Machine Learning (IS ZC464) Session 11:

Artificial Neural Networks(ANN) – Computing Weights using Gradient Descent Algorithm, Learning using ANN.

Review

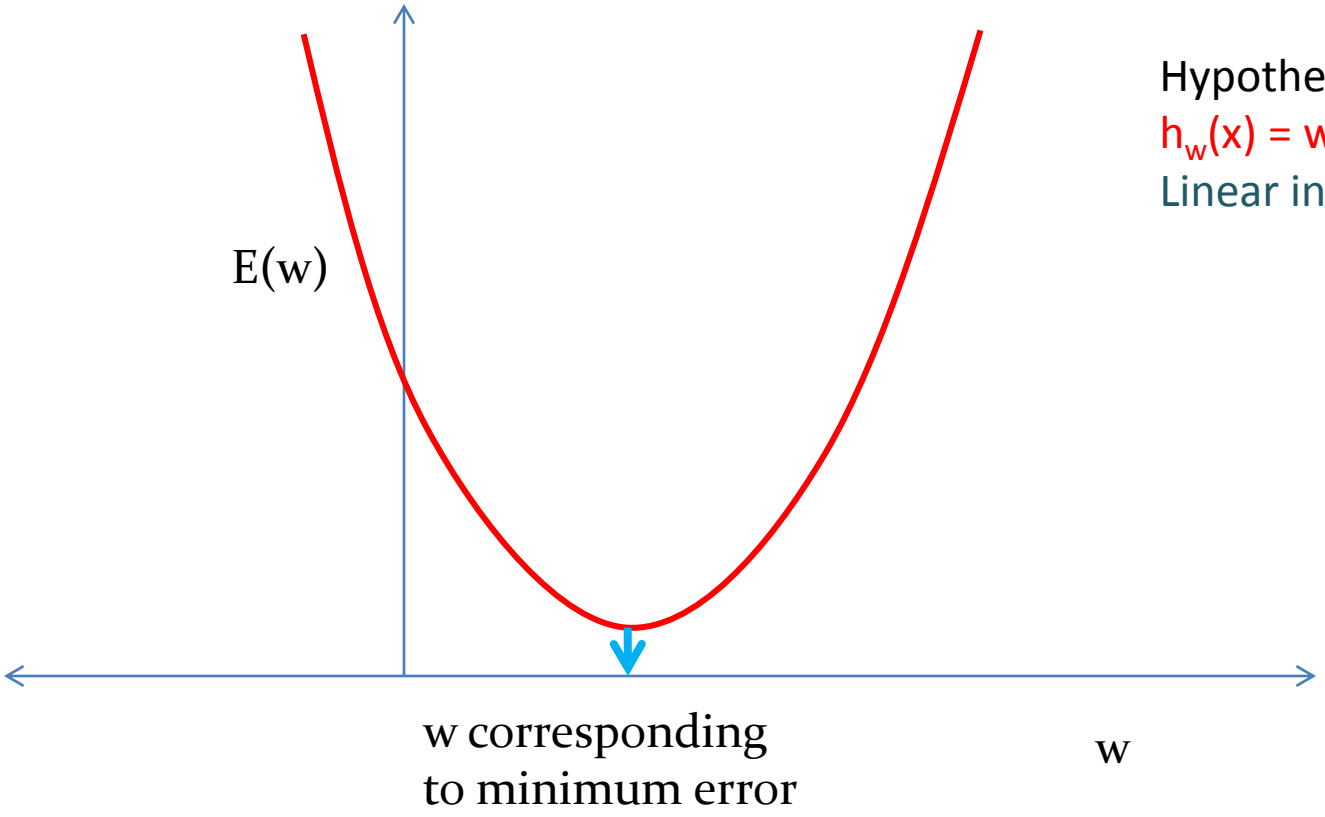
- Plotting 2-D feature vectors with designated class labels on 2-D cartesian plane
- Observing whether the data is linearly separable or not.
- How many decision lines are required to discriminate between class regions?
- Number of neurons and number of decision lines relationship
- Output modelling

Error surface for Neural Network based classification

- Consider m observations $<x^1,y^1>$, $<x^2,y^2>$, ….$<x^m,y^m>$.
- An hypothesis $h_w(x)$ that approximates the function that fits best to the given values of y
- There is likely to be some error corresponding to each observation (say i).
- The magnitude of such error is $y^i - h_w(x^i)$
- Objective is to find such w that minimizes the sum of squares of errors

$$E_{min}(w) = \text{Minimize}_w \sum_i (y^i - h_w(x^i))^2$$

Plotting error when y=f(x)

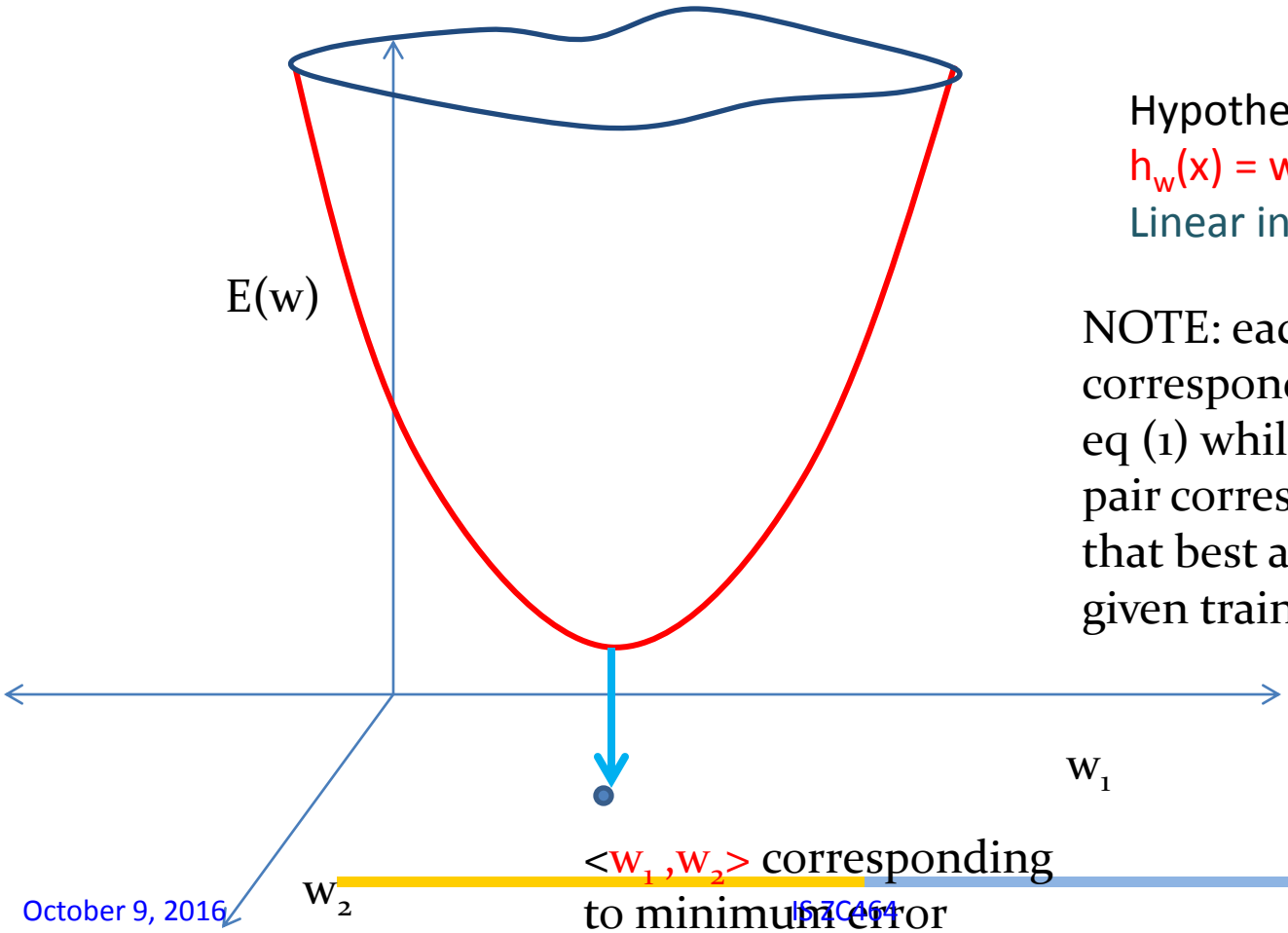


Hypothesis function

$h_w(x) = wx$

Linear in one variable

Plotting error when $y=f(x_1,x_2)$



Hypothesis function

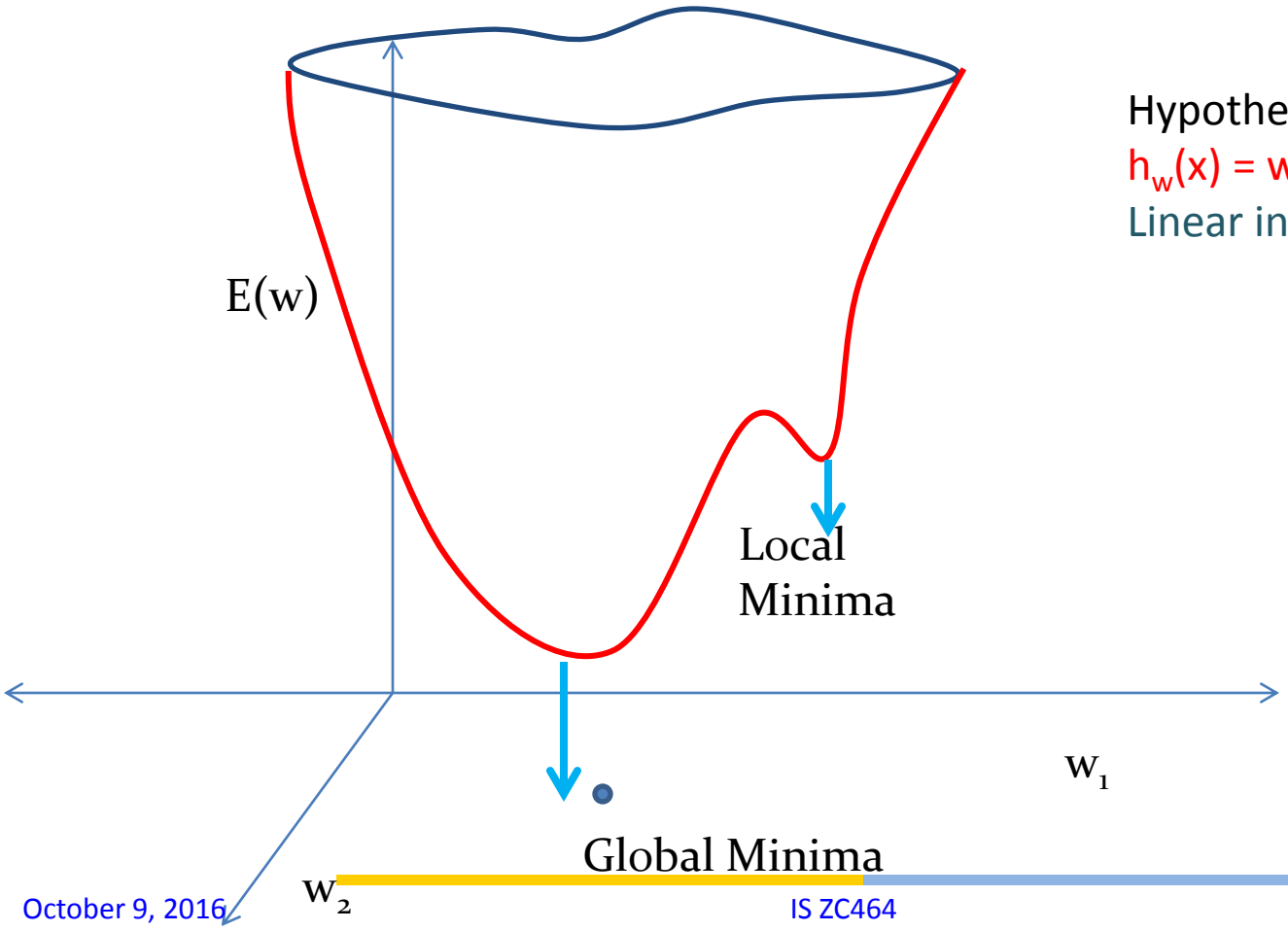
$h_w(x) = w_1x_1 + w_2x_2$ .....(1)

Linear in two variables

NOTE: each pair $< w_1 , w_2 >$ corresponds to a line given by eq (1) while only one such pair corresponds to the line that best approximates the given training data

$<w_1 , w_2>$ corresponding to minimum error

Plotting error when $y=f(x_1,x_2)$

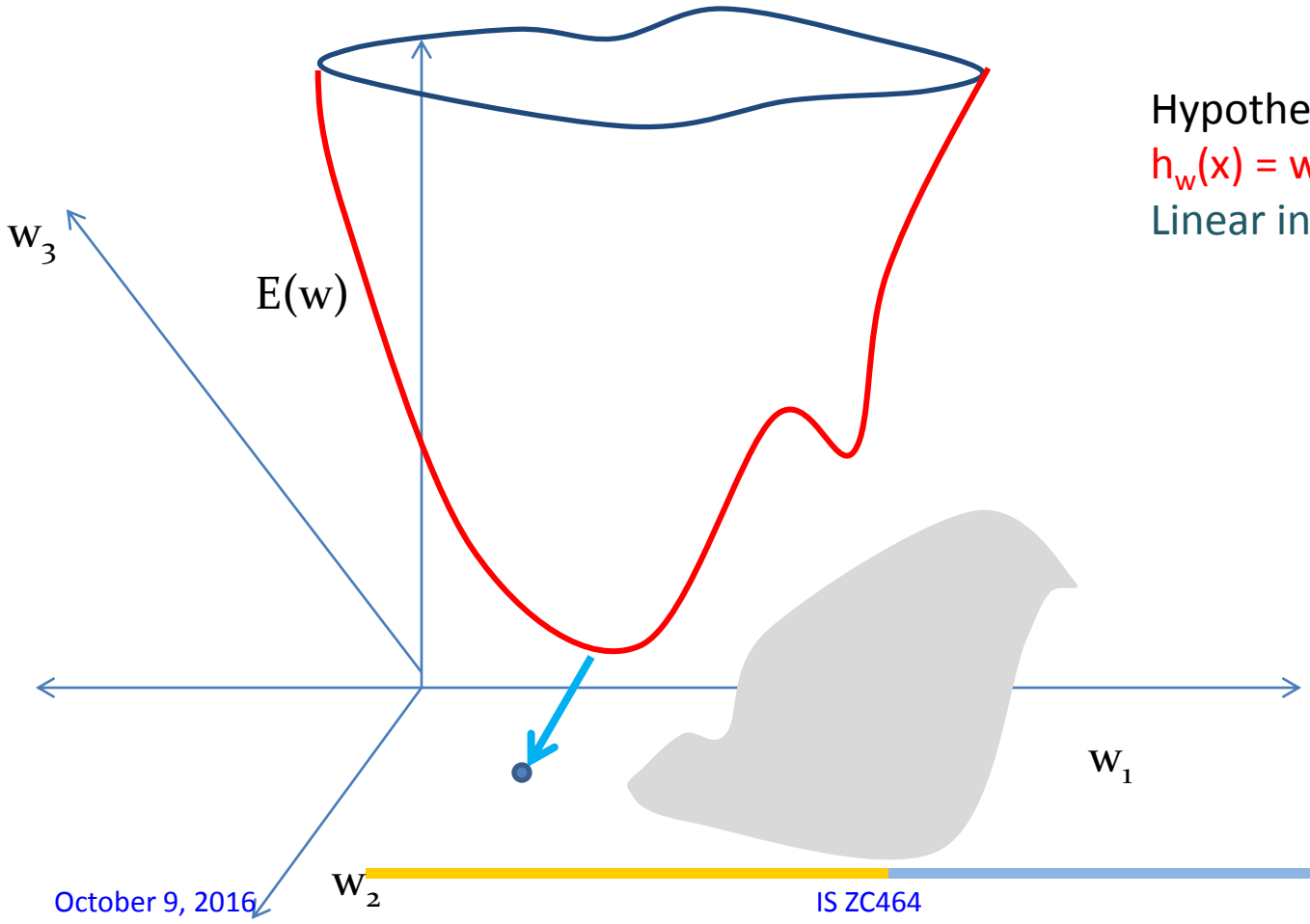


Hypothesis function

$h_w(x) = w_1x_1 + w_2x_2$ .....(1)

Linear in two variables

Difficult to visualize when $y=f(x_1,x_2, x_3)$

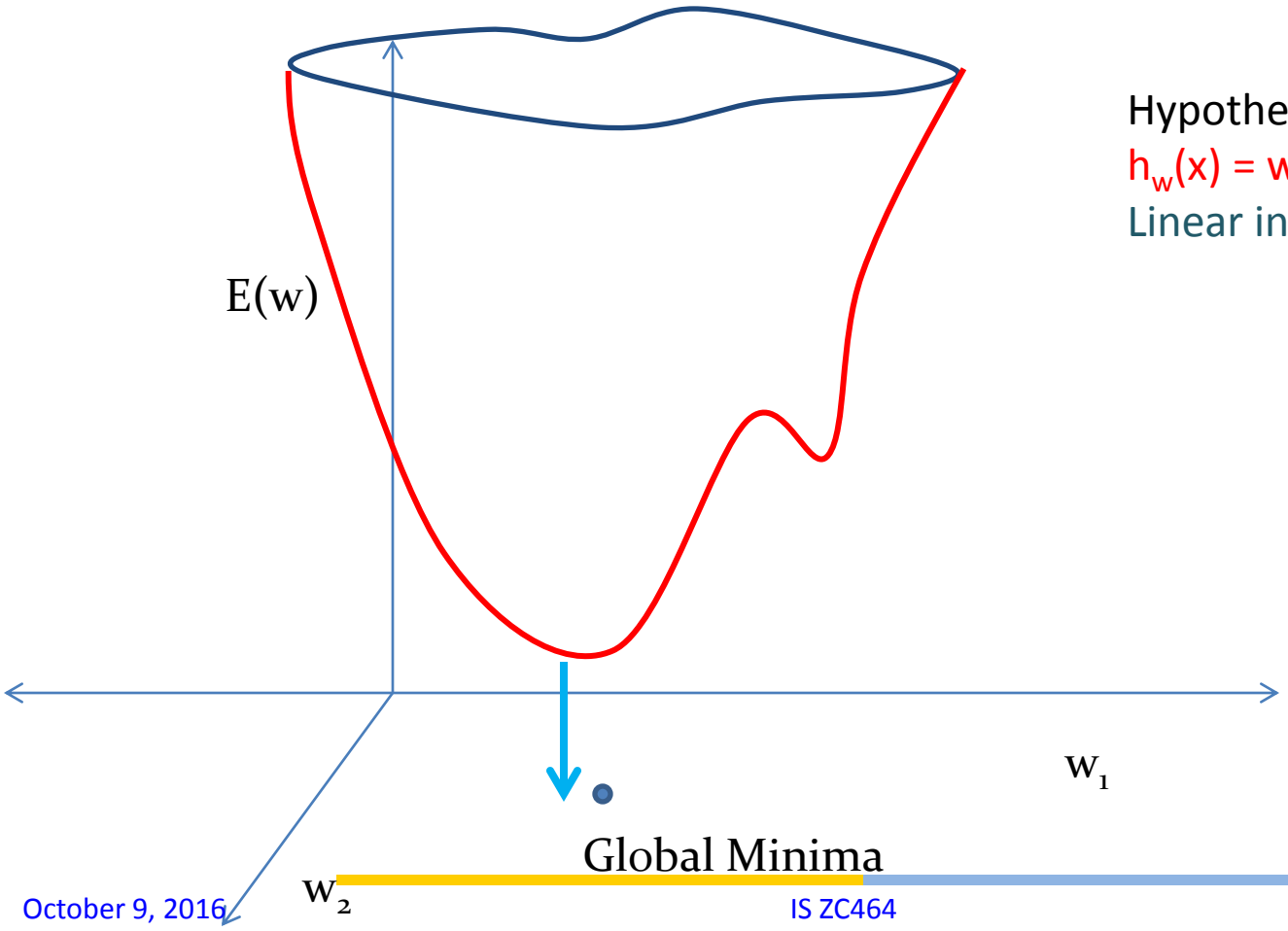


Hypothesis function

$h_w(x) = w_1x_1 + w_2x_2 + w_3x_3$

Linear in three variables

We will visualize in 2D but will extend the concept to n dimensions

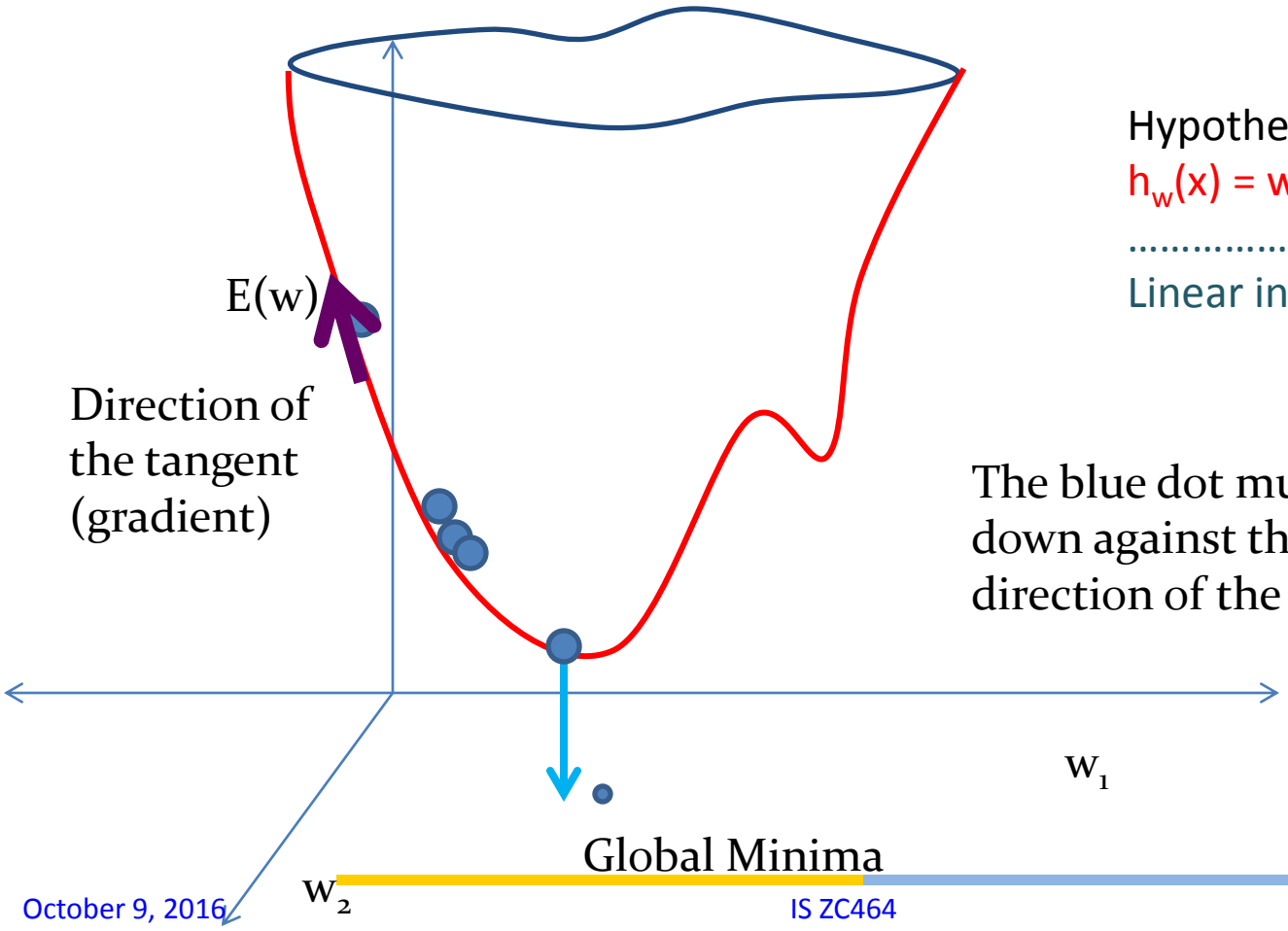


Hypothesis function

$h_w(x) = w_1x_1 + w_2x_2 + \ldots w_nx_n$

Linear in n variables

Initial weight



Hypothesis function

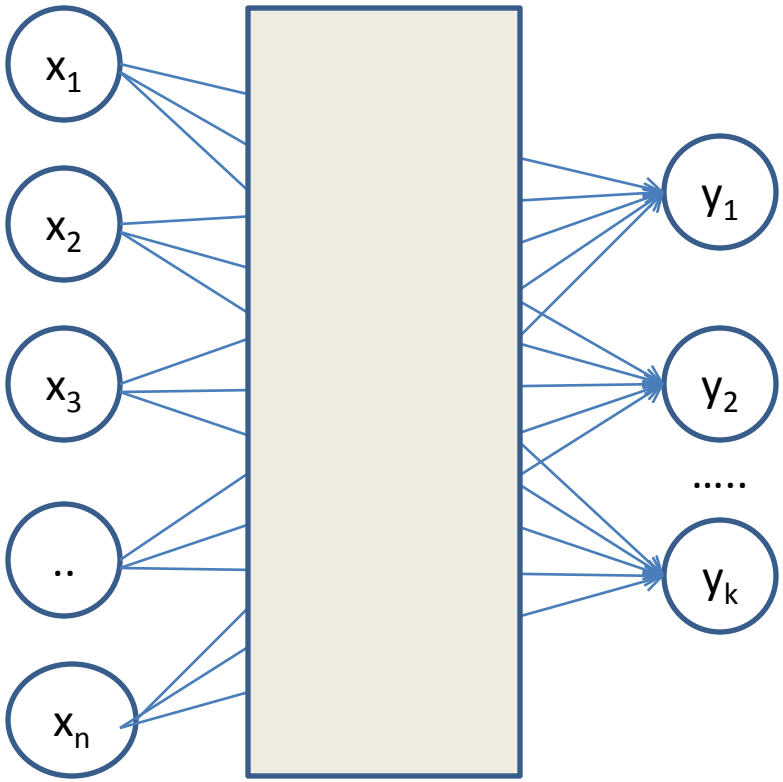
$h_w(x) = w_1x_1 + w_2x_2 + \ldots . w_nx_n$

.....equation 2

Linear in n variables

Visualization of n-dimensional data

- $y_1 = f_1(x_1, x_2, \ldots . x_n)$
- $y_2 = f_2(x_1, x_2, \ldots . x_n)$
- $y_3 = f_3(x_1, x_2, \ldots . x_n)$
- $y_4 = f_4(x_1, x_2, \ldots . x_n)$
- .....
- $y_k = f_k(x_1, x_2, \ldots . x_n)$

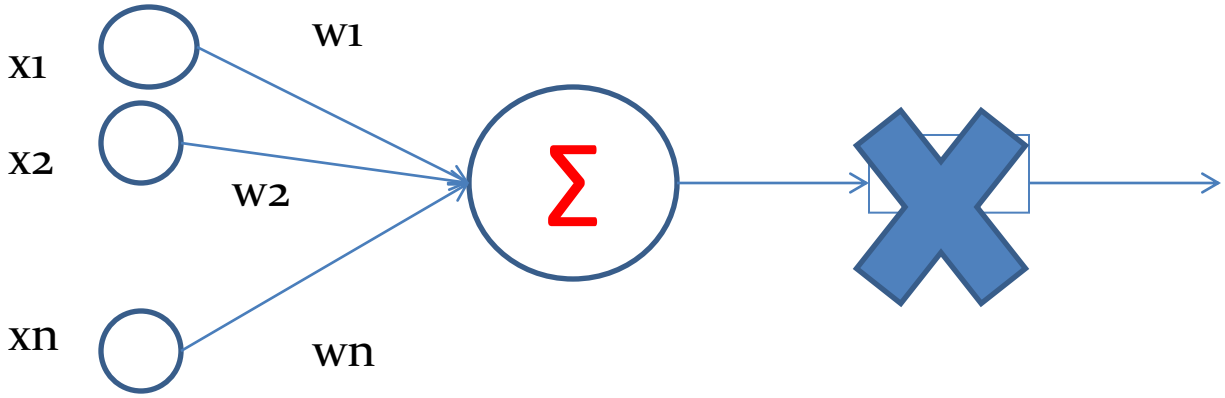


Computing gradient for the simple one perceptron model of neural network

Output of the neural network

$h_w(x) = w_1x_1 + w_2x_2+....w_nx_n$

Human supervised output = y



Error = $y - h_w(x)$

Understanding symbols

- Assume that the feature vector is $(x_1,x_2,x_3,\ldots x_n)$
- There are m observations whose feature vectors are written as follows

$$(x^i_1 ,x^i_2,x^i_3,\ldots x^i_n)$$

For i = 1,2,3,…m

Note: Here the superscript ‘i’ represents the ‘i’th observation and NOT the power of x.

- Let y^i be the output (human supervised)
- Let T_i be the error (note the use of subscript ‘i’ instead of superscript—That is just my way of representing)

Gradient Descent

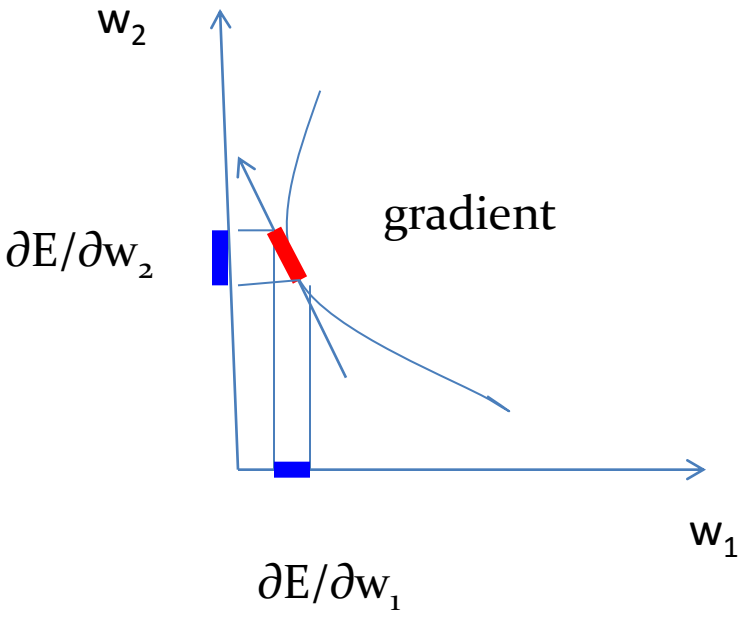
- This technique is used to reduce the squared error by calculating the partial derivative of E with respect to each weight.
- $E(w) = \sum_i T_i^2 = \sum_i (y^i - h_w(x^i))^2$

Replace $h_w(x)$ with the expression given in eq 2

$E(w) = (½)\sum_i (y - (w_1 x^i_1 + w_2 x^i_2 + \ldots w_n x^i_n))^2$

- Observe that E is a function of $w_1, w_2, \ldots w_n$
- Note that the effort is towards finding the equation of a line in n-dimensional space that best fits n dimensional data.
- Normalization with ½ is for computational convenience

Computing gradient



Computing Gradient

$E(w) = \sum_i T_i^2$

$= \sum_i (y^i - h_w(x^i))^2$

where T_i is the error term for the i^{th} observation and is given by the difference between the desired output (y^i) value and the estimated value ($h_w(x^i)$)of the output

$$T_i = y^i - h_w(x^i)$$

$h_w(x^i)$ is the hypothesis function given by

$$h_w(x^i) = w_1 x^i_1 + w_2 x^i_2 + \ldots . w_n x^i_n$$

Observe: E is a function of w.

Note: Here the superscript 'i' represents the 'i'th observation and NOT the power of x.

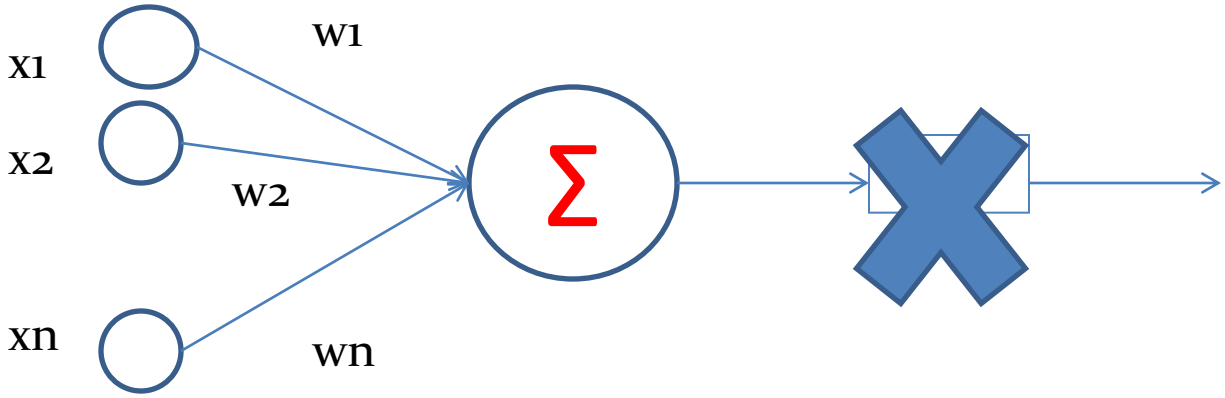
Computing gradient for the simple one perceptron model of neural network

Output of the neural network

$h_w(x) = g(w_1x_1 + w_2x_2 + \ldots w_nx_n)$

Where g is the activation function

Human supervised output = y



Error = $y - g(h_w(x))$

Computing Gradient

$E(w) = \sum_i T_i^2$

$= \sum_i (y^i - g(h_w(x^i)))^2$

where T_i is the error term for the i^{th} observation and is given by the difference between the desired output (y^i) value and the estimated value ($h_w(x^i)$)of the output

$T_i = y^i - g(h_w(x^i))$

$h_w(x^i)$ is the hypothesis function given by

$h_w(x^i) = w_1 x^i_1 + w_2 x^i_2 + \ldots . w_n x^i_n$

Observe: E is a function of w.

Note: Here the superscript 'i' represents the 'i'th observation and NOT the power of x.

Observe

- E is the function of T_i
- Ti is the function of g (assuming y as constant)
- g is the function of h
- h is the function of w

- Chain rule of Differentiation

$$\partial E/\partial w_k = \sum_i \partial E/\partial T_i * \partial T_i/\partial g * \partial g/\partial h * \partial h/\partial w_k$$

Equation 1

Observe

Since

$$E(w) = \sum_i T_i^2$$

$$\partial E/\partial T_i = 2*T_i$$

Chain rule of Differentiation

$$\partial E/\partial w_k = 2*\sum_i T_i * \partial T_i/\partial g * \partial g/\partial h * \partial h/\partial w_k$$

Also since

$$T_i = y^i - g(h_w(x^i))$$

Therefore

$$\partial T_i/\partial g = 0 - 1 = -1$$

Equation 2

Working with derivatives

Equation 2 now becomes

$$\partial E/\partial w_k = 2 * \sum_i T_i * (-1) * \partial g/\partial h * \partial h/\partial w_k$$

Also since

$$\partial g/\partial h = \partial\, g(h_w(x^i))\ /\partial h = g'$$

Equation 3

And

$$h_w(x^i) = w_1 x^i_1 + w_2 x^i_2 + \ldots . w_n x^i_n$$

Therefore

$$\partial h/\partial w_k = x^i_k$$

Hence equation 3 is simplified as

$$\partial E/\partial w_k = -2 * \sum_i T_i * g' * x^i_k$$

Equation 4

Computing gradient in the direction of w_k

- Substitute expression for T_i in equation 4

$$\partial E/\partial w_k = -2 * \sum_i (y^i - g(h_w(x^i))) * g' * x^i_k$$

Equation 5

- The Weight update in the direction of w_k

$$\Delta w_k = -2 * \sum_i (y^i - g(h_w(x^i))) * g' * x^i_k$$

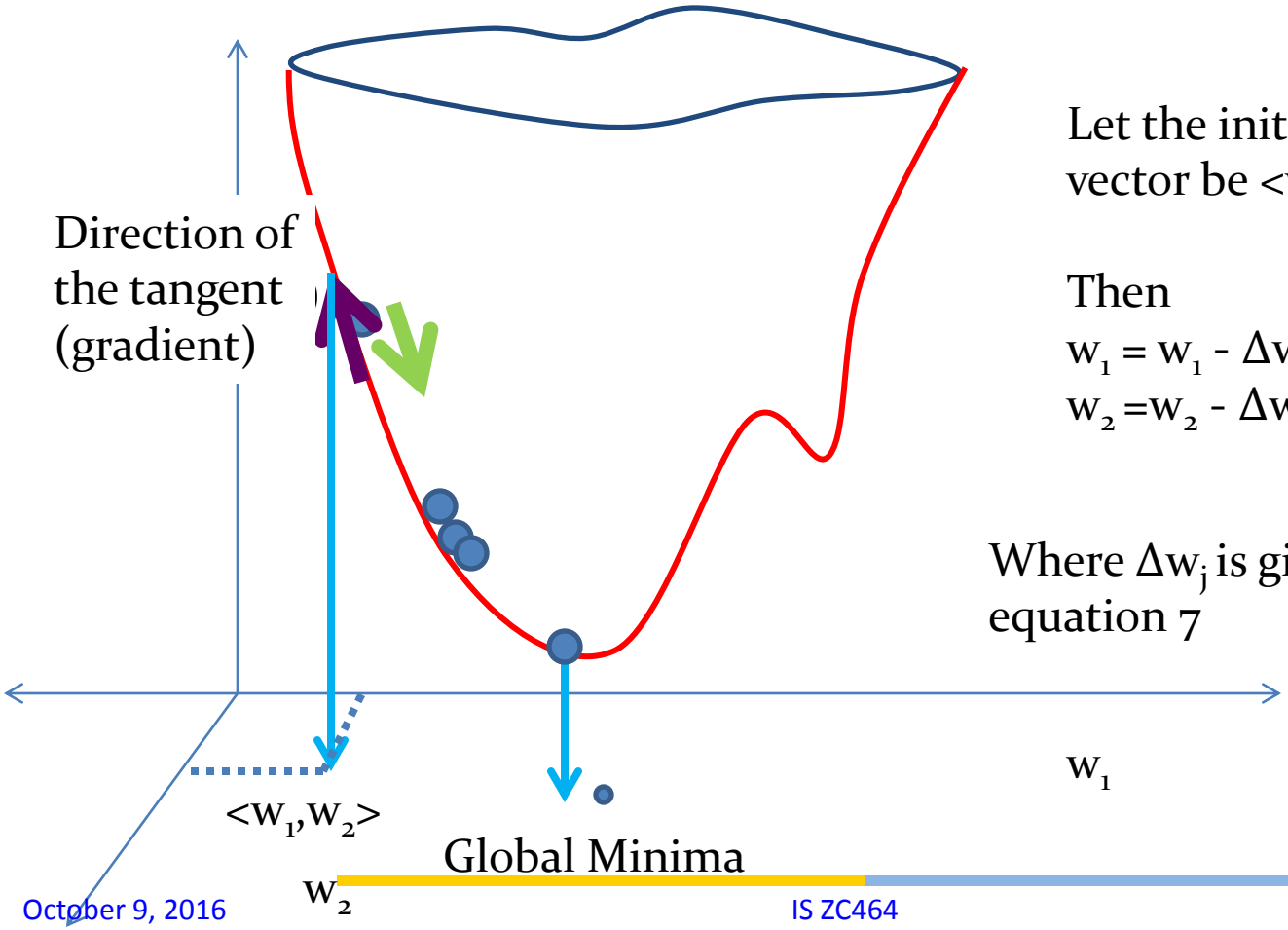
Equation 6

Where 2 can be dropped to bring normalization.

$$\Delta w_k = - \sum_i (y^i - g(h_w(x^i))) * g' * x^i_k$$

Equation 7

Delta Learning: Modification of the Initial weight



Let the initial weight vector be $<w_1,w_2>$

Then

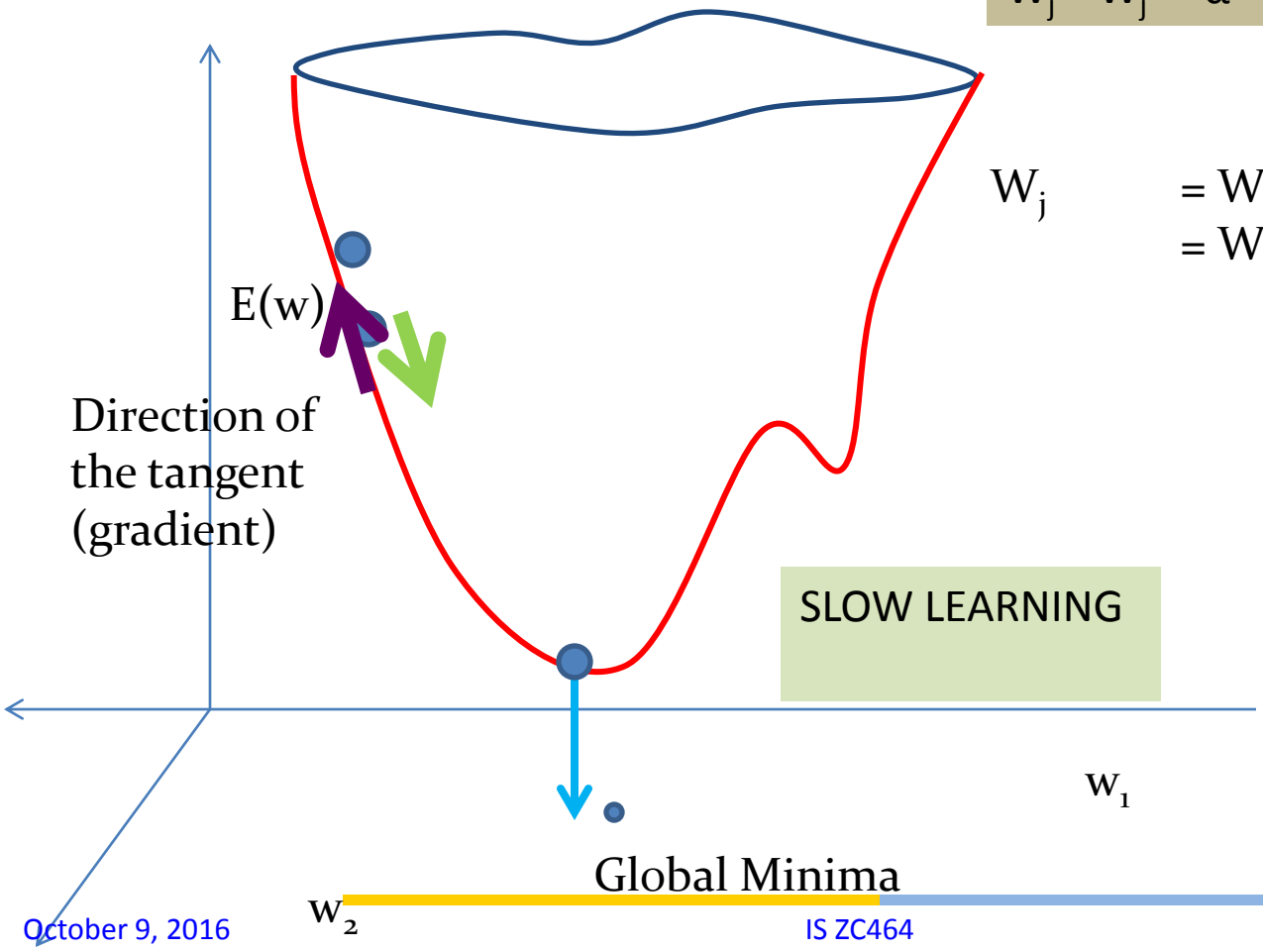
$w_1 = w_1 - \Delta w_1$

$w_2 = w_2 - \Delta w_2$

Where Δw_j is given by equation 7

Learning rate: fast or slow learning

$$W_j = W_j + \alpha * \sum_i (y^i - g(h_w(x^i))) * g' * x^i_k$$



$$W_j = W_j - \Delta w_j$$
$$= W_j + \sum_i T_i * g' * x^i_k$$

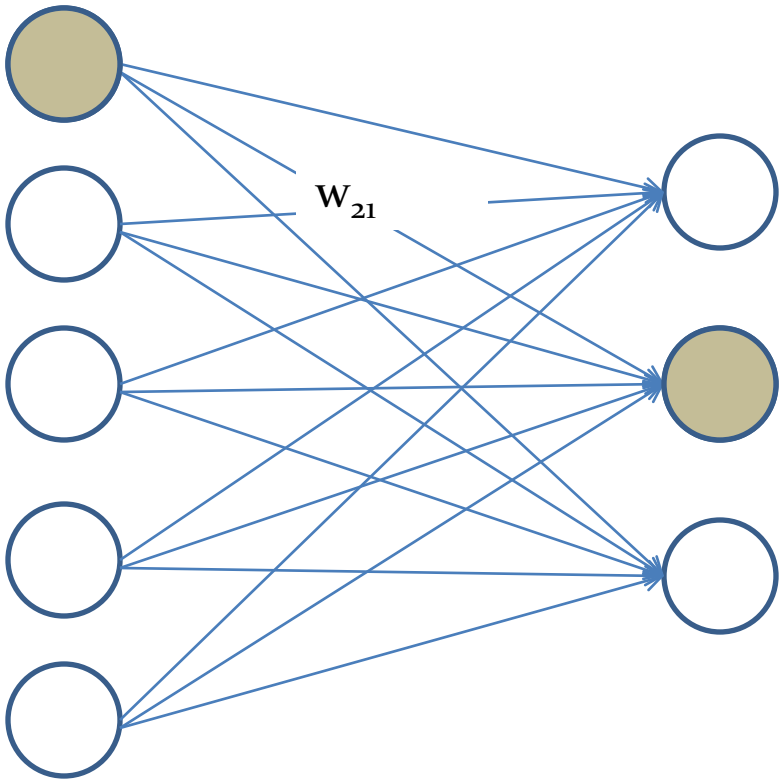
α

α ranges from 0 to 1
If it is 0 , there is no learning
If it is large the learning rate is high

Multilayer Feed Forward neural network

- These represent the class of networks which approximate the complex functions.
- The network has one or more hidden layers.
- The neuron ‘i’ of layer ‘L’ is connected by a synaptic weight w_{ki} to the ‘k’th neuron of layer ‘L+1’

Weight Terminology

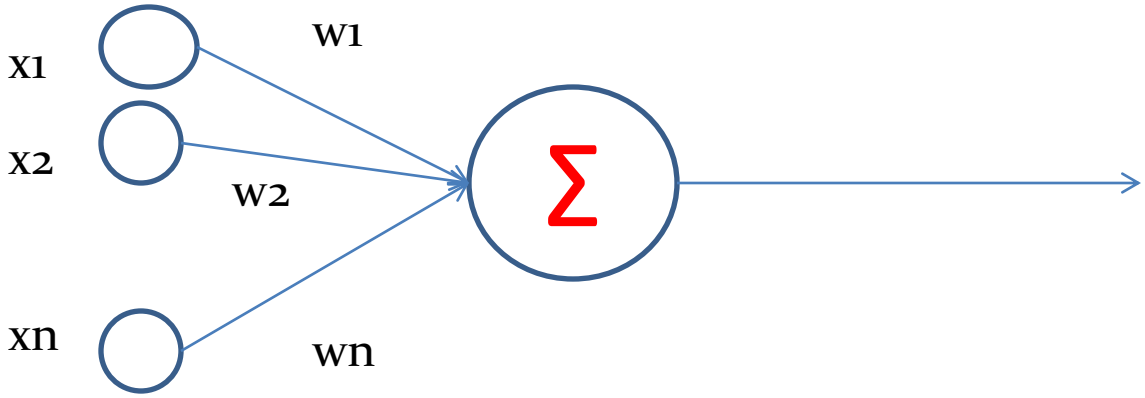


Feed Forward and Back propagation

- In feed forward neural network, the weights are computed on the basis of the input propagating through neurons in the forward direction. In this no neuron receives the modified input.
- In back propagation neural network, the processed input is cycled again through the previous layer neurons and the weights are modified.

Feed Forward Neural Networks

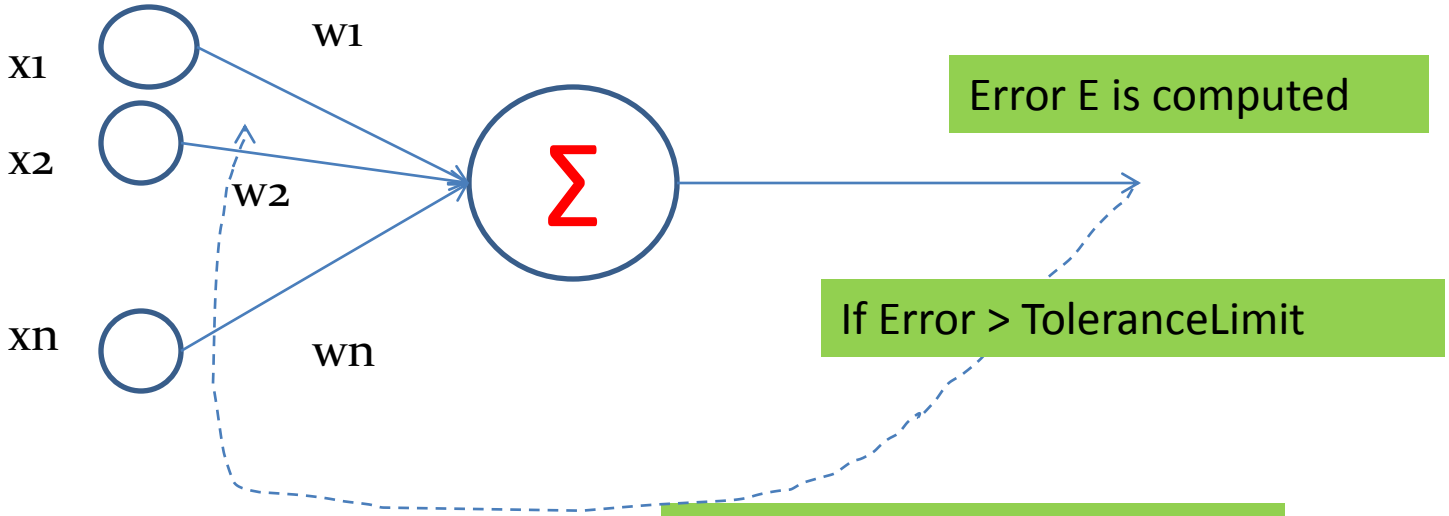
Weights learning is one way



Back Propagation Neural Networks

Weights learning is cyclic

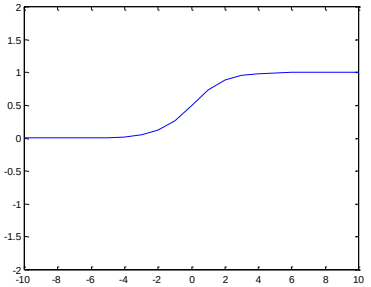
If Error <= ToleranceLimit
Then
terminate



Activation function should be differentiable for Delta Learning

– $\Delta w_k = - \sum_i (y^i - g(h_w(x^i))) * g' * x^i_k$

- The function g' is 0 if g is not differentiable
- Example Activation functions
- Step Function : Not differentiable
- Sigmoid Function : Differentiable



$$y = \frac{1}{1 + \exp(-x)}$$

Gradient Descent Algorithm

1. Initialize weights in the n-dimensional space randomly.
2. Compute error E.
3. Define error tolerance limit L.
4. While (E > L)
   - Modify weights W according to delta rule
   - Compute error E with the modified weights and the given input.

Terminology used in text book

	Used in the slides here	Used in book by Mitchell (Chapter 4)
Set of Training samples	Input : vector x^i : i = 1,2,…m output: y^i : i = 1,2,…m	D is the set of training samples $d \in D$ Input : vector x_d : $d \in D$ output : t_d : $d \in D$
Target (Known-supervised)	y^i	t_d
Input feature vector	$x^i = <x^i_1, x^i_2, x^i_3, \ldots x^i_n>$	$x_d = <x_{d1}, x_{d2}, x_{d3}, \ldots x_{dn}>$
Output-predicted by ANN	$h_w(x^i)$	o_d
error	$y^i - h_w(x^i)$	$t_d - o_d$

What to do with the weights (W) obtained using Gradient Descent Algorithm

- Let W = $<w_1,w_2,w_3,....w_n>$ [Known Now as a result of training]
- Have a new feature vector is x = $(x_1,x_2,x_3,...x_n)$ corresponding to the sample not yet seen by the machine (known as test vector)
- Compute output y as follows
- $y = h_w(x) = w_1x_1 + w_2x_2+....w_nx_n$
- This is the identification of the output **[Machine has learned]**