



UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Chapter 5 – ANN Part 3 : Extended Neural Networks

SKEM 4173 – Artificial Intelligence

www.utm.my

innovative • entrepreneurial • global

5.1 Curve fitting

- Fitting a straight line
- Least Squares Regression

5.2 Gradient descent

- Introduction
- Gradient descent learning for ANN

5.3 Multilayer NN & Backpropagation learning

- Multilayer NN
- Backpropagation algorithm
- Example of BP application

theory

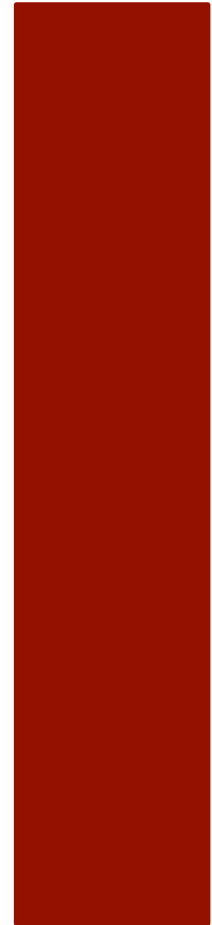
5.4 Radial Basis Function NN

- Introduction
- Training the RBF
- Least squares formula
- Choosing RBF centres

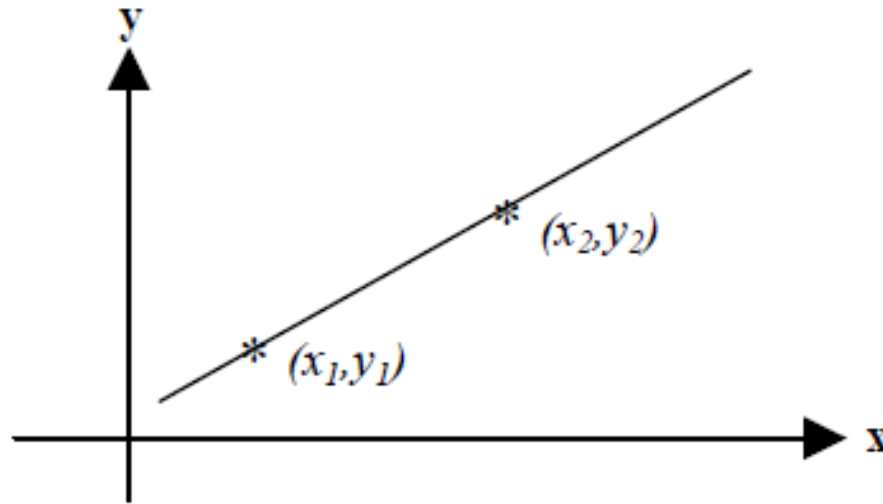
KIV

5.1

Curve fitting



5.1.1 Fitting a Straight Line



The equation representing the line is given by:

$$y = mx + c$$

5.1.1 (i)

- Parameters (m, c) that needs to be ‘tuned’ so that the line passes through the two points
- We want to choose m and c such that the perpendicular distance between the line and the two points is minimized.
- The quadratic error:

$$E(m, c) = [(mx_1 + c) - y_1]^2 + [(mx_2 + c) - y_2]^2$$

5.1.1 (ii)

- This error defines the sum of the perpendicular distances from the two points to the line.
- One way to solve this is to find the critical point of E by differentiating the above equation with respect to m and c

$$\frac{dE}{dm} = 2[mx_1 + c - y_1]x_1 + 2[mx_2 + c - y_2]x_2 \quad \boxed{5.1.1 \text{ (iii)}}$$

$$\frac{dE}{dc} = 2[mx_1 + c - y_1] + 2[mx_2 + c - y_2] \quad \boxed{5.1.1 \text{ (iv)}}$$

- To find the critical point, set equations to be equal to 0.

$$[mx_1 + c - y_1]x_1 + [mx_2 + c - y_2]x_2 = 0$$

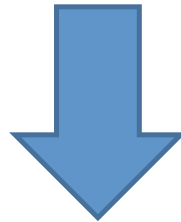
$$[mx_1 + c - y_1] + [mx_2 + c - y_2] = 0$$

- Rearrange:

$$c = \frac{y_1 + y_2 - mx_1 - mx_2}{2}$$

5.1.1 (v)

- Expand

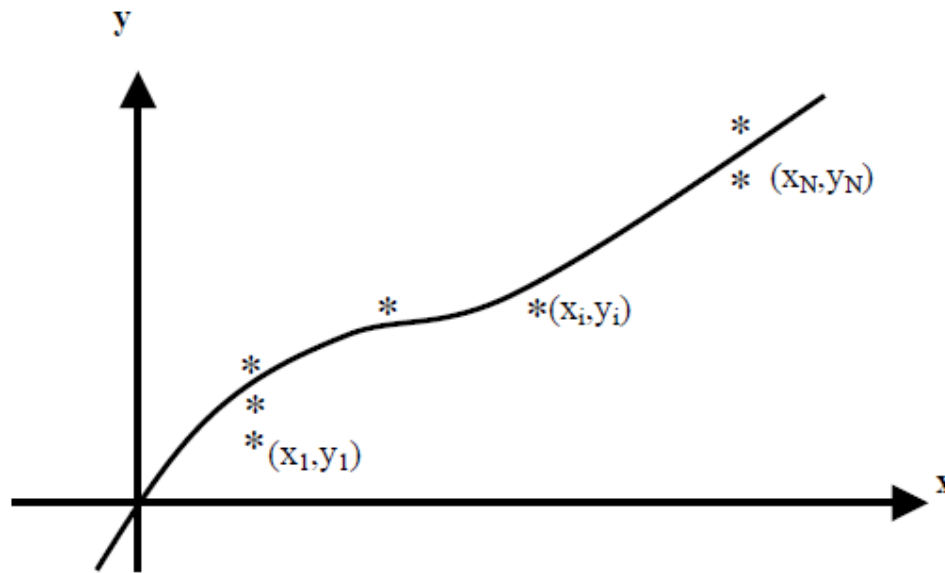


$$\therefore m = \frac{(y_1 - y_2)}{(x_1 - x_2)}$$

5.1.1 (vi)

- This equation is the well known slope formula.

5.1.2 Least Squares Regression



Fitting a model to several data points

Assumption: Say we conduct an experiment in the laboratory and the measurements we made are corrupted by noise, i.e.:

$$y_i = \eta(x_i, \theta) + \gamma$$

5.1.2 (i)

Now we want to fit a polynomial given by

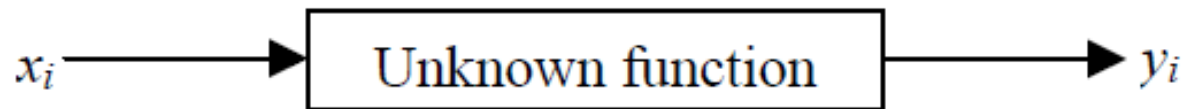
$$\eta(x, \theta) = \theta_0 + \theta_1 x^1 + \dots + \theta_m x^m$$

5.1.2 (ii)

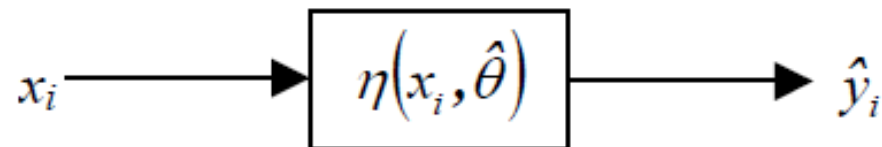
to the data sets

$$D = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

5.1.2 (iii)



\approx



Approximating the experimental data

Define the quadratic error as

$$E(\theta) = \sum_{i=1}^N (\eta(x_i, \theta) - y_i)^2$$

5.1.2 (iv)

Question: What are the best values of $\theta_0, \dots, \theta_m$, such that the square error is minimized?

To simplify the derivation, write the polynomial as

$$\eta(x, \theta) = g(x)^T \theta$$

5.1.2 (v)

where

$$g(x) = [1 \quad x \quad x^2 \quad \dots \quad x^m]^T$$

$$\theta = [\theta_0 \quad \theta_1 \quad \theta_2 \quad \dots \quad \theta_m]^T$$

Given a set of N sampled points $(x_1, y_1), (x_2, y_2), (x_3, y_3), \dots, (x_N, y_N)$,

let

$$F(g(x_i)) = \begin{bmatrix} g(x_1)^T \\ g(x_2)^T \\ \vdots \\ g(x_N)^T \end{bmatrix}$$

5.1.2 (vi)

where F is an $N \times (m+1)$ matrix and

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

5.1.2 (vii)

Define the square error between the actual measurements, y , and the output of the polynomial $\eta(x, \theta)$ as

$$e(\theta) = (y - F\theta)^T (y - F\theta)$$

5.1.2 (viii)

- Expanding:

$$e(\theta) = \theta^T F^T F \theta - 2y^T F \theta + y^T y \quad \boxed{5.1.2 \text{ (ix)}}$$

- Differentiate and setting the derivative equal to 0:



Finally

$$\hat{\theta} = (F^T F)^{-1} F^T y \quad N \geq (m+1), \quad \boxed{5.1.2 \text{ (x)}}$$

If $N < (m+1)$, then it is possible to use the pseudo inverse of F

$$\hat{\theta} = F^T (F F^T)^{-1} y \quad N < (m+1) \quad \boxed{5.1.2 \text{ (xi)}}$$

Hence, the predicted output of the ‘trained’ polynomial is given by

$$\hat{y}(x, \hat{\theta}) = g(x)^T \hat{\theta}$$

5.1.2 (xii)

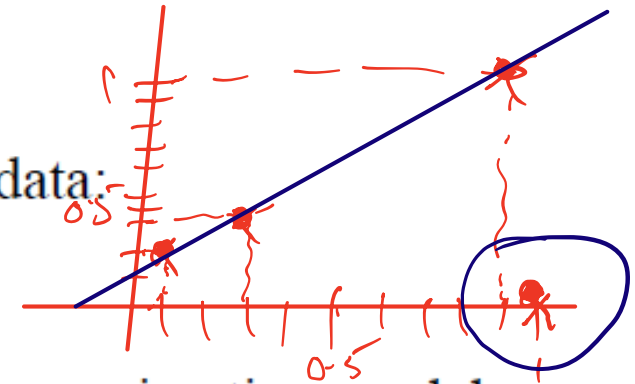
Note: Recall that $e(\theta) = \theta^T F^T F \theta - 2y^T F \theta + y^T y$. Here e depends linearly on θ since the polynomial is *linear* (i.e. $\eta(x, \theta) = g(x)^T \theta$

- The polynomial is called a **linear model** since it depends linearly on its adaptive parameters, although it depends non-linearly on its input variables.
- We will look at the case when the regression model is **non-linear**, i.e. when η is an Artificial Neural Network.

Example 5.1

Given a curve fitting problem with the following data:

$$D = \{(x, y) : (1, 0), (0.1, 0.2), (0.3, 0.3), (0.9, 1)\}$$



where x is the input and y the output. State an approximating model that you would use to solve the problem and justify your decision. Also, give a suggestion on the appropriate number of adaptive parameters of your model and explain why you choose it to be so.

Example 5.2

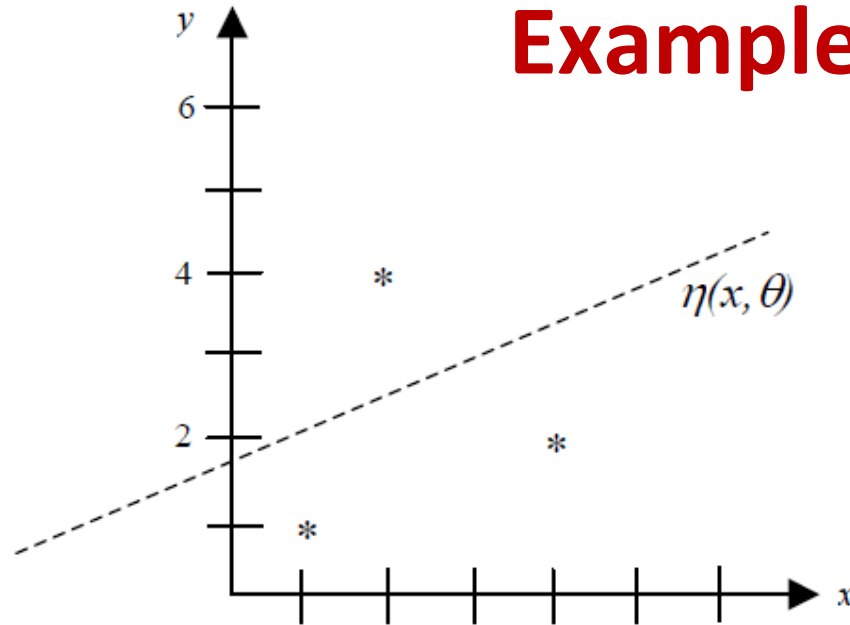


Figure E2.2 illustrates a function approximation problem with 3 experimental data indicated by ‘*’. A straight line $\eta(x, \theta) = \theta_0 + \theta_1 x$ is to be fitted to the data sets to approximate the unknown function that produced the experimental data. The optimum values of the line parameters can be found using the least squares formula:

$$\begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = (F^T F)^{-1} F^T y$$



- Find the matrix F and the vector y in this case.
- Write a MATLAB code to solve the problem and to plot the

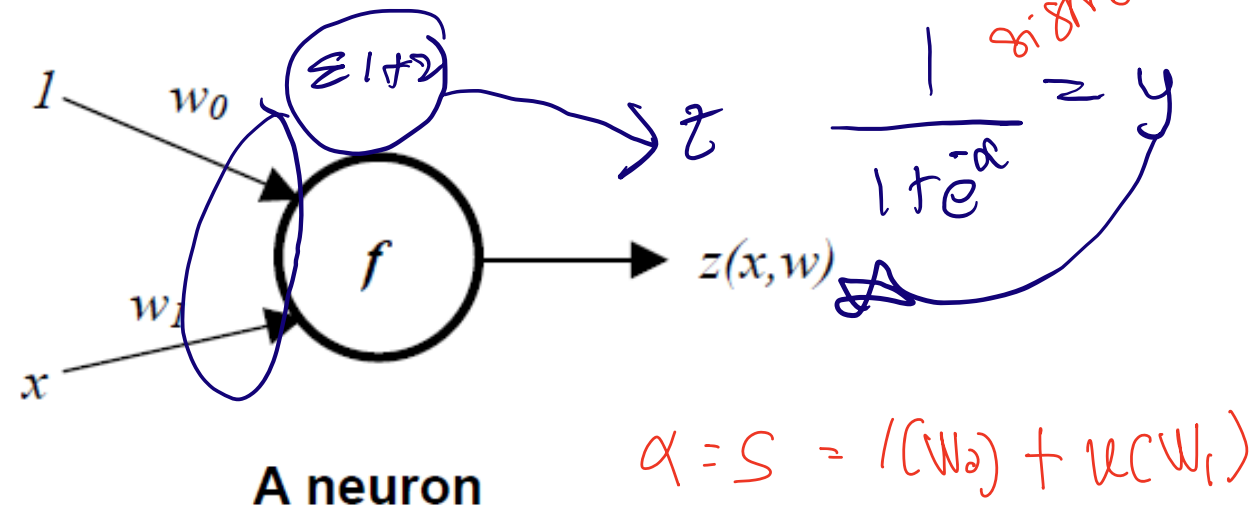
```
y=[1 4 2]';  
F=[1 1; 1 2; 1 4];  
theta=inv(F'*F)*F'*y  
  
x=[0:0.1:6]';  
for i=1:length(x)  
    yhat(i,1)=[1 x(i,1)]*theta;  
end  
  
figure; plot(x,yhat,[1 2 4]',y,'r*'); grid;  
axis([0 6 0 6]);  
xlabel('x'); ylabel('y');
```

5.2

Gradient descent and ANN

5.2.1 Introduction to Gradient Descent

- Here, we will look at how to solve the regression problem using a non-linear model
- In this case, the non-linear model that we will use is an ANN.





The output of the neuron shown can be written as:

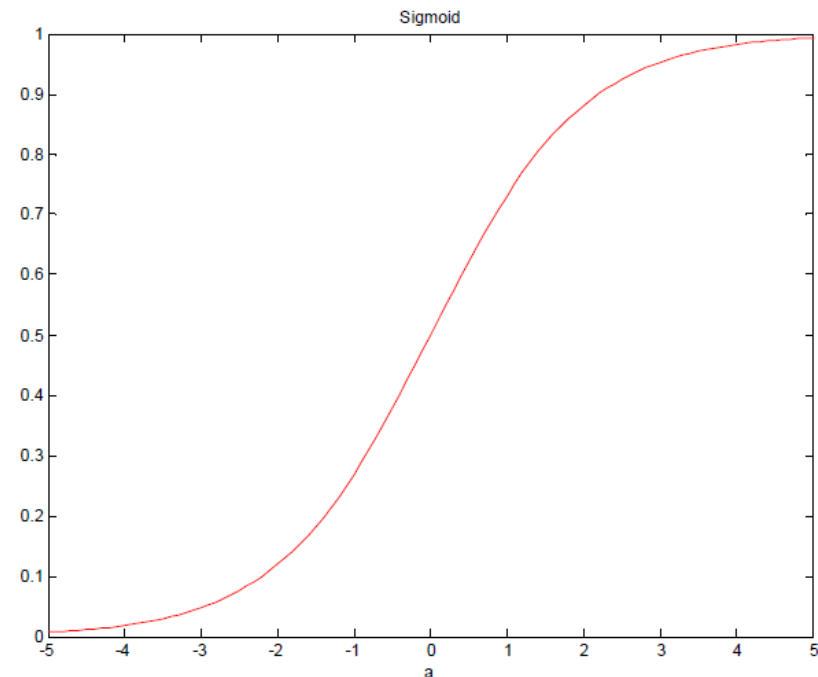
$$z(x, w) = f(xw_1 + w_0)$$

5.2.1 (i)

Normally, the activation function f , is chosen to be the differentiable, 'S-shaped', sigmoid activation function given by

$$f(a) = \frac{1}{1 + e^{-a}}$$

5.2.1 (ii)



If the sigmoid is used, then the output of the neuron is now:

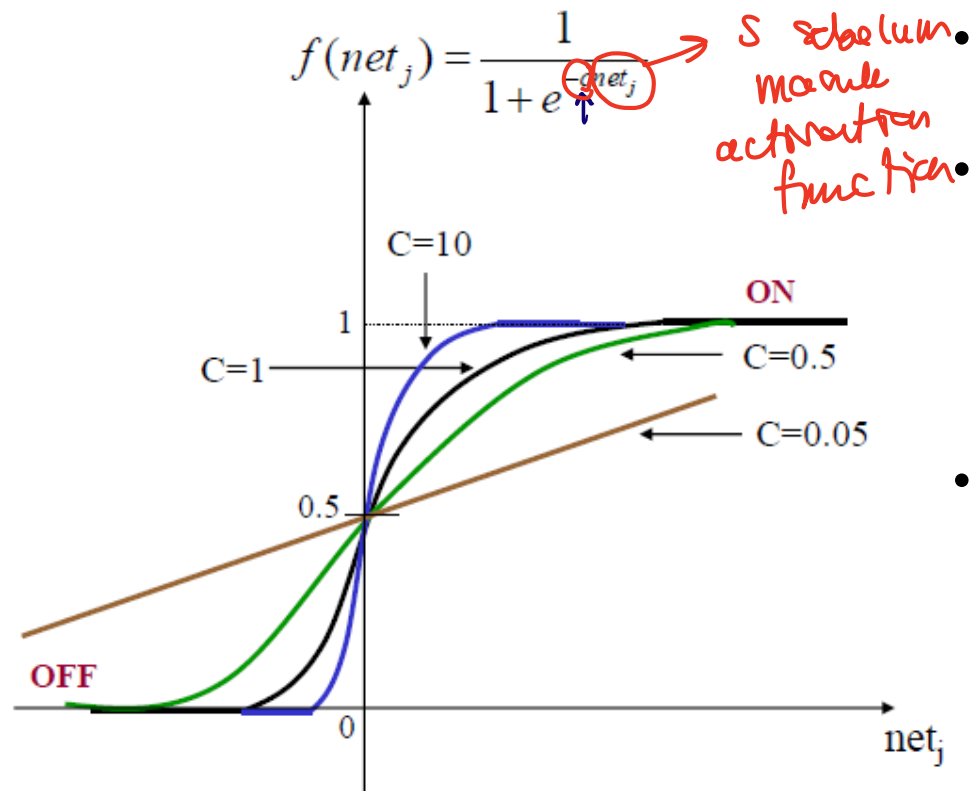
$$z(x, w) = \frac{1}{1 + e^{-(xw_1 + w_0)}}$$

5.2.1 (iii)

Notice now that the output z , depends *non-linearly* on the adaptive parameters w_1 and w_0 , and also on its input x . The question now is, how to find the optimum values for the adaptive parameters when the model is used to solve the regression/curve fitting problem?

Sigmoid activation function (additional notes):

- Also known as **logistic** activation function.
- This function is semilinear in characteristic, differentiable and produces a value between 0 and 1.



c controls the firing angle of the sigmoid.

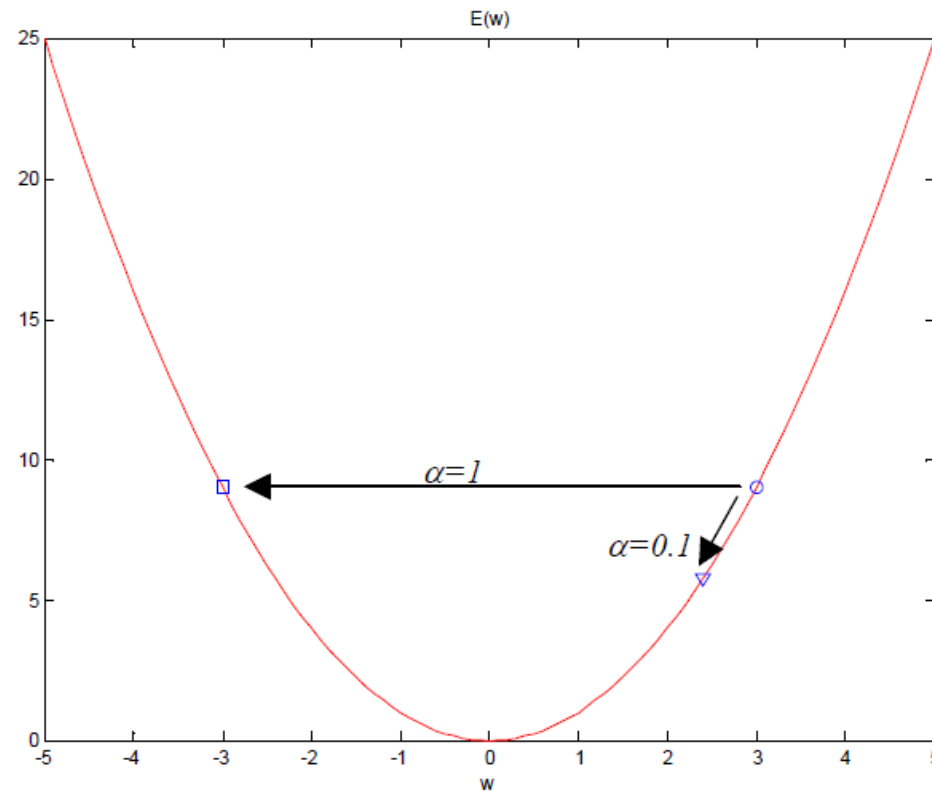
When c is large, the sigmoid becomes like a threshold function and when c is small, the sigmoid becomes more like a straight line (linear).

- When c is large learning is much faster but a lot of information is lost, however when c is small, learning is very slow but information is retained.

- Non-linear optimization approaches generally fall into two categories:
 1. stochastic based
 2. or gradient based.
- Here, we will introduce a non-linear gradient based optimization approach called **Gradient Descent** or **Steepest Descent**.

$$E = (y - t)^2$$

- The basic idea is to start with a random value of initial weights and move iteratively along the direction of the **negative gradient of the error function**. As the number of iteration increases, we would expect to find **a set of weights that would minimize the error function**.



Assume that the error function is just a quadratic function:


$$E(w) = w^2$$

5.2.1 (iv)

- The optimum value for w will be when it minimizes E . In this case, it is simple since we can see from the above figure that the minimum is at $w=0$.

- However, in general, the error function is much more complicated where normally it involves **multiple minima** with a higher dimensional weight space.

learning rate



$$\frac{dE}{dw} = 2w$$

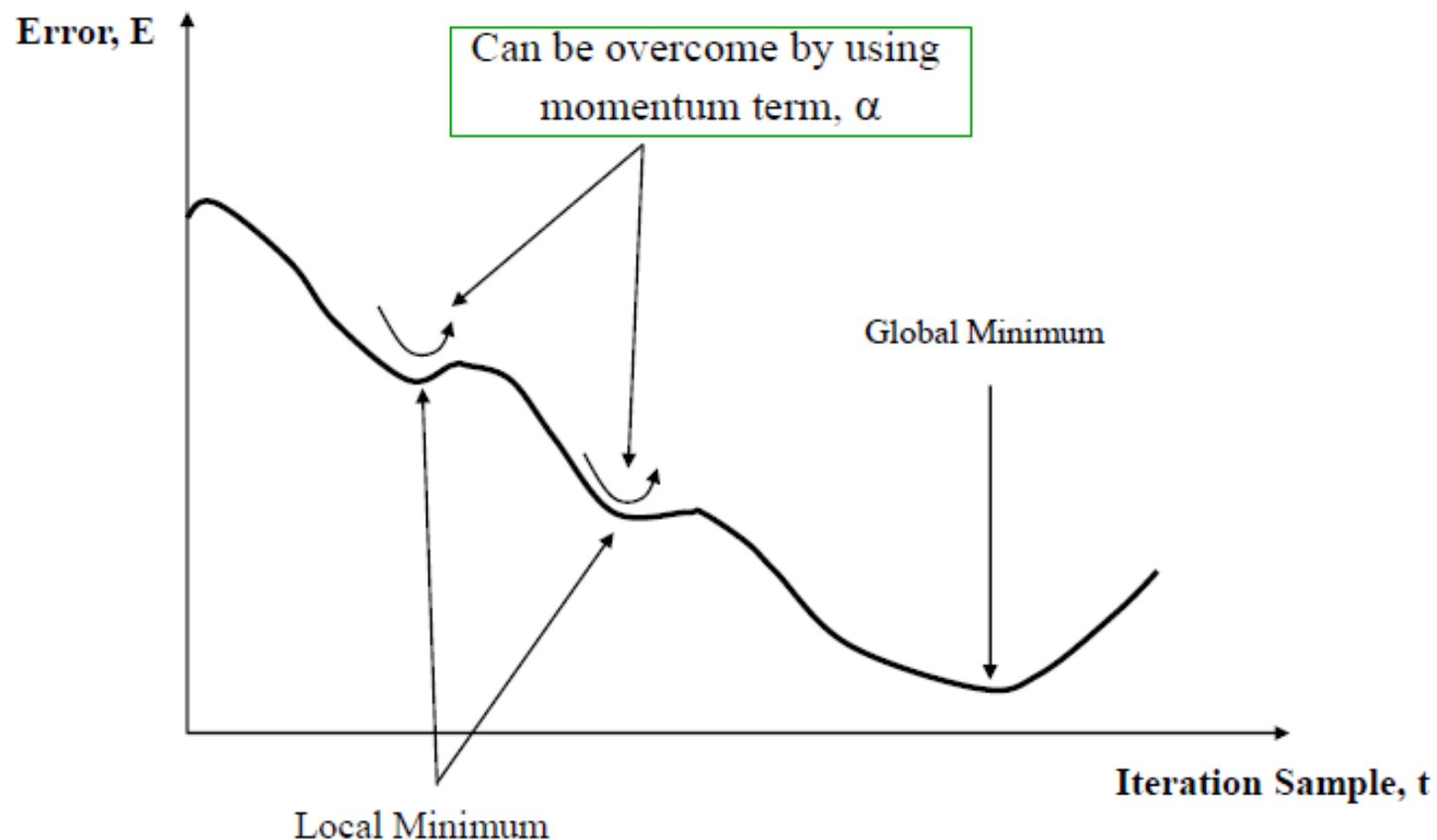
5.2.1 (v)

$$w(new) = w(old) - \alpha \frac{dE}{dw} \Big|_{w(old)}$$

5.2.1 (vi)

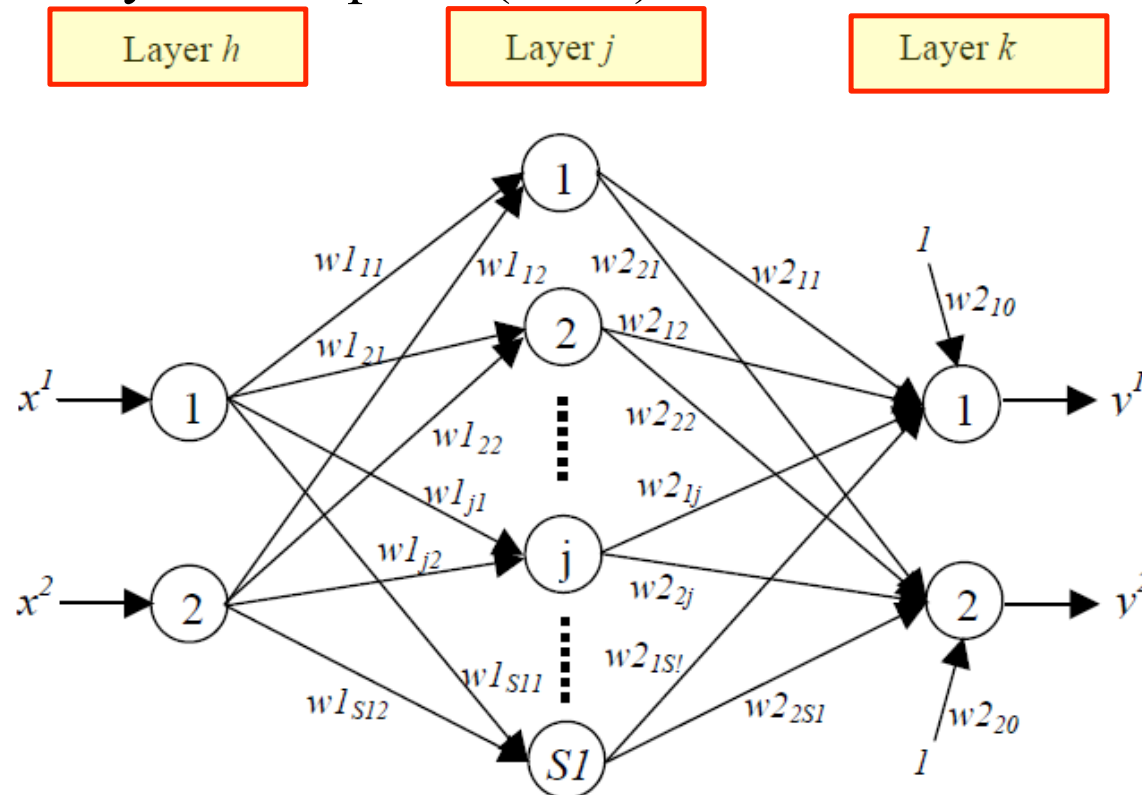
- Note too that the **speed** of convergence depends on the value of α . If it is set to **be too small**, then we need to repeat the update several times before ending at the minimum. However, if **α is too big**, then we might miss the minimum altogether

The Gradient Descent Approach is trying to find the global minimum



7.2.2 Gradient Descent Learning for ANN

- Consider a feedforward two-input two-output, single hidden layer Multilayer Perceptron (MLP).



- $S1$ represents the total number of **neurons in the hidden layer**, w are the **adaptive weights and biases** and x and y represents the MLP input and output respectively.



- Let the neurons in layers h and k use linear activation function while those in layer j use sigmoid activation function.
- For the linear output neurons,

$$y^1 = w2_{10} + w2_{11}z_1 + w2_{12}z_2 + \dots + w2_{1s1} = \sum_{j=0}^{s1} w2_{1j}z_j$$

$$y^2 = w2_{20} + w2_{21}z_1 + w2_{22}z_2 + \dots + w2_{2s1} = \sum_{j=0}^{s1} w2_{2j}z_j$$

hence

$$y^k = \sum_{j=0}^{s1} w2_{kj}z_j$$

where

$$z_0 = 1 \quad ; \quad z_j = f(a_j) = \frac{1}{1 + e^{-a_j}} \quad ; \quad a_j = \sum_{h=1}^2 w1_{jh}x^h$$

sigmoid

$$\frac{dE}{dw} = 2w$$

- Sum of squares error function:

$$E^k(w) = \sum_{i=1}^N (y^k(x_i, w) - t_i^k)^2 \quad w(new) = w(old) - \alpha \frac{dE}{dw} \Big|_{w(old)}$$

- The goal of the ANN ‘learning’ is now to find a set of weights, w , such that the error, $E^k(w)$ is minimized by using an iterative search algorithm (gradient descent)

- Gradient descent learning:**

1. Initialize the weights with random numbers.
2. Update the weights such that at step τ . The update rule can be written as:

$$w_{\tau+1} = w_{\tau} - \alpha \nabla E \Big|_{w_{\tau}} \quad W_{new} = W_{old} - \alpha \frac{dE}{dw} \Big|_{old \text{ weights}}$$

3. Repeat step 2 until a stopping criterion is reached. Examples of stopping criteria are:

- Maximum epoch reached.
- Minimum error reached. E_{min}
- Weight changes too small.

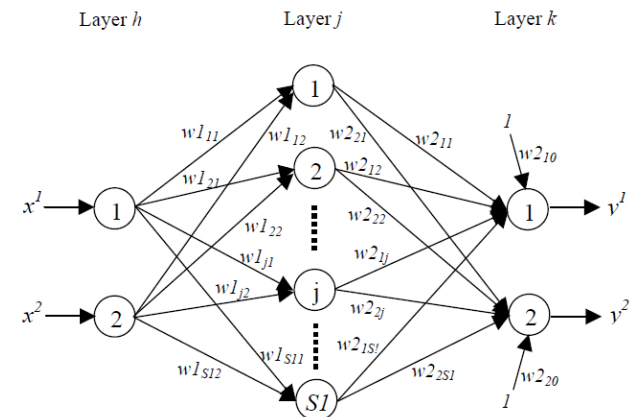
- Consider first of all the biases in **layer k**:

$$\nabla E|_{w_\tau} = \frac{\partial E^k}{\partial w_{2_{k0}}} |_{w_\tau} = \frac{\partial}{\partial w_{2_{k0}}} \left(\sum_{i=1}^N (y^k(x_i, w) - t_i^k)^2 \right)$$

$$= 2 \sum_{i=1}^N (y^k(x_i, w) - t_i^k)$$

Hence,

$$w_{2_{k0}(\tau+1)} = w_{2_{k0}(\tau)} - \alpha \left[2 \sum_{i=1}^N (y^k(x_i, w) - t_i^k) \right]$$



- Next, consider the weights in **layer k-j**.

$$\frac{\partial E^k}{\partial w_{2_{kj}}} = \frac{\partial}{\partial w_{2_{kj}}} \left(\sum_{i=1}^N (y^k(x_i, w) - t_i^k)^2 \right) = 2 \sum_{i=1}^N (y^k(x_i, w) - t_i^k) \frac{\partial y^k(x_i, w)}{\partial w_{2_{kj}}}$$

$$w_{2_{kj}(\tau+1)} = w_{2_{kj}(\tau)} - \alpha \left[2 \sum_{i=1}^N z_j (y^k(x_i, w) - t_i^k) \right]$$

- Finally, consider the weights in layer j - h . This is a little bit more complicated since **both errors E^1 and E^2 will be influenced** by any particular weight in this layer.

$$\frac{\partial E}{\partial w_{jh}} = \frac{\partial E^1}{\partial y^1(x_i, w)} \frac{\partial y^1(x_i, w)}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{jh}} + \frac{\partial E^2}{\partial y^2(x_i, w)} \frac{\partial y^2(x_i, w)}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{jh}}$$

$$\frac{\partial E}{\partial w_{jh}} = \sum_{k=1}^2 \left[\frac{\partial E^k}{\partial y^k(x_i, w)} \frac{\partial y^k(x_i, w)}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{jh}} \right]$$

- Since the hidden layer neurons uses sigmoid activation function, we can expand the above equation:

$$\frac{\partial E}{\partial w_{jh}} = 2 \sum_{i=1}^N \sum_{k=1}^2 \left[(y^k(x_i, w) - t_i^k) w_{kj} z_j (1 - z_j) x_i^h \right]$$

- Therefore, the update rule for the weights in layer j - h is given by:

$$w1_{jh(\tau+1)} = w1_{jh(\tau)} - 2\alpha \sum_{i=1}^N \left[\sum_{k=1}^2 (y^k(x_i, w) - t_i^k) w2_{kj} z_j (1 - z_j) x_i^h \right]$$

- Summary:

Weights in layer j - h
$w1_{jh(\tau+1)} = w1_{jh(\tau)} - 2\alpha \sum_{i=1}^N \left[\sum_{k=1}^2 (y^k(x_i, w) - t_i^k) w2_{kj} z_j (1 - z_j) x_i^h \right]$
Weights in layer k - j
$w2_{kj(\tau+1)} = w2_{kj(\tau)} - \alpha \left[2 \sum_{i=1}^N z_j (y^k(x_i, w) - t_i^k) \right]$
Biases in layer k
$w2_{k0(\tau+1)} = w2_{k0(\tau)} - \alpha \left[2 \sum_{i=1}^N (y^k(x_i, w) - t_i^k) \right]$

5.2.3 Batch and Sequential Version of Gradient Descent

- The gradient descent update rule that was previously is one version of gradient descent called the ‘**Batch version**’.
- Another version of gradient descent is the ‘**Sequential or Online version**’ where the update is done one pattern at a time, that is, an update is made for each input-output training pair.
- The resulting difference is that the **error surface** is continuously changing whenever an update is made which is different than in the batch case where the error surface is fixed.
- For the sequential version, the error function is simply

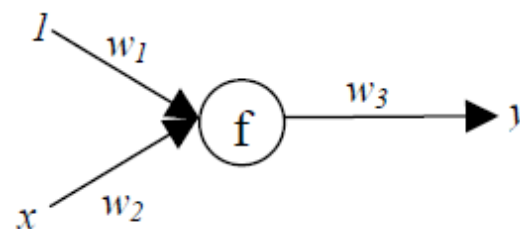
$$E_i^k(w) = (y^k(x_i, w) - t_i^k)^2$$

- An **advantage** of sequential approach over batch methods is when there is a high degree of redundant information in the data set.
- Summary of Sequential Gradient Descent updating rule:

Weights in layer j-h
$w1_{jh(\tau+1)} = w1_{jh(\tau)} - 2\alpha \sum_{k=1}^2 (y^k(x_i, w) - t_i^k) w2_{kj} z_j (1 - z_j) x_i^h$
Weights in layer k-j
$w2_{kj(\tau+1)} = w2_{kj(\tau)} - 2\alpha z_j (y^k(x_i, w) - t_i^k)$
Biases in layer k
$w2_{k0(\tau+1)} = w2_{k0(\tau)} - 2\alpha (y^k(x_i, w) - t_i^k)$

Example 5.3

For the neuron shown in Figure E3.1, the activation function used is the sigmoid function given by $f(\varsigma) = \frac{1}{1 + e^{-\varsigma}}$. The input output training data are given by $(x,y) = \{(1,2), (3,4), (5,6)\}$. The square error function $E = \sum_{i=1}^N (y(x_i, w) - t_i)^2$ and the steepest descent training rule are used to train the neuron. Complete Table E3.1 if the learning rate used is $\alpha = 0.1$.



Iteration	w_1	w_2	w_3
0	1	2	0
1			
2			
3			

5.3

Backpropagation (BP) Algorithm for ANN

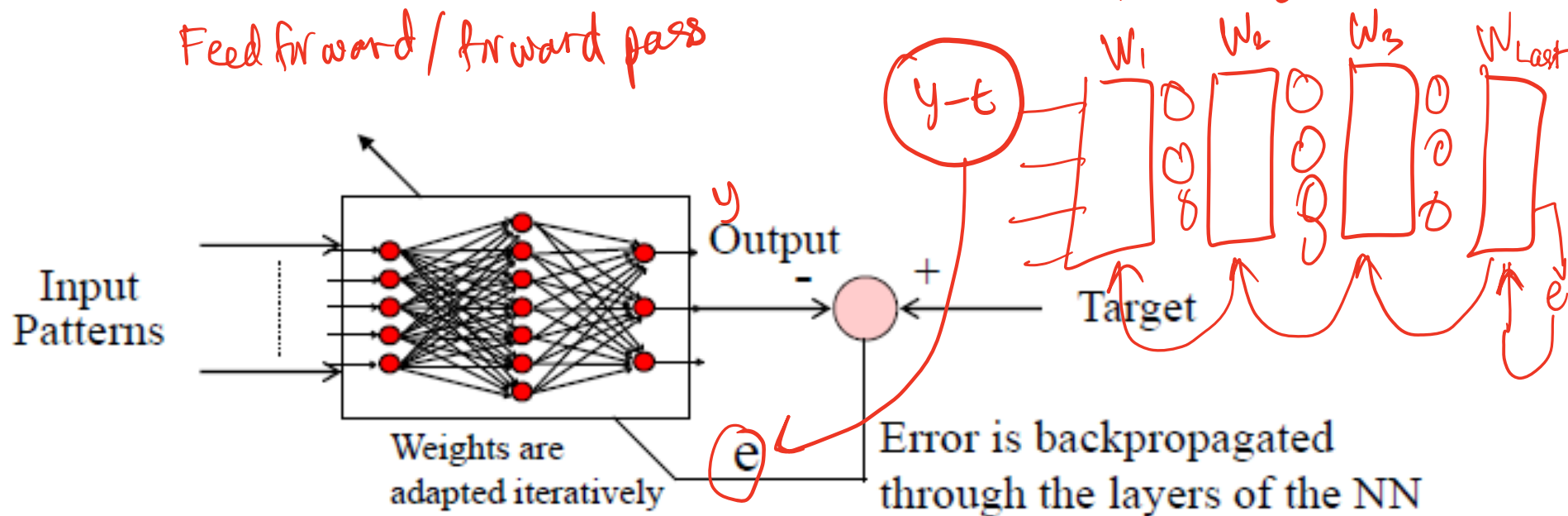
5.3.1 Introduction

- The BP algorithm is perhaps the most popular and widely used neural paradigm.
- The BP algorithm is based on the generalized delta rule proposed by the PDP research group in **1985** headed by Dave Rumelhart based at **Stanford University**, California, U.S.A.
- The BP algorithm overcame the **limitations of the perceptron** algorithm.
- Among the first applications of the BP algorithm is speech synthesis called NETalk developed by Terence Sejnowski.
- The BP algorithm created a sensation when a large number of researchers used it for many applications and reported its successful results in many technical conferences.

5.3.2 Learning mode

- Before the BP can be used, it requires **target** patterns or signals as it a **supervised** learning algorithm.
- The configuration for training a neural network using the BP algorithm is shown in the figure below in which the training is done offline.
- The objective is to **minimize the error** between the target and actual output and to find Δw .
- The error is calculated at every iteration and is backpropagated through the layers of the ANN to adapt the weights.

- The weights are adapted such that the **error is minimized**. Once the error has reached a justified minimum value, the training is stopped, and the neural network is reconfigured in the recall mode to solve the task.



5.3.3 Derivation of BP Algorithm

- From the derivation of Gradient Descent, notice the fact that the gradient of E with respect to the weights in different layers of the ANN model **share many common terms**.
- What we want to do is to **reuse** the terms that has been already computed in order to **reduce amount of computations** when training the network.
- Referring to the **sequential version** of gradient descent and the ANN model in Section 5.2.2, define the derivative of E with respect to the input of the neurons as:

$$\delta_k = \frac{\partial E_i^k}{\partial a_k} = \frac{\partial E_i^k}{\partial y_k} = 2(y^k(x_i, w) - t_i^k)$$

5.3.3 (i)

**Activation
function at
output layer-k**

$$\delta_j = \frac{\partial E_i}{\partial a_j} = \sum_{k=1}^2 \frac{\partial E_i^k}{\partial a_j} = \sum_{k=1}^2 \frac{\partial E_i^k}{\partial y^k} \frac{\partial y^k}{\partial z_j} \frac{\partial z_j}{\partial a_j}$$

Refer to Slide 31

$$= \sum_{k=1}^2 2[(y^k(x_i, w) - t_i^k) w_{kj} z_j (1 - z_j)]$$

$$= \sum_{k=1}^2 [\delta_k w_{kj} z_j (1 - z_j)]$$

5.3.3 (ii)

- Observe the fact that the δ for a lower layer, say j , is computed by reusing the value of δ from a higher layer, say k .
- This is essentially the principle of backpropagation where the values of the **derivative in a lower layer**, can be computed using some of the values already computed when finding the **derivatives in a higher layer**.

- Next, define the formula for the derivative of E with respect to the weights in the ANN as:

Refer to Slide 30

$$\frac{\partial E_i}{\partial w_{kj}} = \delta_k z_j$$

5.3.3 (iii)

- which means the derivative of the error, E , with respect to a particular weight in the feedforward ANN is given by the **delta in a higher layer times the output of the neuron in a lower layer**. Similarly, we can write

$$\frac{\partial E_i}{\partial w_{jh}} = \delta_j z_h = \delta_j x^h$$

5.3.3 (iv)

- Therefore, the computational cost of finding the derivative of the error to the adaptive weights in the ANN can be reduced by backpropagating the δ in a higher layer down to a lower layer.
- This is very **significant in large networks** with several hidden layers.

- Finally to summarize, referring to the following figure, the backpropagation formula can be generalized as:

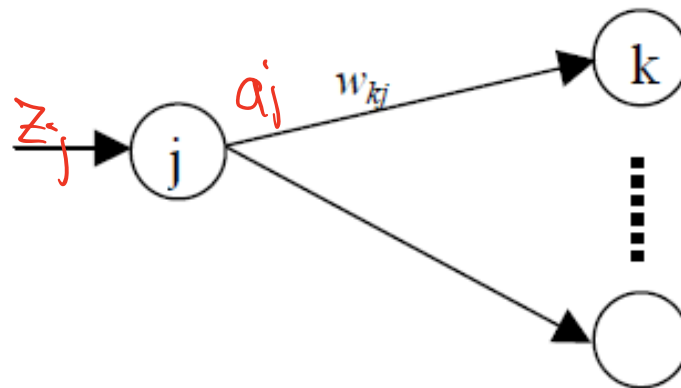
$$\delta_j = \frac{\partial E}{\partial a_j} = \frac{dz_j}{da_j} \sum_k \delta_k w_{kj}$$

Error Function → $\frac{\partial E}{\partial w_{kj}} = \delta_k z_j$

5.3.3 (v)

5.3.3 (vi)

- where z and a is the input and output of neuron j respectively.

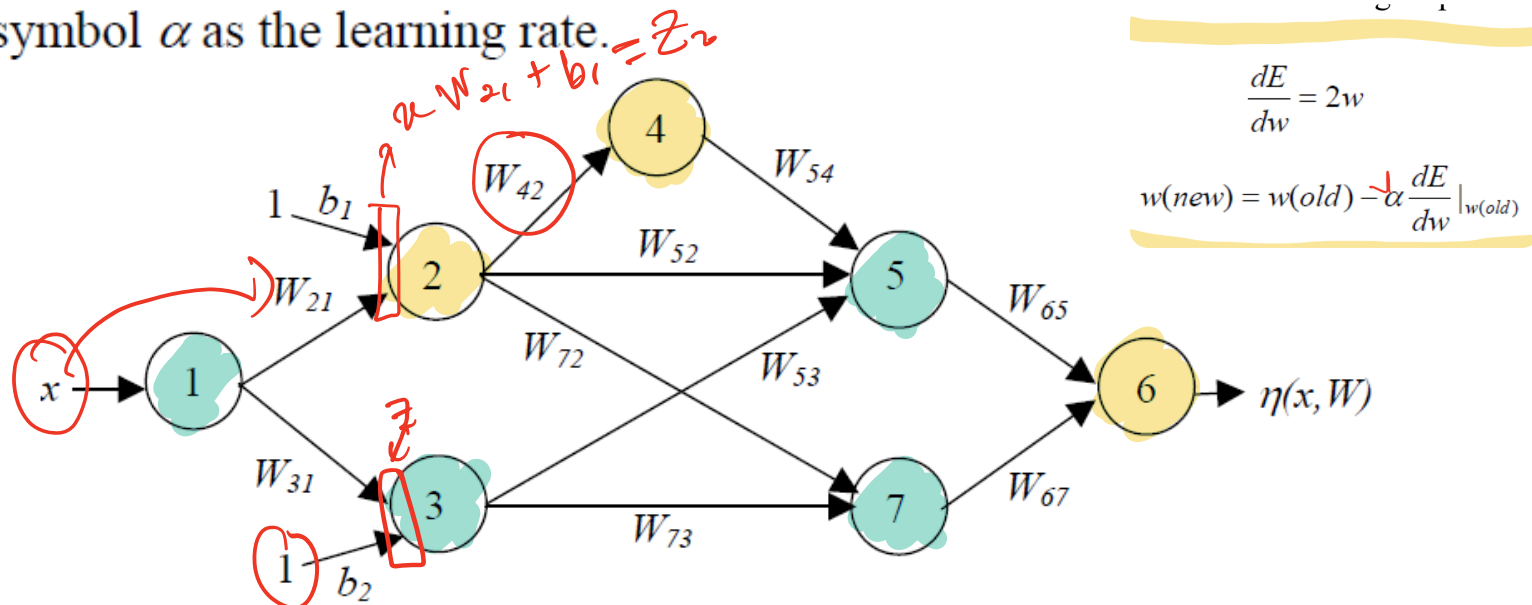


Example 5.4

For the ANN shown in Figure E3.3, if the error function

$$E^n(w) = \frac{1}{2} (\eta(x^n, w) - y^n)^2 \text{ and the gradient descent update rule is}$$

used, write the update rule for b_2 using backpropagation notation, if all of the even numbered neurons use linear activation function while the odd numbered neurons use sigmoid activation function. Use the symbol α as the learning rate.





Answer 5.4

$$\frac{dE}{dw} = 2w$$

$$w(\text{new}) = w(\text{old}) - \alpha \frac{dE}{dw} \Big|_{w(\text{old})}$$

- For gradient descent, the update rule for b_2 is given by:

$$b_{2(\tau+1)} = b_{2(\tau)} - \alpha \frac{\partial E}{\partial b_2} \Big|_{w_\tau}$$

- Using eq. 5.3.3(vi),

$$\frac{\partial E}{\partial b_2} = \delta_3 (1)$$

$$\frac{\partial E_i}{\partial w_{kj}} = \delta_k z_j$$

input

$$\frac{\partial E}{\partial w_{42}} = \delta_4 z_2$$

- From eq. 5.3.3(iii),

$$\delta_{1c} = \sum_{k=1}^2 [\delta_k w_{kj} z_j (1 - z_j)]$$

$$\delta_4 = z_4 (1 - z_4) [\delta_5 W_{54} + \delta_7 W_{74}]$$

- Next, define the rest of the terms contained in the expression above:

$\delta_5 = \delta_6 W_{65} z_5 (1 - z_5)$	$\delta_7 = \delta_6 W_{67} z_7 (1 - z_7)$	$\delta_6 = \frac{\partial E}{\partial a_6} = \eta(x^n, w) - y^n$
$z_3 = \frac{1}{1 + e^{-(W_{31}x + b_2)}}$	$z_5 = \frac{1}{1 + e^{-a_5}}$	$a_5 = W_{54}a_4 + W_{52}a_2 + W_{53}z_3$
$a_4 = W_{42}a_2$	$a_2 = b_1 + W_{21}x$	$z_7 = \frac{1}{1 + e^{-(W_{72}a_2 + W_{73}z_3)}}$

5.4

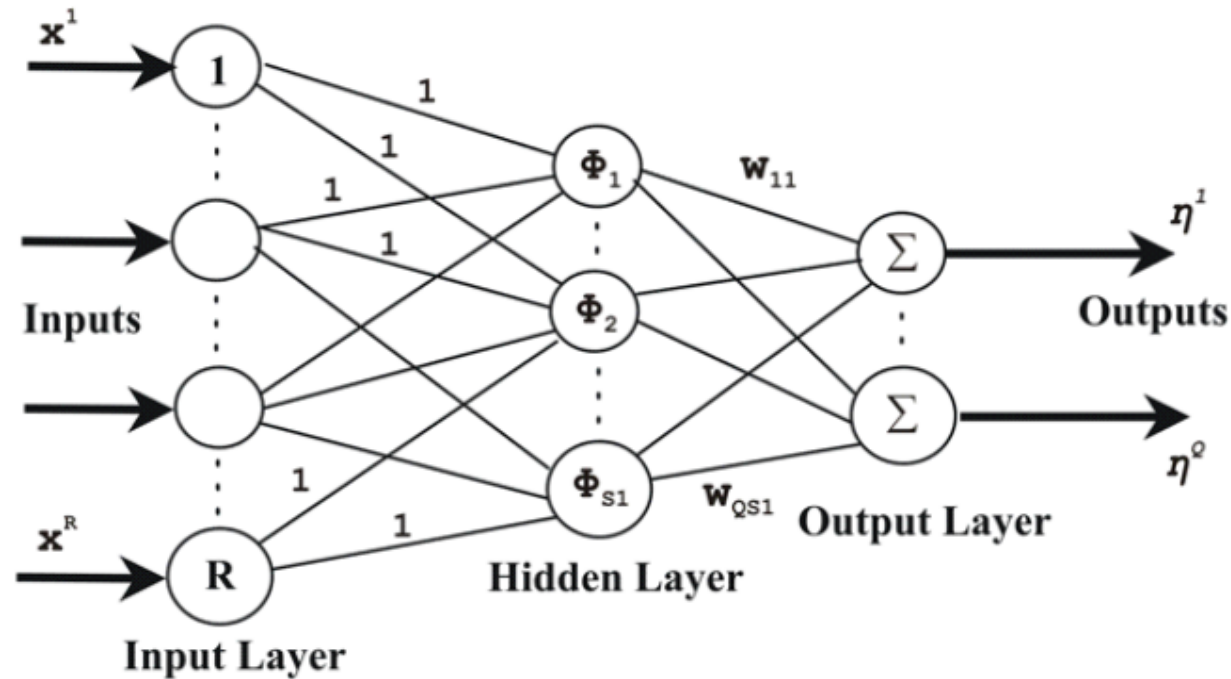
Radial Basis Function Neural Networks (RBFNN)



5.4.1 Introduction to RBFNN

- The radial basis functions were first used to design ANN in **1988 by Broomhead and Lowe**.
- RBF Artificial Neural Networks offer several advantages compared to MLP ANN, e.g.:
 1. They can be trained using fast 2 stages training algorithm without the need for time consuming non-linear optimization techniques.
 2. ANN RBF possesses the property of ‘best approximation’.
 3. The number of hidden neuron is updated automatically. It may require very less neuron for simple problems.

- The architecture:



- If the number of output, $Q = 1$, the output of the RBF NN in the figure is calculated according to

$$\eta(x, w) = \sum_{k=1}^{S1} w_{1k} \phi(\|x - c_k\|_2)$$

5.4.1 (i)

where $x \in \mathbb{R}^{R \times 1}$ is an input vector, $\phi(\cdot)$ is a basis function, $\|\cdot\|_2$ denotes the Euclidean norm, w_{Ik} are the weights in the output layer, SI is the number of neurons (and centers) in the hidden layer and $c_k \in \mathbb{R}^{R \times 1}$ are the RBF centers in the input vector space.

- Equation 7.4.1(i) can also be written as:

$$\eta(x, w) = \phi^T(x)w$$

5.4.1 (ii)

where

$$\phi^T(x) = [\phi_1(\|x - c_1\|) \quad \phi_2(\|x - c_2\|) \quad \cdots \quad \phi_{SI}(\|x - c_{SI}\|)]$$

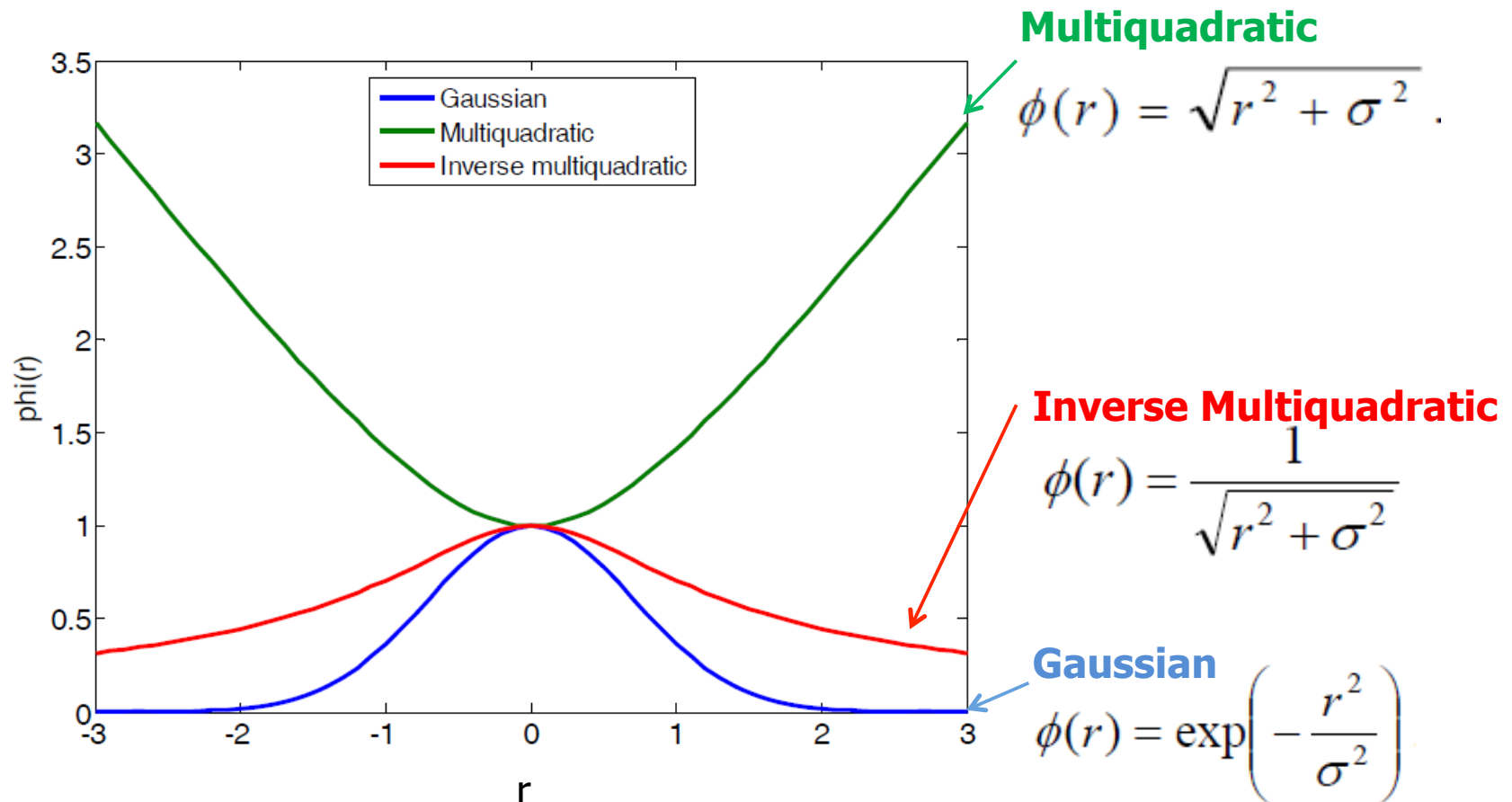
5.4.1 (iii)

and

$$w^T = [w_{11} \quad w_{12} \quad \cdots \quad w_{1SI}]$$

5.4.1 (iv)

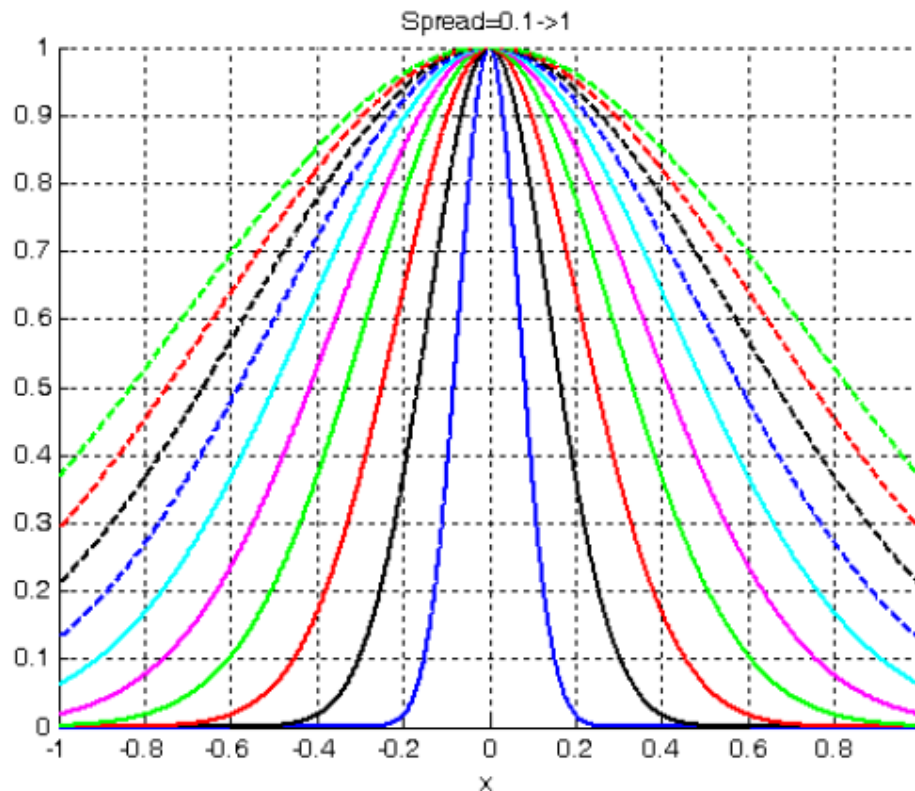
- The output of the neuron in a hidden layer is a nonlinear function of the distance. Some typical choices for the functional form are as follows:



Gaussian basis function:

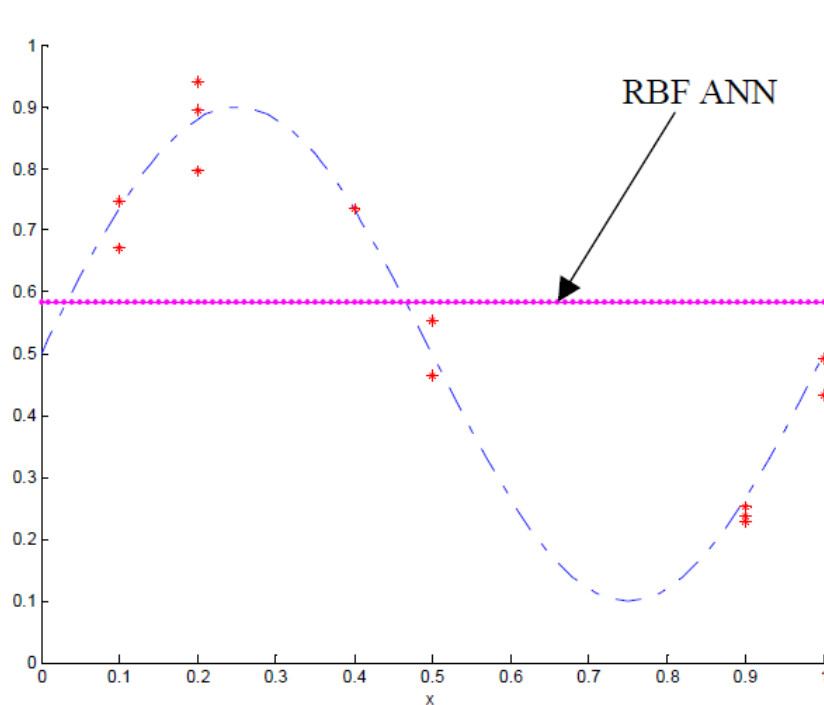
$$\phi(x) = e^{-x^2/\beta^2}$$

- It is most commonly used where the parameter β controls the “width” of the RBF and is commonly referred to as the **spread**

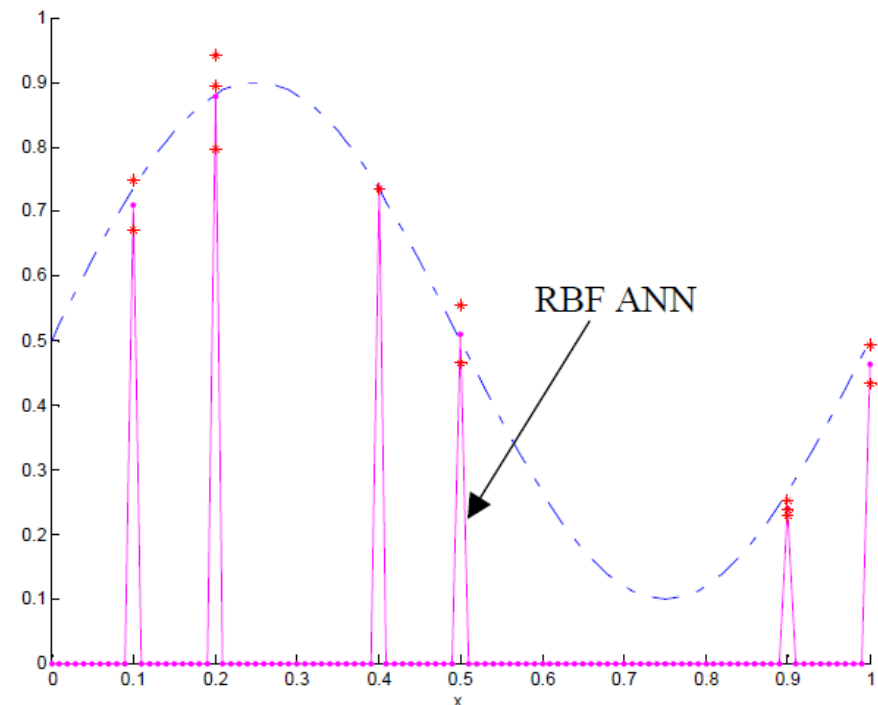


Gaussian basis function with different spread parameter

- The value of β that is too big or too small will cause a degradation in the performance of the RBF ANN.



β too big



β too small

7.4.2 Training the RBFNN

- Consider the following input-output training pairs:

$$\{(\text{input}, \text{output})\} = \{(x_i, y_i)\} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$$

- The RBFNN can be trained as follows:
 - Set the centers c_k using all of the input values or by using a clustering algorithm.
 - Set the spread parameter β using rule of thumb or by using an algorithm.
 - Write eq. 5.4.1(i) using vector matrix form:

$$\begin{bmatrix} \eta(x_1, w) \\ \vdots \\ \eta(x_N, w) \end{bmatrix} = \begin{bmatrix} \phi_1(x_1, c_1) & \cdots & \phi_{S1}(x_1, c_{S1}) \\ \vdots & \vdots & \vdots \\ \phi_1(x_N, c_1) & \cdots & \phi_{S1}(x_N, c_{S1}) \end{bmatrix} \begin{bmatrix} w_{11} \\ \vdots \\ w_{1S1} \end{bmatrix}$$

$$= \Phi w$$

5.4.2 (i)

4. Define the quadratic error

$$e(w) = (y - \Phi w)^T (y - \Phi w)$$

5.4.2 (ii)

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}.$$

5. Solve for optimum set of second layer weights, \hat{w} , using the least squares formula (eq. 7.4.3(i)).

7.4.3 Derivation of Least Squares

Formula for RBFNN

Expanding $e(w)$,

$$e(w) = y^T y - 2y^T \Phi w + w^T \Phi^T \Phi w$$

Differentiating with respect to w , and setting the derivative to zero, we find

$$\frac{de}{dw} = -2y^T \Phi + 2w^T \Phi^T \Phi = 0$$

$$\Phi^T \Phi w = \Phi^T y$$

$$\Rightarrow \hat{w} = (\Phi^T \Phi)^{-1} \Phi^T y$$

5.4.3 (i)

Example 5.5

Consider the Radial Basis Function (RBF) Artificial Neural Network (ANN) shown in Figure E4.1. The basis function ϕ , is given by

$$\phi_k(\|x - c_k\|_2) = e^{-\frac{\|x - c_k\|_2^2}{\beta^2}} \quad (\text{E5.5.1})$$

where

x is the input,
 c_k are the basis function centres,
 $\|\cdot\|_2$ denotes the euclidean distance,
and β is the spread parameter.

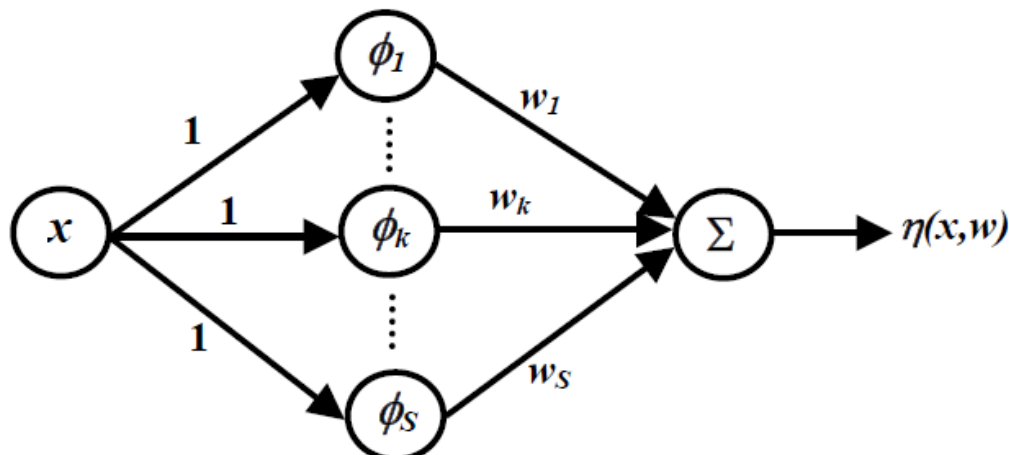


Figure E5.5

- b. Assuming that a training data set, D , was formed from the experimental data given by:

$$D = \{(x_i, y_i)\} = \{(0.1, 0.74), (0.2, 0.87), (0.9, 0.25), (1.0, 0.42)\} \quad \text{(E5.5.2)}$$

- i. Write the complete expression for the RBF output, $\eta(x, w)$, using Figure E7.5 equations E7.5.1 and E7.5.2 and using $\beta=1$ if all of the training input vectors are chosen as the basis function centers.
- ii. For the case when the quadratic error is given by



$$E = \frac{1}{2} \sum_{i=1}^N [\eta(x_i, w) - y_i]^2$$

is used, the optimum values for the second layer weights can be found using the least squares formula given by

$$\hat{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_S \end{bmatrix} = (\Phi^T \Phi)^{-1} \Phi^T \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}$$

Write the matrix Φ complete with all its entries (elements) in this case. Write the matrix in general form first before computing the numerical values of its entries. Use all the input training data as the RBF centers.

Answer 5.5

b.

i.

$$\eta(x, w) = w_1 e^{-\|x-0.1\|_2^2} + w_2 e^{-\|x-0.2\|_2^2} + w_3 e^{-\|x-0.9\|_2^2} + w_4 e^{-\|x-1.0\|_2^2}$$

ii.

$$\Phi = \begin{bmatrix} \phi(x_1, c_1) & \phi(x_1, c_2) & \phi(x_1, c_3) & \phi(x_1, c_4) \\ \phi(x_2, c_1) & \phi(x_2, c_2) & \phi(x_2, c_3) & \phi(x_2, c_4) \\ \vdots & \vdots & \vdots & \vdots \\ \phi(x_4, c_1) & \phi(x_4, c_2) & \phi(x_4, c_3) & \phi(x_4, c_4) \end{bmatrix}$$

$$\Phi = \begin{bmatrix} e^{-\|0.1-0.1\|_2^2} & e^{-\|0.1-0.2\|_2^2} & e^{-\|0.1-0.9\|_2^2} & e^{-\|0.1-1.0\|_2^2} \\ e^{-\|0.2-0.1\|_2^2} & e^{-\|0.2-0.2\|_2^2} & e^{-\|0.2-0.9\|_2^2} & e^{-\|0.2-1.0\|_2^2} \\ e^{-\|0.9-0.1\|_2^2} & e^{-\|0.9-0.2\|_2^2} & e^{-\|0.9-0.9\|_2^2} & e^{-\|0.9-1.0\|_2^2} \\ e^{-\|1.0-0.1\|_2^2} & e^{-\|1.0-0.2\|_2^2} & e^{-\|1.0-0.9\|_2^2} & e^{-\|1.0-1.0\|_2^2} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0.99 & 0.5273 & 0.4449 \\ 0.99 & 1 & 0.6126 & 0.5273 \\ 0.5273 & 0.6126 & 1 & 0.99 \\ 0.4449 & 0.5273 & 0.99 & 1 \end{bmatrix}$$

5.4.4 Choosing the centers

- The centers c_k are defined points that are assumed to perform an adequate sampling of the input space.
- Common practice is to select a relatively **large number** of input vectors as the centers to ensure an adequate input space sampling.
- After the network has been trained, some of the centers may be removed in a systematic manner without significant degradation of the network mapping performance.
- The simplest procedure for selecting the basis function centres c_k is to set them **equal to the input vectors** or a **random subset** of the input vectors from the training set.
- Clearly this is not an optimal procedure since it leads to the use of an **unnecessarily large number** of basis functions in order to achieve adequate performance on the training data.

***K*-means clustering:**

- In K-means clustering, the number of ‘desired centres’, K , must be decided in advance.
- One simple way of choosing the value of K is to set it equal to a **fraction of total training data** samples.
- The K-means algorithm is as follows:
 1. Assign the input data to random K sets.
 2. Compute the mean of each set.
 3. Reassign each point to a new set according to which is the nearest mean vector.
 4. Recompute the mean of each set.
 5. Repeat steps 3 and 4 until there is no further change in the grouping of data points.
 6. The mean of the sets will be the RBF centers.

Example 5.6

- Use K-means clustering to find the RBF centers when the input output training data is given by:

$$D = \{(x, y) = (1,10), (3,3), (10,5), (11,2), (5,5)\}$$

- Use $K=2$.

End of Chapter 5(3)