

$$E(w) = \sum_{i=1}^{N} e(i)^2 = \sum_{i=1}^{N} (d(i) - y(i))^2$$

is minimized?

What is the simplest way to solve this optimization problem?

Steepest Descent Method

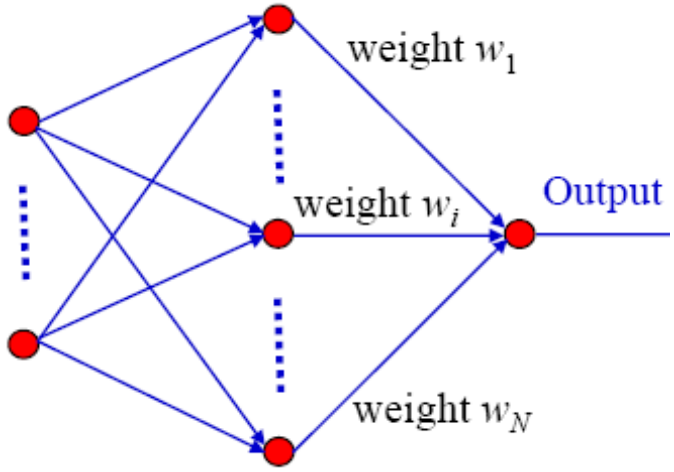
$$w_{ij}(n+1) = w_{ij}(n) - \eta \frac{\partial E(w)}{\partial w_{ij}}$$

Can we also derive Backpropagation learning algorithm for RBFN in this way?

We can certainly use steepest descent, but the algorithm will not have the property of "BP"!

$$y(x) = \sum_{i=1}^{M} w_i \varphi_i(\| x - \mu_i \|) + b$$

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



In MLP, the synaptic weights play similar roles in different layers. They are the weights for summation.

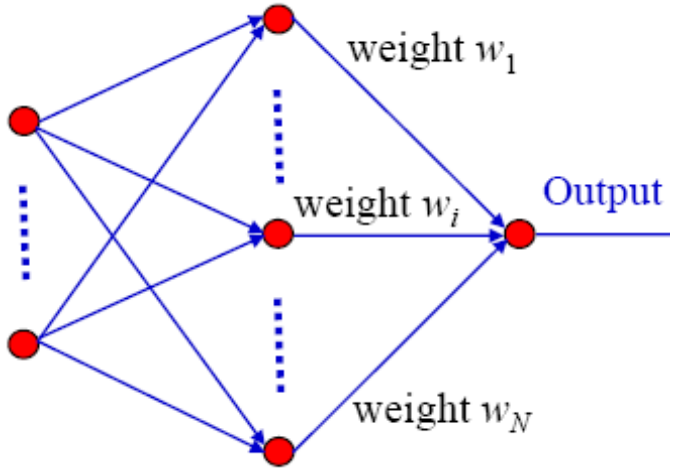
How about the parameters for RBFN? Do the "weights" in the hidden units, the centers and the spreads, play the similar role as the weights in the output layer?

The hidden and output layers play very different roles in RBF networks, and the corresponding "weights" have very different meanings and properties. It is therefore appropriate to use different learning algorithms for them.

The input to hidden "weights" (e.g., basis function parameters $\{\boldsymbol{\mu}_i\}$) can be trained/set using any of a number of unsupervised learning techniques.

After the input to hidden "weights" are found, they are kept fixed for the second stage of training during which the hidden to output weights are learned.

Hybrid training of RBF networks



Two stage 'hybrid' learning process:

(Stage 1) Parameterize hidden layer of RBFs.

- hidden unit number (M)
- centre/position (μ_i)
- spread/width (σ_i)

Use unsupervised learning methods as they are quick.

(Stage 2) Find weight values between hidden and output units.

Aim: Minimize sum-of-squares error between actual outputs and desired responses.

Use supervised learning methods.

Finding the Output Weights

Given the hidden unit activations $\varphi_i(\boldsymbol{x}, \boldsymbol{\mu}_i, \sigma_i)$ are fixed while we determine the output weights $\{w_i\}$, a simple way is to solve the set of linear equations:

$$y(\boldsymbol{x}_j) = \sum_{i=0}^{M} w_i \varphi_i(\boldsymbol{x}_j) = d_j$$

$\varphi_0 = 1$

$$\Rightarrow \underbrace{\begin{bmatrix} \varphi_0(\boldsymbol{x}_1) & \varphi_1(\boldsymbol{x}_1) & \cdots & \varphi_M(\boldsymbol{x}_1) \\ \varphi_0(\boldsymbol{x}_2) & \varphi_1(\boldsymbol{x}_2) & \cdots & \varphi_M(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_0(\boldsymbol{x}_N) & \varphi_1(\boldsymbol{x}_N) & \cdots & \varphi_M(\boldsymbol{x}_N) \end{bmatrix}}_{\boldsymbol{\Phi}} \underbrace{\begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_M \end{bmatrix}}_{\boldsymbol{w}} = \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_N \end{bmatrix}}_{\boldsymbol{d}}$$

How to solve equation $\boldsymbol{\Phi w} = \boldsymbol{d}$?

If Φ is a nonsingular square matrix, then it is simple.

$$w = \Phi^{-1} d$$

If Φ is a non-square matrix, and the number of samples N<M+1 (number of weights), then the number of equations is less than the number of weights. How many solutions?

There are infinite number of solutions!

Then you can choose the solution to meet other requirements, like minimizing the magnitudes.

If the number of samples N>M+1 (number of weights), then the number of equations is more than the number of weights. Does the solution always exist?

Not really.

What if the solution does not exist (most of the time)? How to determine the weights?

We need to choose weights to minimize the summation of squared errors:

$$E(w)=\sum_{i=1}^{N}e(i)^2=\sum_{i=1}^{N}(y(i)-d(i))^2=(\Phi w-d)^T(\Phi w-d)$$

What is the optimality condition? $\dfrac{\partial E(w)}{\partial w}=0$

Did we solve a similar problem before?

Yes. Because the function is linear in parameters, this is the same problem as standard linear least squares (pages 49-51 in lecture two).

$$w=(\Phi^T\Phi)^{-1}\Phi^T d$$

So you can get the weights in one step calculation! Is it simpler than MLP?

Yes.

Basis Functions Optimization

So we have seen that computing the weights in the output layer is a piece of cake once the basis functions in the hidden units are fixed. This is the main advantage of breaking down the learning of RBFN into two stages.

The second stage is straightforward. For the first stage, there are numerous approaches in the literature. We shall first look at one simple method of doing this by "fixed centres selected at random".

This is an unsupervised technique, which is considered to be a "sensible" approach, provided that the training data are distributed in a representative manner for the problem at hand.

Fixed Centres Selected at Random

The simplest and quickest approach to setting the RBF parameters is to take their centers as fixed at M points selected at random from the N data points, and to take their widths to be equal and fixed at an appropriate size for the distribution of data points.

Specifically, we can use RBFs centred at $\{\boldsymbol{\mu}_i\}$ defined by

$$\varphi_i(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2}{2\sigma_i^2}\right)$$

$$\varphi_i(\boldsymbol{x}) = \exp\left(-\frac{M}{d_{\max}^2}\|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2\right) \quad \text{where} \quad i = 1, 2, \ldots, M$$

where $d_{\max}$ is the maximum distance between the chosen centres. The spreads are are

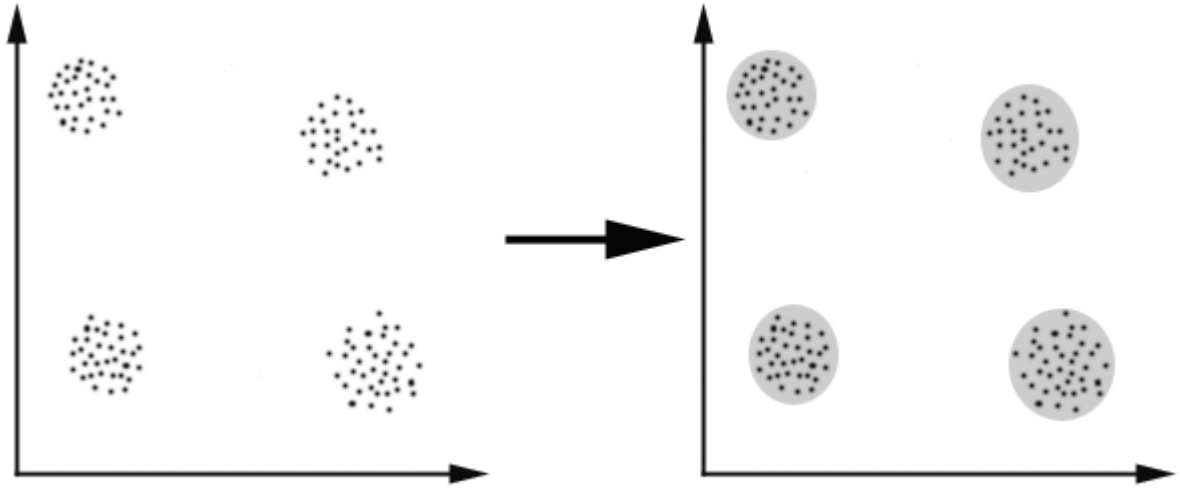
$$\sigma_i = \frac{d_{\max}}{\sqrt{2M}}$$

which ensures that individual RBFs are neither too peaked nor too flat. For large training sets, this simple approach gives reasonable results.

The main problem with the method of fixed centers is that it may require a large training set for satisfactory performance. The number of the hidden units is also large.

One way to overcome this limitation is to use clustering algorithm.

Clustering is the assignment of objects into groups (called *clusters*) so that objects from the same cluster are more similar to each other than objects from different clusters.

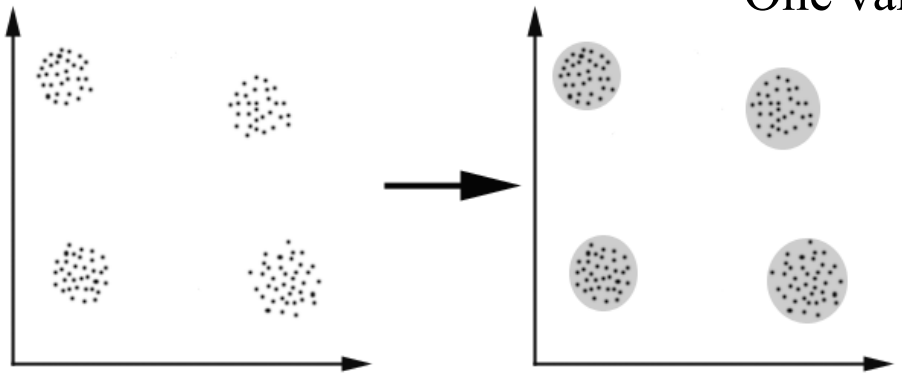


Clustering is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis and bioinformatics.

It is can be simply done visually in low dimensional space.

It is not trivial at all for high dimensional space, and it is still a hot research topic today!

Next we will introduce the classical k-means algorithm.



One variant of K-means clustering algorithm:

Assume the number of the centers for RBFN (also the number of clusters) is given as C.

$\mu_k(n), k = 1 \cdots C$ The centers at iteration n

1. *Initialization*. Choose random values for the initial centers $\mu_k(0)$

2. *Sampling*. Draw a sample vector x(n) from the training set.

3. *Assignment*. Compute the distance from the sample vector x(n) to each center, $\mu_k(n)$, choose the center which is closest to the sample and assign the vector to the cluster which is represented by that center.

4. *Updating*. After all the samples are assigned to various clusters, update the centers by the means (average values) of the samples for each cluster. Then we will get $\mu_k(n+1)$.

5. *Continuation*. Go back to step 2, and continue the procedure until convergence.

Is this supervised learning or unsupervised learning?

Unsupervised learning!

Supervised RBF Network Training

Supervised training of the basis function parameters will generally give better results than unsupervised procedures, but the computational costs are usually enormous.

The obvious approach is to perform gradient descent on a sum squared output error function as we did for MLPs. The error function could be,

$$E = \frac{1}{2}\sum_{j=1}^{N}\left(e_j\right)^2$$

$$\varphi_i(\cdot) = \exp\left(-\frac{\left\|\boldsymbol{x}_j - \boldsymbol{\mu}_i\right\|^2}{2\sigma_i^2}\right)$$

$$e_j = d_j - y(\boldsymbol{x}_j) = d_j - \sum_{i=0}^{M} w_i \varphi_i(\boldsymbol{x}_j, \boldsymbol{\mu}_i, \sigma_i)$$

and we would iteratively update the weights/basis function parameters using

$$\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n+1) - \boldsymbol{w}(n) = -\eta \cdot \boldsymbol{g}(n)$$

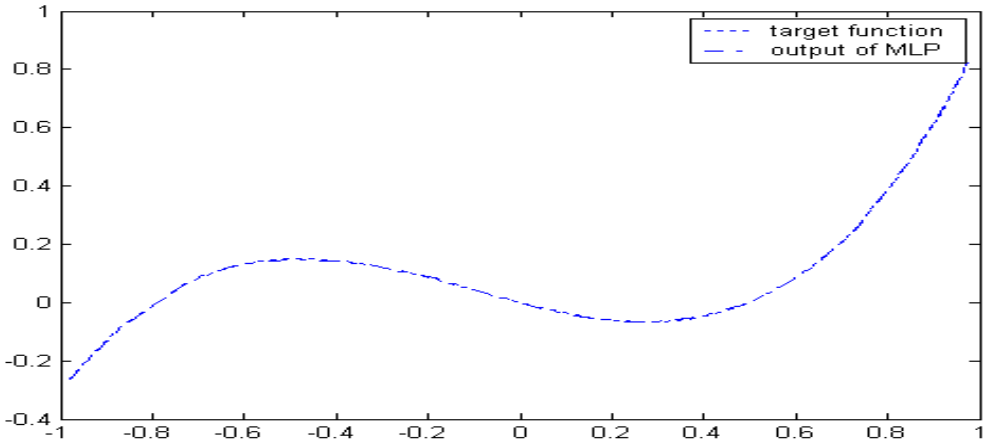
$$\Delta w_i = -\eta_w \frac{\partial E}{\partial w_i} \qquad \Delta \boldsymbol{\mu}_i = -\eta_\mu \frac{\partial E}{\partial \boldsymbol{\mu}_i} \qquad \Delta \sigma_i = -\eta_\sigma \frac{\partial E}{\partial \sigma_i}$$

We have all the problems of choosing the learning rates η, avoiding local minima and so on, that we had for training MLPs by gradient descent.

Second order methods can also be attempted.

Open Question:

How to choose the number of hidden neurons for RBFN?



For MLP, we already have a clear answer. But this still remains an open problem for RBFN.

Why is finding out minimal structure so important?

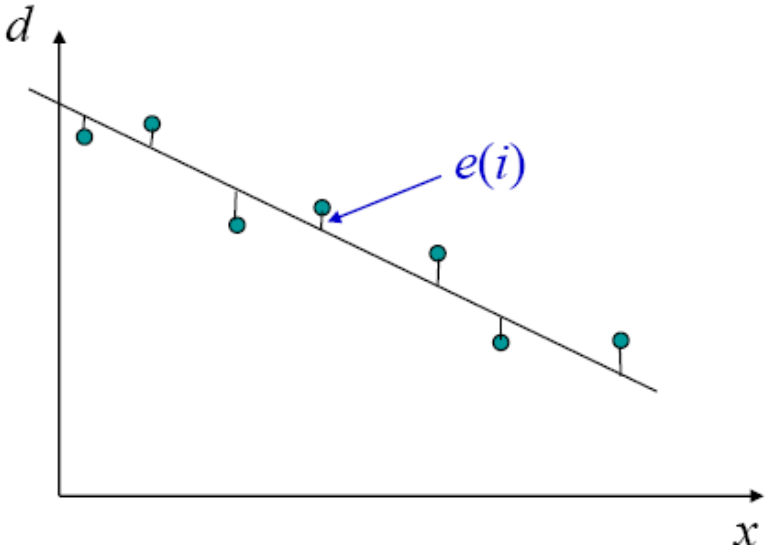
Limiting the size of the network is a good strategy for overcoming the problem of over-fitting.

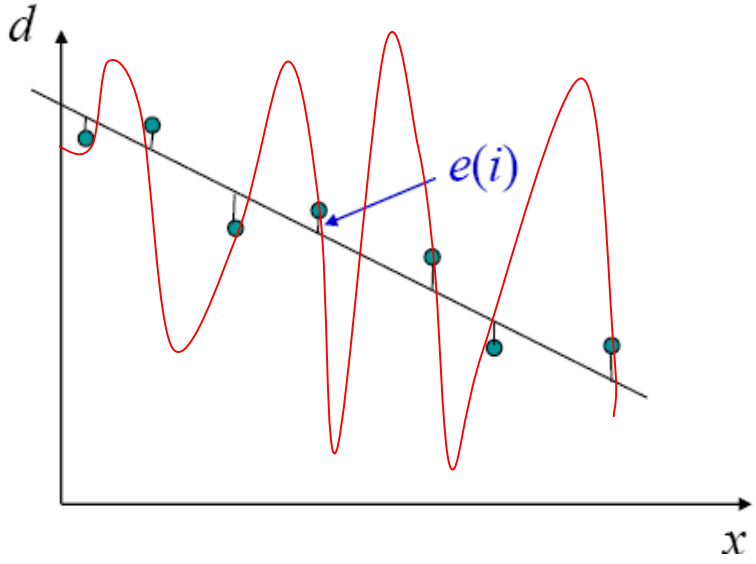
If it is not easy to determine the good size, then there is another strategy to deal with over-fitting. **What is that?**

Limit the size of the weights!-→Regularization technique!

Regularization Theory for RBF Networks

RBFN can also lead to to over-fitting (learns the noise present in the training data) and result in poor generalization.





An alternative approach to cope with over-fitting comes from the theory of regularization, which is a method of controlling the *smoothness* of mapping functions.

Regularization Methods

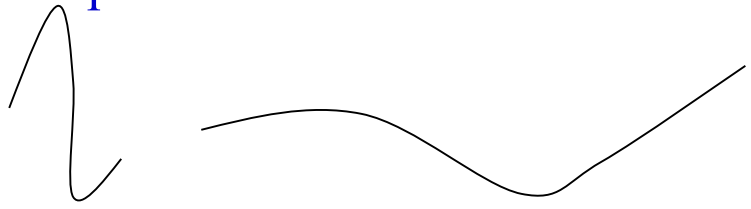
Cost function:

$$F = E_D + \lambda E_w$$

training error

cost on smoothness

It involves adding an extra term to the error measure which penalizes mappings that are not smooth.



How to measure the smoothness of a function?

The magnitudes of the derivatives (slopes)!

$$f(x) = \sum_{i=0}^{M} w_i \varphi(\| x - \mu_i \|) \quad \Longrightarrow \quad \frac{\partial f(x)}{\partial x} = \sum_{i=0}^{M} w_i \frac{\partial \varphi(\| x - \mu_i \|)}{\partial x}$$

How do the weights affect the smoothness?

The smaller the weights, the smaller the slope!

We need to minimize the size of the weights just as that for MLP!

A simple way to choose the penalty term is the size of the weights.

Cost function: $$F(w)=\frac{1}{2}\sum_{i=1}^{N}(y(i)-d(i))^2+\frac{1}{2}\lambda\|w\|^2=\frac{1}{2}(\Phi w-d)^T(\Phi w-d)+\frac{1}{2}\lambda w^T w$$

$$\begin{bmatrix} \varphi_0=1 & & & \\ \varphi_0(x_1) & \varphi_1(x_1) & \cdots & \varphi_M(x_1) \\ \varphi_0(x_2) & \varphi_1(x_2) & \cdots & \varphi_M(x_2) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_0(x_N) & \varphi_1(x_N) & \cdots & \varphi_M(x_N) \end{bmatrix} \leftarrow \Phi$$

How to minimize F(w)?

$$\frac{\partial F(w)}{\partial w}=0$$

In order to use the chain rule, let's define the error vector:

$$e=\Phi w-d$$

$$F(w)=\frac{1}{2}e^T e+\frac{1}{2}\lambda w^T w$$

$$\frac{\partial F(w)}{\partial w}=e^T\Phi+\lambda w^T=(\Phi w-d)^T\Phi+\lambda w^T$$

$$\frac{\partial F(w)}{\partial w}=w^T\Phi^T\Phi-d^T\Phi+\lambda w^T=0 \Longrightarrow w^T(\Phi^T\Phi+\lambda I)=d^T\Phi$$

$$(\Phi^T\Phi+\lambda I)w=\Phi^T d \Longrightarrow w=(\Phi^T\Phi+\lambda I)^{-1}\Phi^T d$$

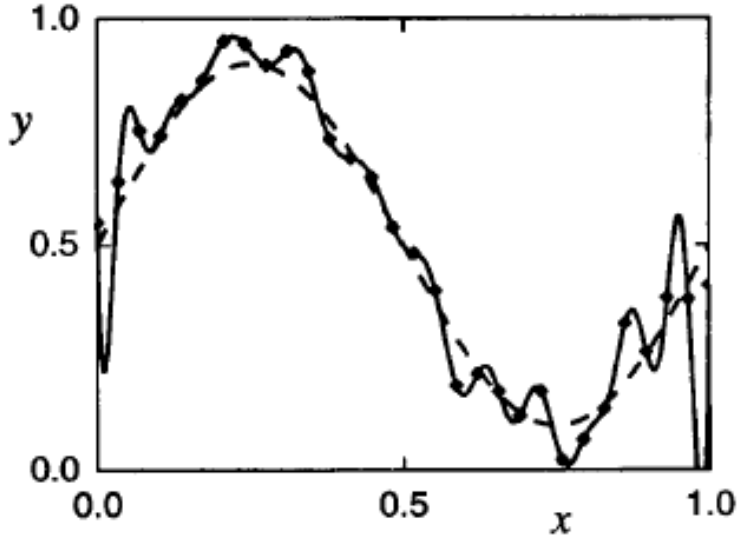
λ is the regularization factor

The regularization parameter (λ) plays an important role in the theory of regularized RBF networks.

λ = 0 unconstrained (smoothness not enforced).

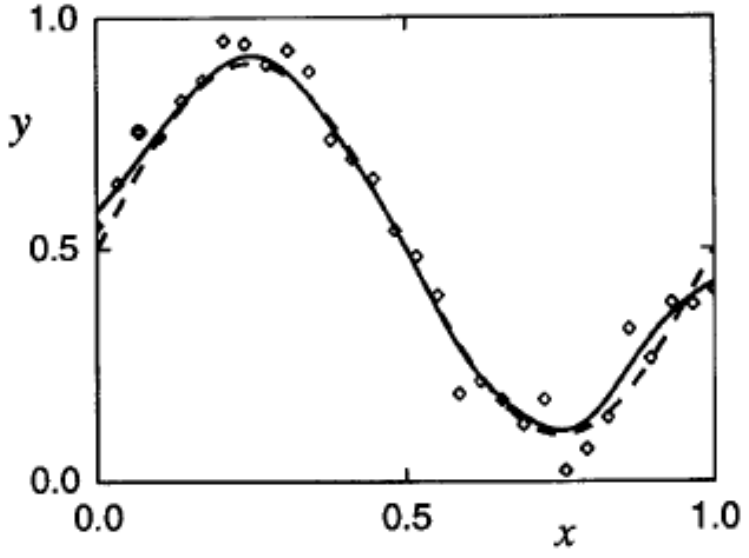
λ = infinity, smoothness constraint dominates and less account is taken for training data error.

λ controls the balance (trade-off) between a smooth mapping and fitting the data points exactly.



$\lambda = 0$

1-30-1

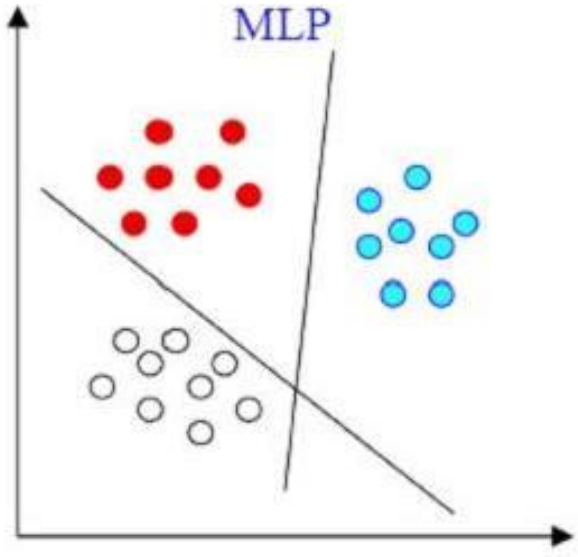


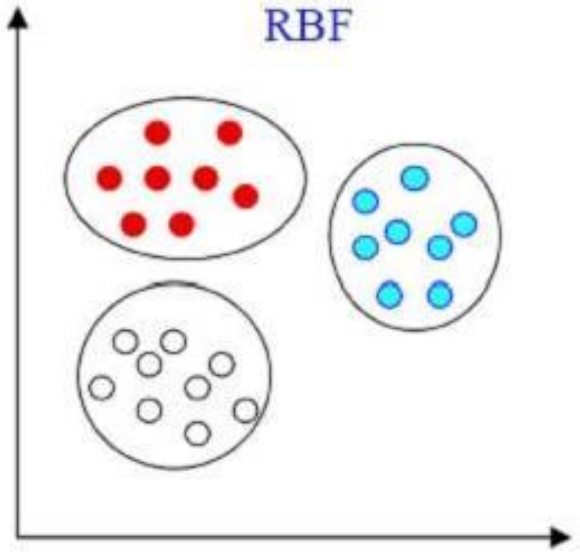
$\lambda = 40$

1-30-1

RBF Networks for Classification

So far we have concentrated on RBF networks for function approximation. They are also useful for classification problems. Suppose we have a data set that falls into three classes:





An MLP would separate the classes with hyper-planes in the input planes (left).

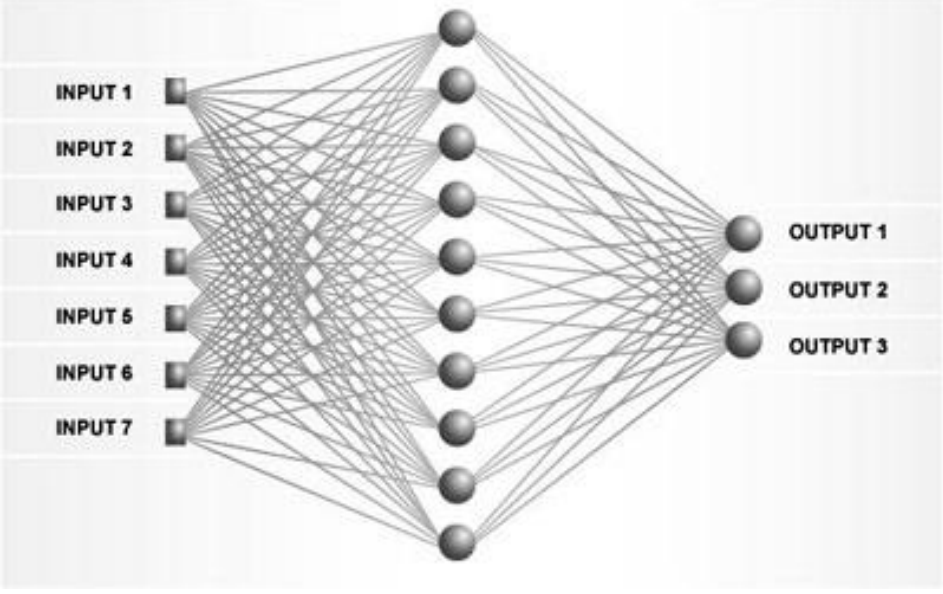
An alternative approach would be to model the separate class distributions by localized basis functions (as on the right).

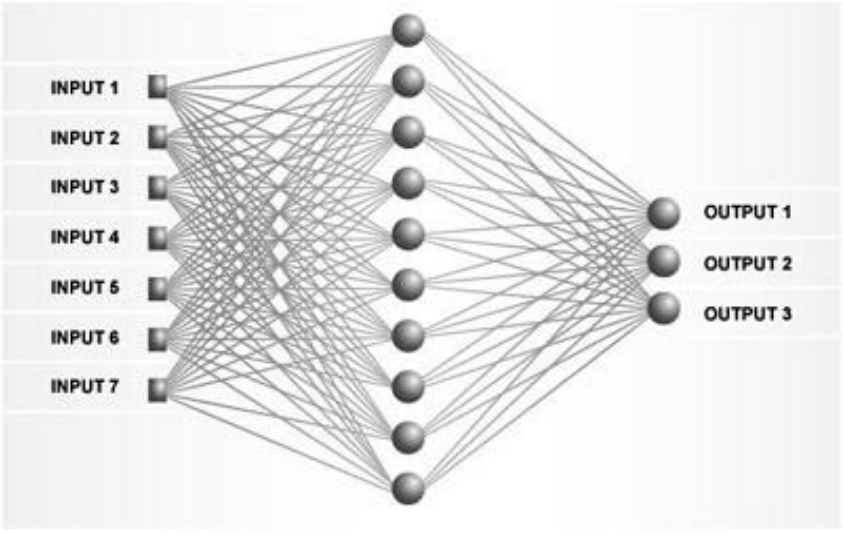
Implementing an RBF Classification Network

In principle it is easy to have an RBF network performs classification - we simply need to have an output function $y_k(\boldsymbol{x})$ for each class k with appropriate targets.

$$d_k^p = \begin{cases} 1 & \text{if pattern } p \text{ belongs to class } k \\ 0 & \text{otherwise} \end{cases}$$

For instance, for a three-class problem, we can construct a RBFN with three output neurons:





Can RBFN solve nonlinearly separable problem?

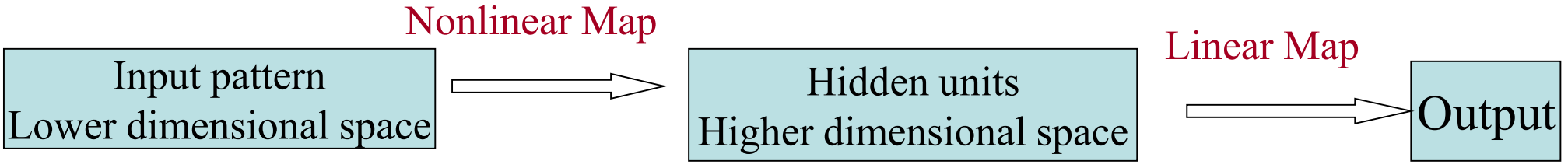
Yes.

All the output neurons are linear neurons.

All the computation power lies in the hidden neurons!

What is the magic of the hidden layer?

In many cases, the number of hidden units is larger than the number of inputs.

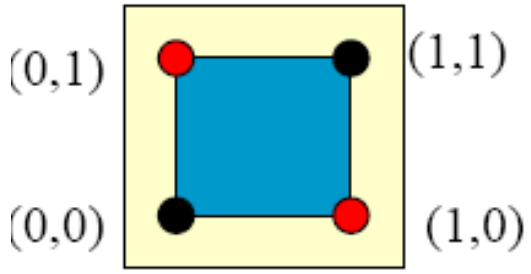


The underlying justification is found in *Cover's theorem* (section 5.2):
"A complex pattern classification problem *cast in a high dimensional space* ***nonlinearly*** is more likely to be *linearly separable* than in a low dimensional space".

We know that once we have linearly separable patterns, the classification problem is easy to solve.

Is this the same reason why MLP can solve XOR problem? Yes.

The XOR Problem Revisited



Recall that sensible RBFs are M Gaussian functions $\varphi_i(\boldsymbol{x})$ centred at training data points, i.e.,

$$\varphi_i(\boldsymbol{x}) = \exp\left(-\frac{M}{d_{\max}^2}\|\boldsymbol{x}-\boldsymbol{\mu}_i\|^2\right) \quad \text{where} \quad i = 1, 2, \ldots, M$$

To perform the XOR classification in an RBF network, we start by deciding how many basis functions we need. Given there are four training patterns and two classes, $M = 2$ seems a reasonable first guess.

Then we need to decide on the basis function centres. The two separated zero targets seem a good bet, so we can set $\boldsymbol{\mu}_1 = (0, 0)$ and $\boldsymbol{\mu}_2 = (1, 1)$ and the distance between them is $d_{\max} = \sqrt{2}$.

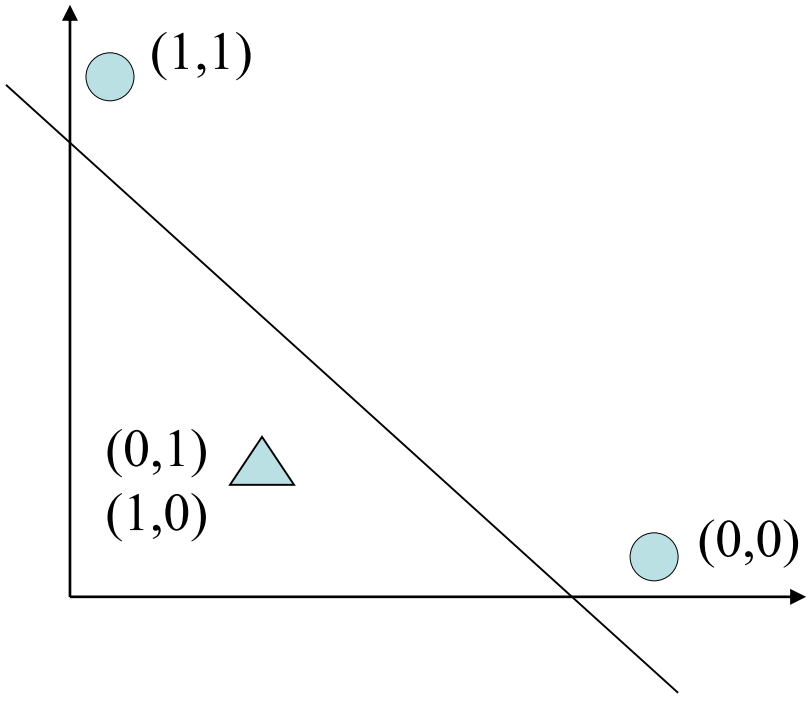
We thus have the two basis functions

$$\varphi_1(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x}-\boldsymbol{\mu}_1\|^2\right) \quad \text{with } \boldsymbol{\mu}_1 = (0,0)$$

$$\varphi_2(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x}-\boldsymbol{\mu}_2\|^2\right) \quad \text{with } \boldsymbol{\mu}_2 = (1,1)$$

This hopefully transforms the problem into a linearly separable form.

Since the hidden unit activation space is only two dimensional,
we can easily plot how the input patterns have been transformed.



$$\|x_2 - x_1\| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$\varphi_1(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{\mu}_1\|^2\right) \quad \text{with } \boldsymbol{\mu}_1 = (0,0)$$

$$\varphi_2(\boldsymbol{x}) = \exp\left(-\|\boldsymbol{x} - \boldsymbol{\mu}_2\|^2\right) \quad \text{with } \boldsymbol{\mu}_2 = (1,1)$$

p	x_1	x_2	ϕ_1	ϕ_2
1	0	0	1.0000	0.1353
2	0	1	0.3678	0.3678
3	1	0	0.3678	0.3678
4	1	1	0.1353	1.0000

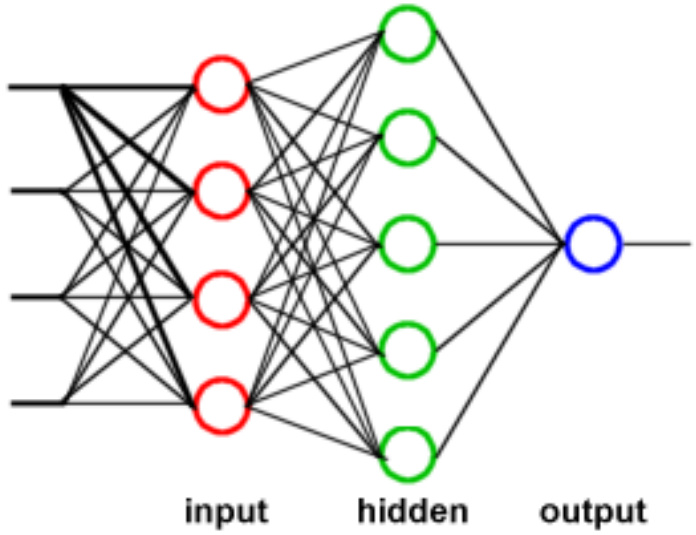
exp(-1) exp(-2)

We can see that the patterns are now linearly separable.

How many hidden units did we use? Only two!

Note that in this case we do not have to increase the dimensionality from the input space to the hidden unit/basis function space
- the non-linearity of the mapping is sufficient here.

Comparison of RBF Networks with MLPs



There are several similarities between RBFN and MLP:

Similarities

1. They are both non-linear feed-forward networks.
2. They are both universal approximators.
3. They are used in similar application areas.

It is not surprising, then, to find that there always exists an RBF network capable of accurately mimicking a specified MLP, or vice versa. However the two networks also differ from each other in a number of important aspects:

Differences

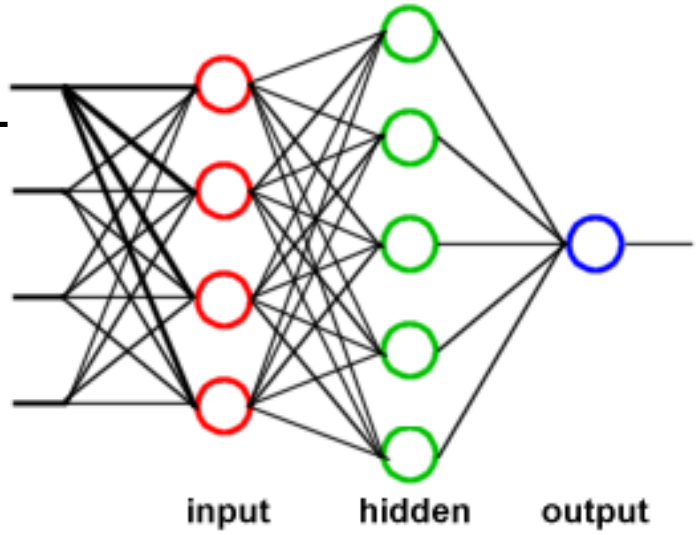
Is it possible for RBFN to have two hidden layers?

An RBF network (in its natural form) has a single hidden layer, whereas MLPs can have any number of hidden layers.

Do all the neurons compute in a similar fashion for MLP?

Do the hidden units and output neuron operate similarly for RBFN?

In MLPs the computation nodes (processing units) in different layers share a common neuron model, though not necessarily the same activation function.



In RBF networks the hidden nodes (basis functions) operate very differently, and have a very different purpose, to the output nodes.

In RBF networks, the argument of each hidden unit activation function is the *distance* between the input and the "weights" (RBF centres), whereas in MLPs it is the *weighted summation of the inputs (the induced local field).*

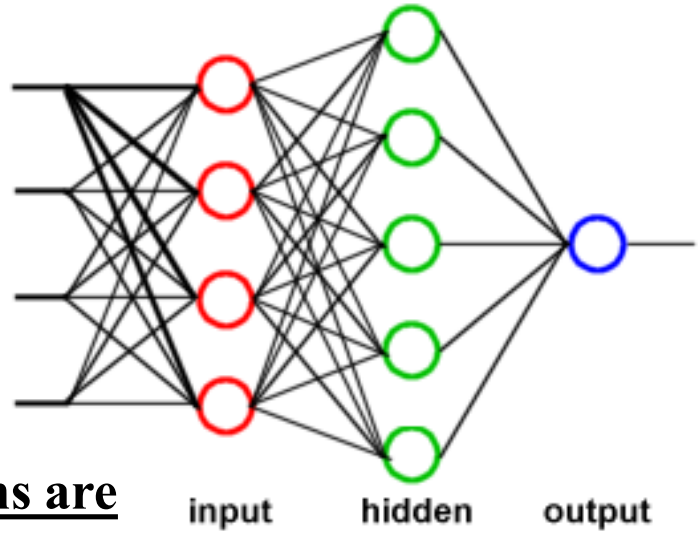
Does MLP learn the synaptic weights in two stages?

MLPs are generally trained with a single global supervised algorithm, whereas RBF networks are usually trained one layer at a time with the first layer unsupervised.

Although MLPs may require a smaller number of parameters, the RBF networks can be trained much faster for approximating non-linear input-output mappings.

What is the most significant difference between MLP and RBFN?

The functions of the neurons in MLP are the same, while the functions of the hidden neurons and output neurons are dramatically different for RBFN.



Given a MLP, if the activation functions of the hidden neurons are Gaussian, would that make it into a RBFN?

$$v(x) = b + \sum_{i=1}^{d} w_i x_i \qquad \varphi(v) = \exp(-\frac{v^2}{2\sigma^2})$$

RBF: a function which depends only on the radial distance.

No. v(x) is not a RBF, so the overall function is not a radial basis function!

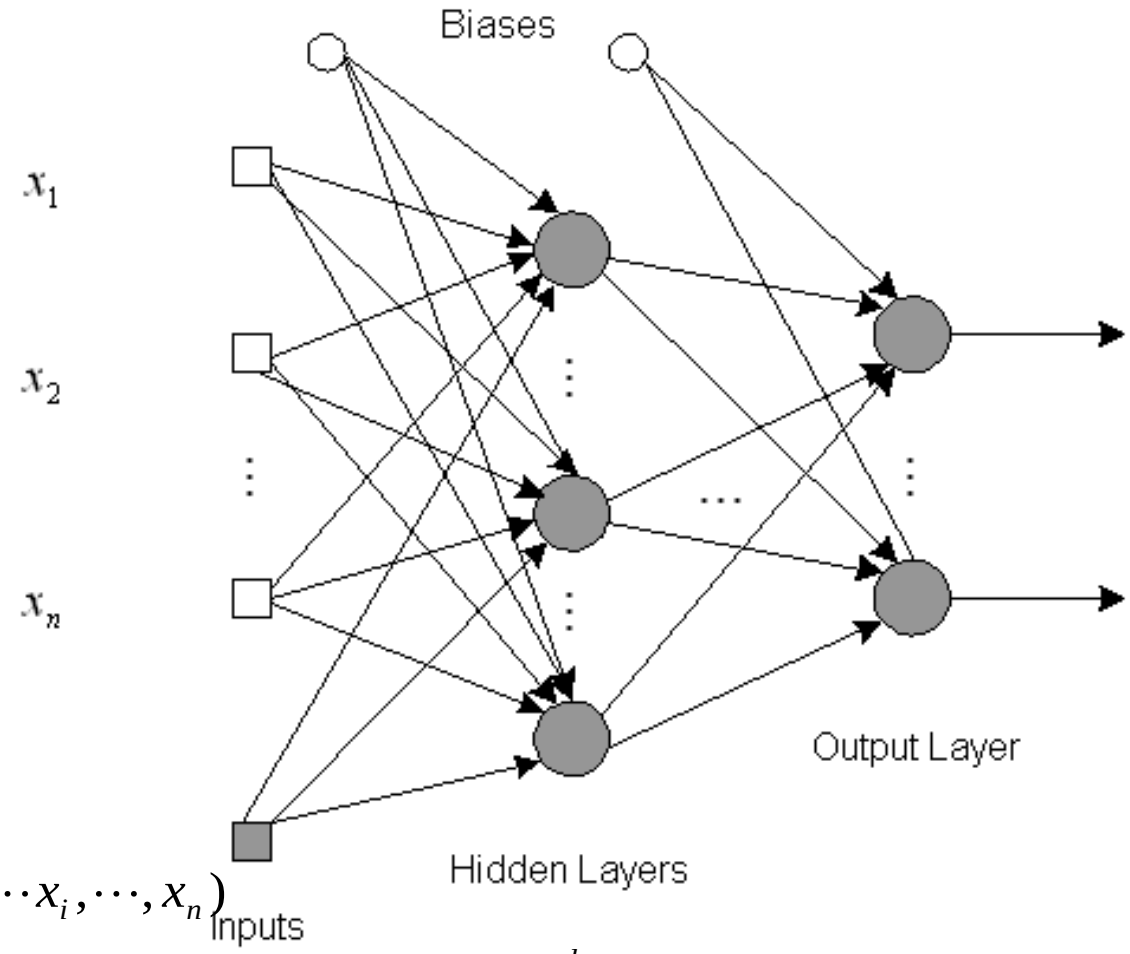
Is it possible to construct a network which is both the MLP and RBFN at the same time?

Yes. It is possible.

Please refer to the following papers for further technical details:

1. Ridella, S., Rovetta, S. and Zunino, R., "Circular backpropagation networks for classification", *IEEE Transactions on Neural Networks,* vol.8, no. 1, pp. 84-97, 1997.
2. Ding SQ and Xiang C, "From Multilayer Perceptron to Radial Basis Function Network: A Comparative Study," in the proceedings of 2004 *IEEE Conf. on Cybernetics and Intelligent Systems,* pp. 69-74, 2004.

MLP with additional inputs



Induced local field: $v(x, w) = b + \sum_{i=1}^{d} w_i x_i + w_{d+1} f(x)$

How to choose the additional input f(x) such that the induced local field v(x) becomes a radial basis function?

v(x) $\Longrightarrow$ v(||x-c||)=v(r)

Circular Backpropagation Network (CBP)

$$f(x_1, \cdots x_i, \cdots, x_n) = \sum_{i=1}^{n} x_i^2$$

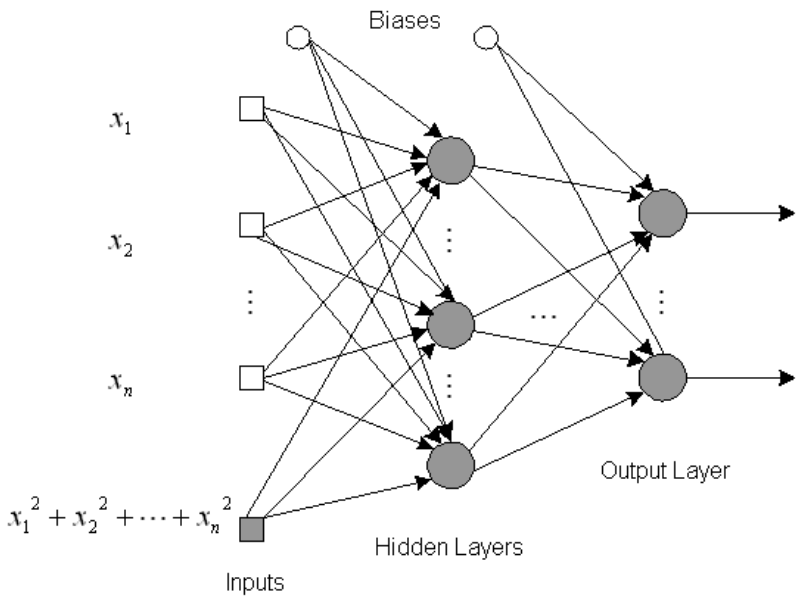
Induced local field $v(x, w) = b + \sum_{i=1}^{M} w_i x_i + w_{M+1} \sum_{i=1}^{M} x_i^2$

Completing the square,

$$\Rightarrow \quad v(x, w) = w_{M+1} \sum_{i=1}^{M} (x_i + \frac{w_i}{2w_{M+1}})^2 + **$$

$$\Rightarrow \quad v(x) = g(\|x - c\|^2 + \theta)$$

where $g = w_{M+1}$ $\quad c_i = -w_i / 2w_{M+1}$ $\quad \theta = \frac{1}{w_{M+1}} (b - \sum_{i=1}^{M} \frac{w_i^2}{4w_{M+1}})$



Let the radial distance $r = \| x - c \|$

Then $v(x) = g(\|x - c\|^2 + \theta) = gr^2 + a$

Is this a radial basis function?

Yes. v(x) is RBF, and hence $\phi(v)$ is also a RBF!

Therefore, CBP is also a RBFN

NUS National University of Singapore

CBP is both a MLP and RBFN!

How to implement it in MATLAB?

It is implemented as a normal MLP with all the built-in functions.
All you need to do is adding an extra term in the input vector!

Would it be better than both the conventional MLP and RBFN?

Simulation Studies

MLP---trainlm

CBP---trainlm

RBFN-1 (Random selection of centers)

RBFN-2 (K-means clustering method)

S-RBFN (Supervised learning based RBFN)

A Simple 1-D approximation example

$$f(x) = x^3 + 0.3x^2 - 0.4x$$

$$x \in [-1,1]$$

Goal: 0.0001 (error bound)

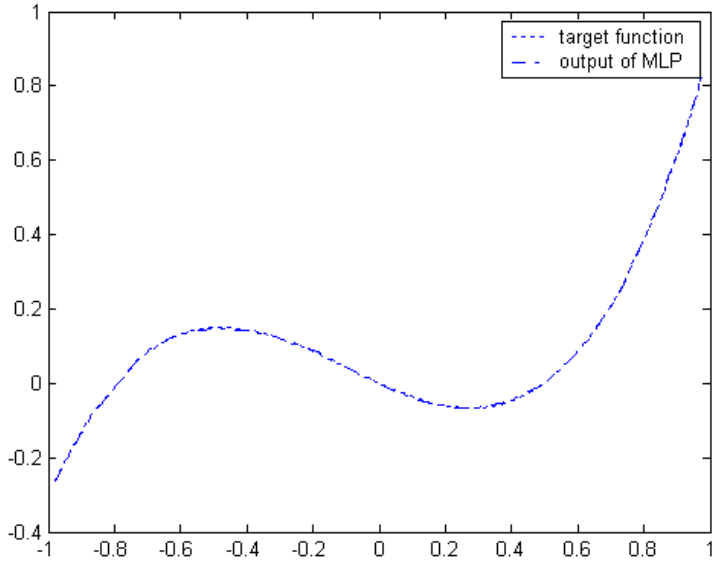


Table 1: The minimum number of hidden neurons needed

MLP	CBP	RBFN 1	RBFN 2	S-RBFN
3	2	5	5	3

NUS National University of Singapore

Gaussian hill and valley problem

$$f(x,y) = 3\exp(-(x-2)^2 - (y-2)^2) - 4\exp(-(x+2)^2 - y^2)$$

$$x, y \in [-4,4]$$

Goal: 0.05 (error bound)

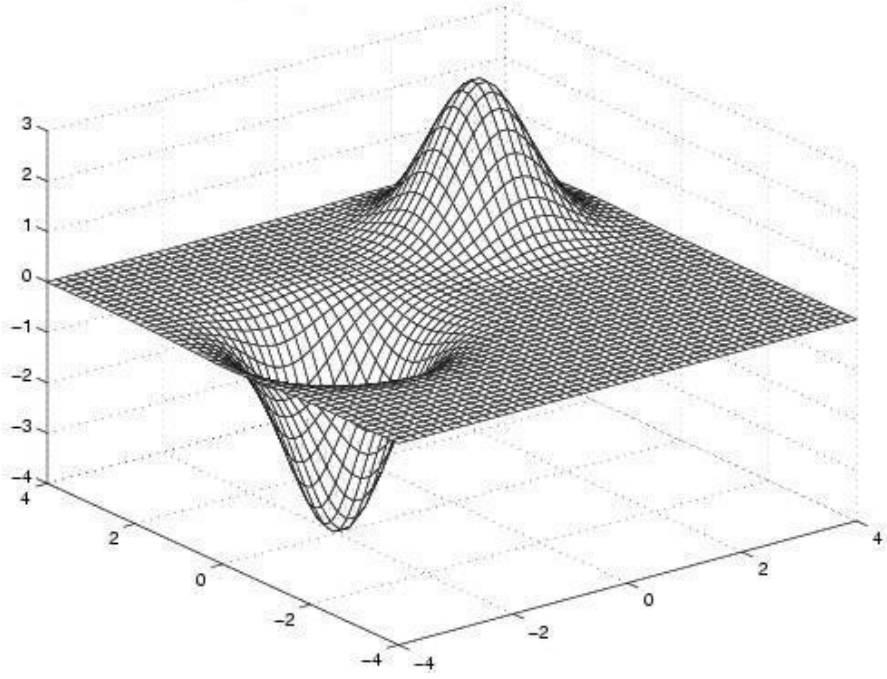


Table 3: The minimum number of hidden neurons needed

MLP	CBP	RBFN 1	RBFN 2	S-RBFN
30	2	93	92	2

Two spiral problem

Goal: 0 misclassification

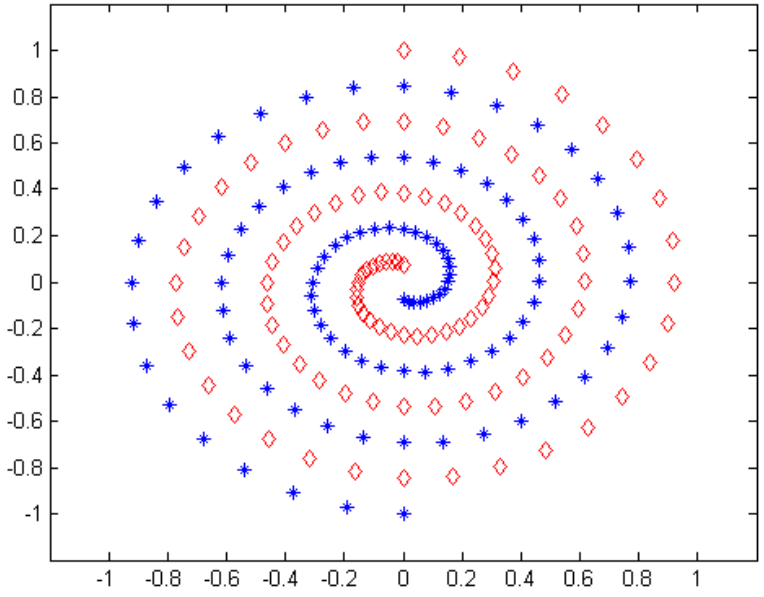
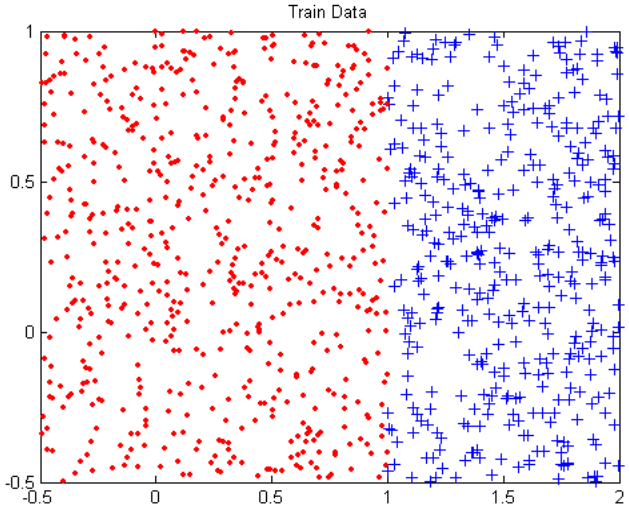


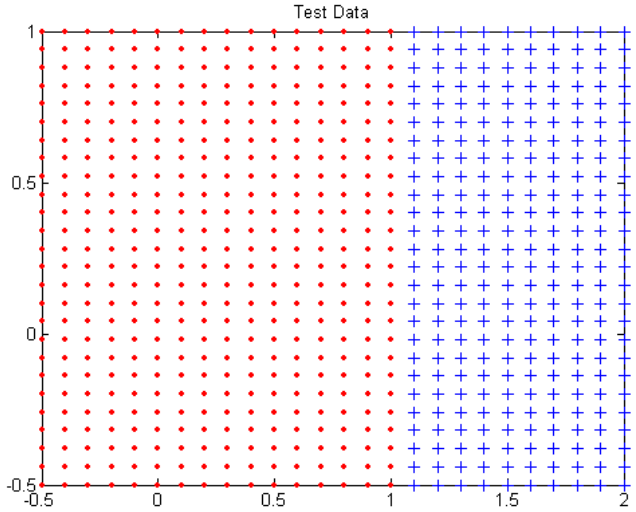
Table 4: The minimum number of hidden neurons needed

MLP	CBP	RBFN 1	RBFN 2	S-RBFN
29	8	130	98	Failed

Linear separable problem



(a) Training data



(b) Test data

Goal: 0 misclassification

Table 5: The minimum number of hidden neurons needed

MLP	CBP	RBFN 1	RBFN 2	S-RBFN
1	1	5	7	10

Conclusions from the Simulation Studies

CBP outperforms both MLP and RBFNs in most of the simulation studies in terms of accuracy and structure.

The number of weights required for MLP is generally smaller than that for RBFN.

Clustering method is just slightly better than random selection.

How about the performance of the one-stage trained RBFN and two-stage trained RBFN?

It is problem dependent!

History of RBFN

David Broomhead
(1950-2014)

Radial Basis Function (RBF) networks were first introduced by Broomhead & Lowe in 1988. 2 years after MLP with BP.

D. Broomhead was the professor at the department of Mathematics, the University of Manchester.

Although the basic idea of RBF was developed 30 years ago under the name -- *method of potential function*, the work by Broomhead & Lowe opened a new frontier in the neural network community.

Tomaso Poggio
(1947-present)

Another major contribution to the theory and design of RBFN is due to Poggio and Girosi in 1990. They emphasized the use of regularization theory in RBFN to improve the performance.

Tomaso Poggio (at MIT) is one of the founders of computational neuroscience. He made many key contributions, and RBFN is just one of them. He has visited NUS for a couple of times.

NUS National University of Singapore

Q & A…