Least Mean Square algorithm for Single Layer Network

Outline

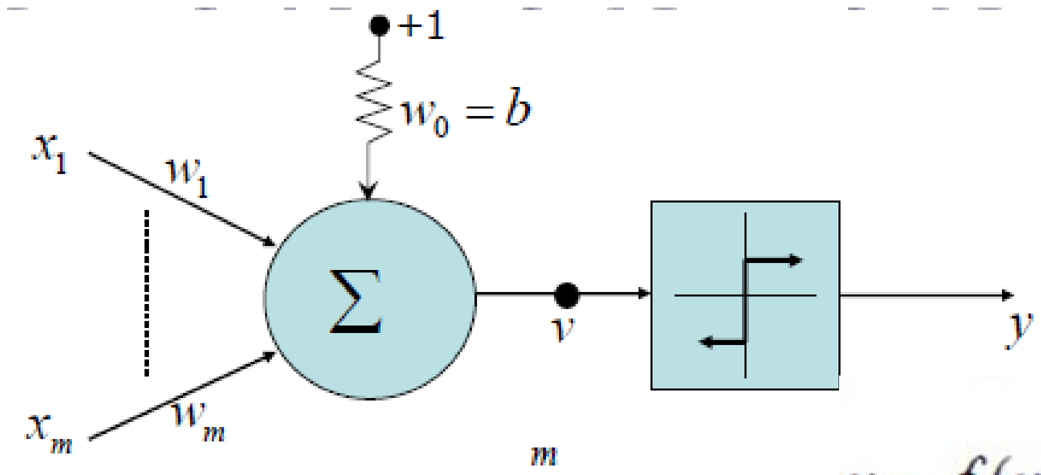
Perceptron Learning rule

- Adaline (**Ada**ptive **Line**ar Neuron) Networks
- Derivation of the LMS algorithm
- Example
- Limitation of Adaline

Perceptron

- **The Perceptron** is presented by Frank Rosenblatt (1958, 1962)
- **The Perceptron**, is a feedforward neural network with no hidden neurons. The goal of the operation of the perceptron is to learn a given transformation using learning samples with input x and corresponding output y = f (x).
- It uses the **hard limit transfer function** as the activation of the output neuron. Therefore the perceptron output is limited to either 1 or –1.

Perceptron network Architecture



$$v = \sum_{i=0}^{m} x_i w_i$$

$$y = f(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v < 0 \end{cases}$$

- The update of the weights at iteration n+1 is:

$$W_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

Since:

$$\Delta w_{kj}(n) = \eta(d_k - y_k)x_j(n)$$

Limit of Perceptron Learning rule

- If there is no separating hyperplane, the perceptron will never classify the samples 100% correctly.
- But there is nothing from trying. So we need to add something to stop the training, like:
  - Put a **limit** on the number of iterations, so that the algorithm will terminate even if the sample set is not linearly separable.
  - Include an **error bound**. The algorithm can stop as soon as the portion of misclassified samples is less than this bound. This ideal is developed in the Adaline training algorithm.

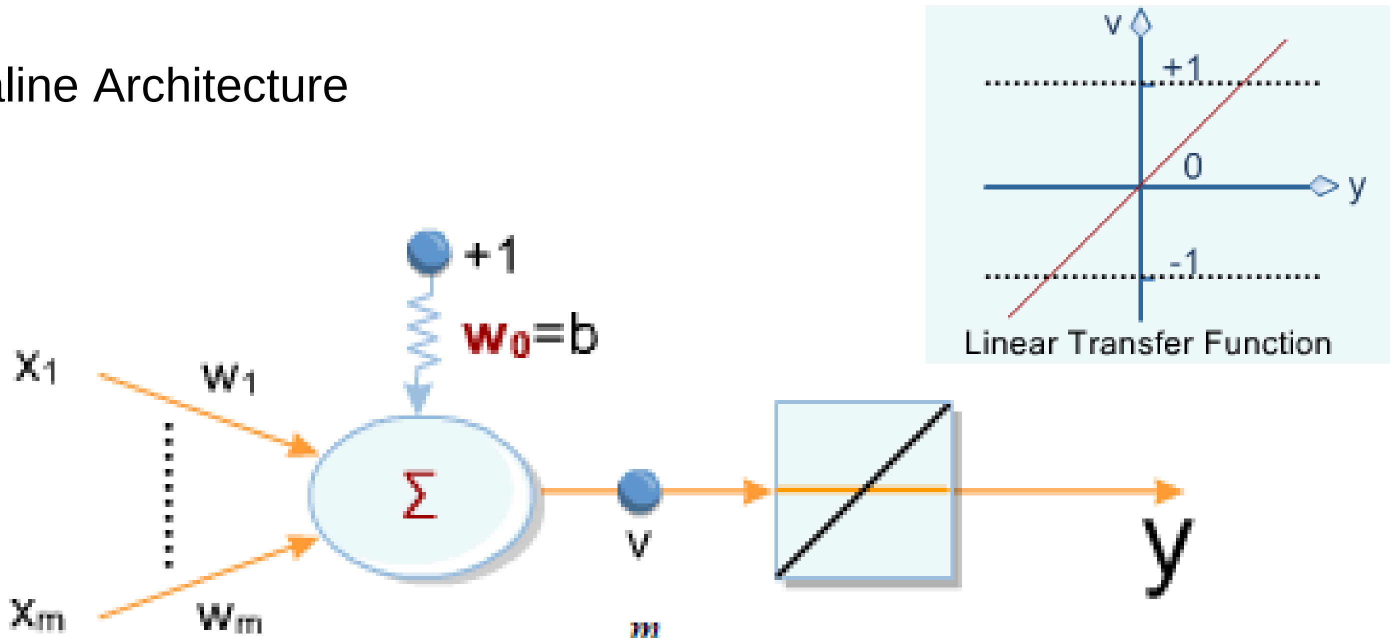
Error Correcting Learning

- The objective of this learning is to start from an arbitrary point error and then move toward a global minimum error, in a step-by-step fashion.
  - The arbitrary point error determined by the initial values assigned to the synaptic weights.
  - It is closed-loop feedback learning.
- Examples of error-correction learning:
  - the **least-mean-square (LMS) algorithm** (Windrow and Hoff), also called ***delta rule***
  - and its generalization known as the ***back-propagation*** **(BP) algorithm.**

Adaline (**Ada**ptive **Line**ar Neuron) Networks

- **1960** - **Bernard Widrow** and his student **Marcian Hoff** introduced the ADALINE Networks and its learning rule which they called the Least mean square (LMS) algorithm (or Widrow-Hoff algorithm or delta rule)
- The Widrow-Hoff algorithm
  - can only train single-Layer networks.
- Adaline similar to the perceptron, the differences are ….?
- Both the Perceptron and Adaline can only solve linearly separable problems
  - (i.e., the input patterns can be separated by a linear plane into two groups, like AND and OR problems).

Adaline Architecture



$$v = \sum_{i=0}^{m} x_i w_i \qquad y = f(v) = v$$

- Given:
  - $x_k(n)$: an input value for a neuron *k* at iteration *n*,
  - $d_k(n)$: the desired response or the target response for neuron *k*.
- Let:
  - $y_k(n)$: the actual response of neuron *k*.

ADALINE's Learning as a Search

- Supervised learning: $\{p_1,d_1\}$, $\{p_2,d_2\}$,…,$\{p_n,d_n\}$
- The task can be seen as a search problem in the weight space:
  - Start from a random position (defined by the initial weights) and find a set of weights that minimizes the error on the given training set

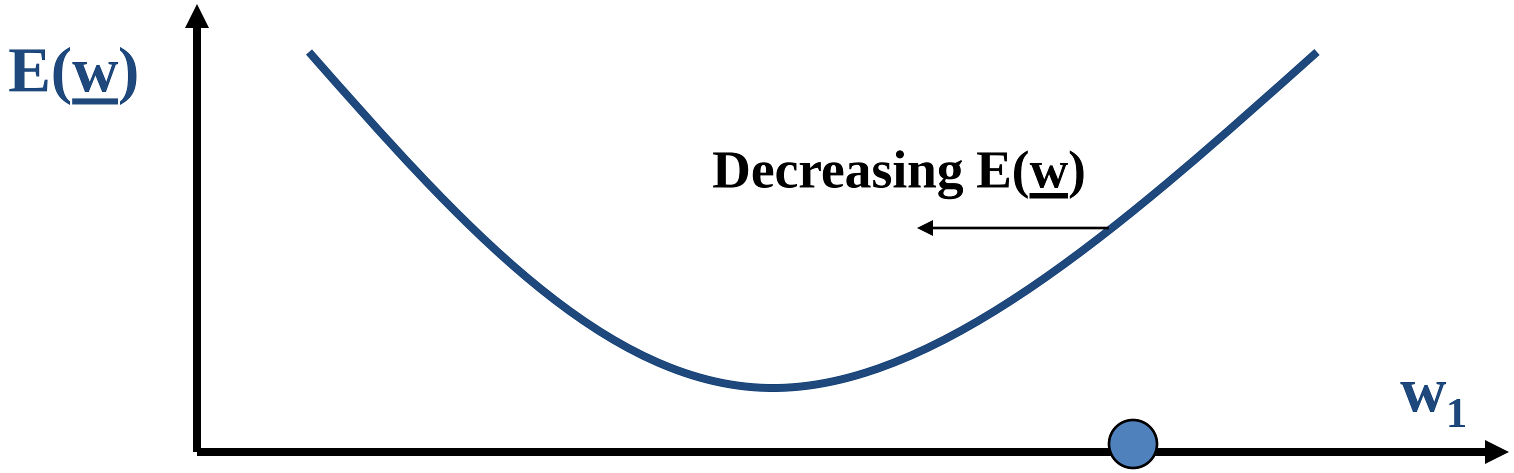
The error function: Mean Square Error

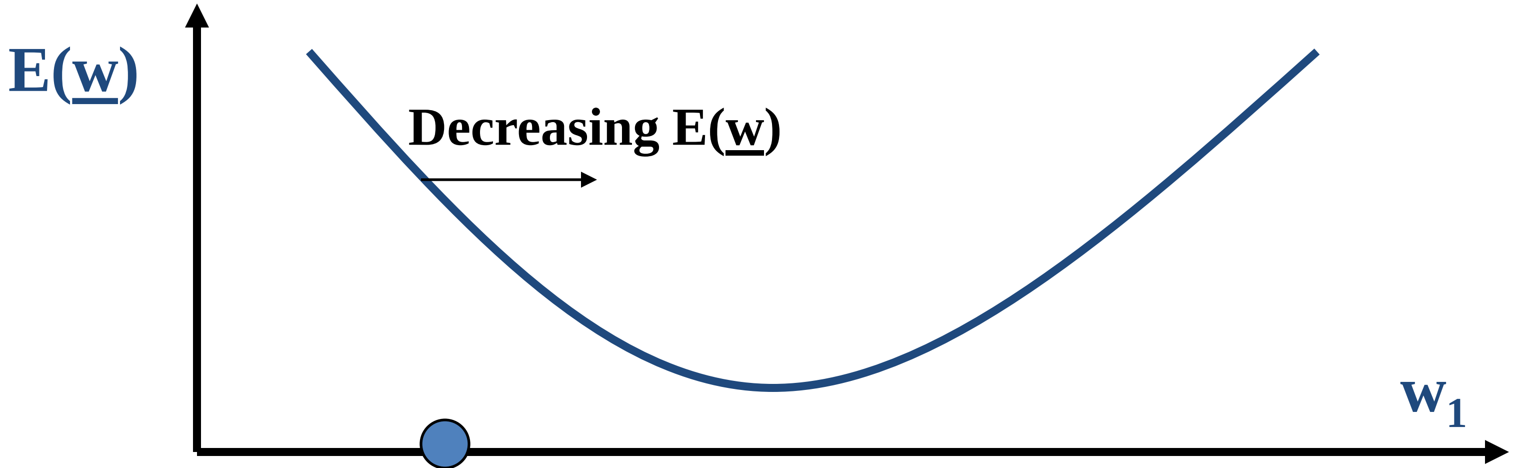
- ADALINEs use the Widrow-Hoff algorithm or Least Mean Square (LMS) algorithm to adjusts the weights of the linear network in order to minimize the mean square error
- Error : difference between the target and actual network output (delta rule).

  error signal for neuron k at iteration n: $e_k(n) = d_k(n) - y_k(n)$

Error Landscape in Weight Space

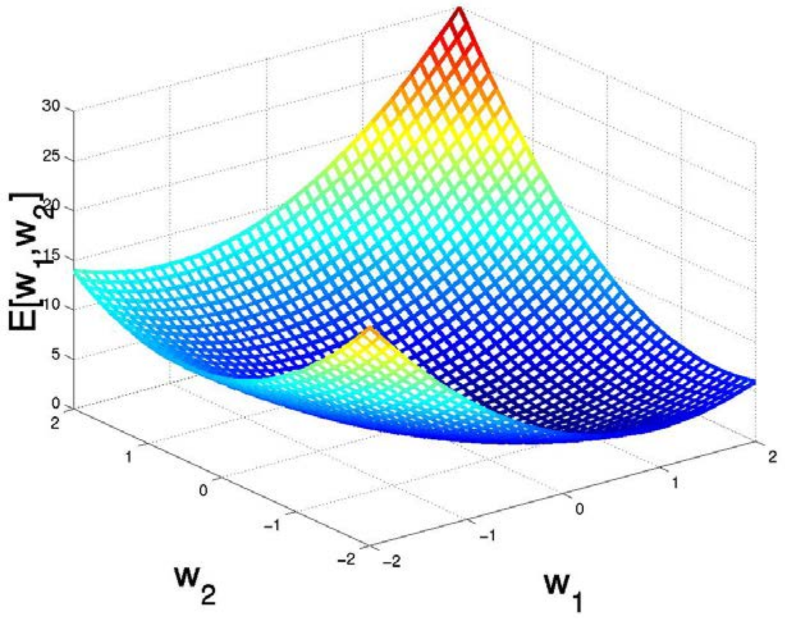
- Total error signal is a function of the weights
- Ideally, we would like to find the global minimum (i.e. the optimal solution)





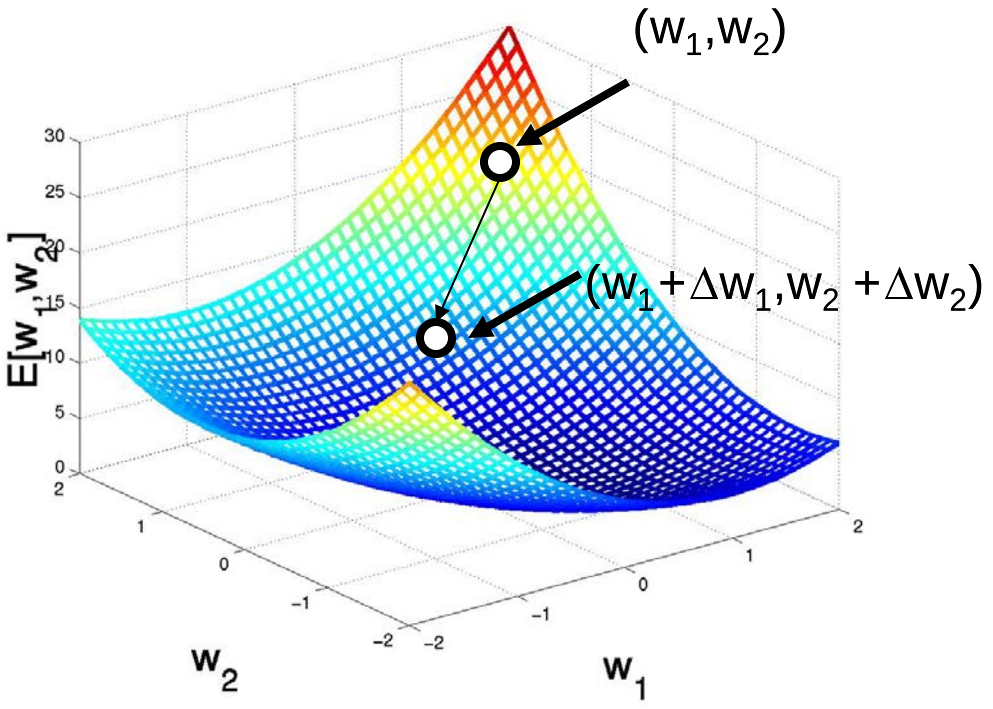
Error Landscape in Weight Space, cont.

- The error space of
  - the linear networks (ADALINE's) is a parabola (in 1d: one weight vs. error) or
  - a paraboloid (in high dimension)
- and it has only one minimum called the global minmum.



Error Landscape in Weight Space, cont.

- Takes steps downhill



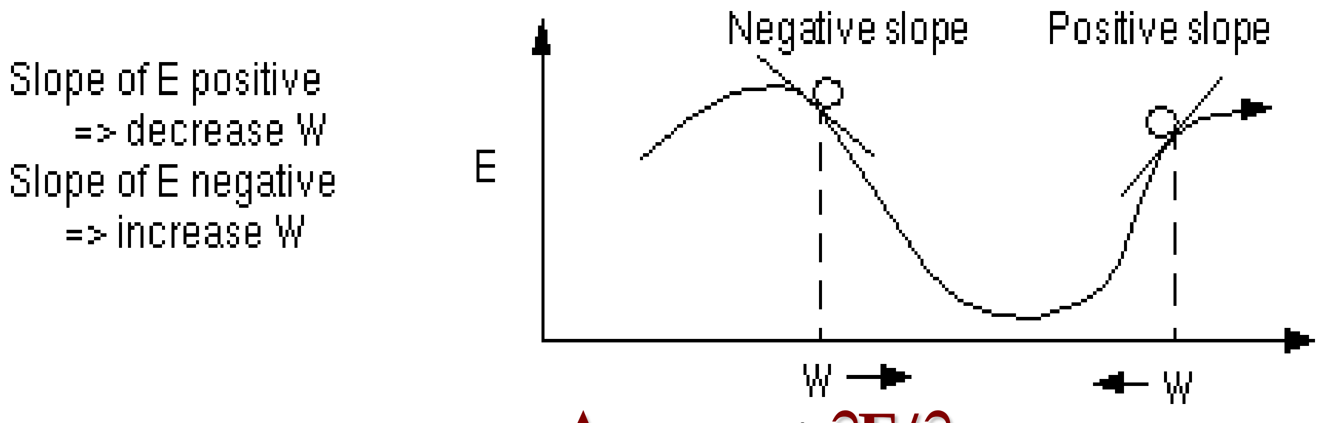
- Moves down as fast as possible

i.e. moves in the direction that makes the largest reduction in error

- how is this direction called?

Steepest Descent

- The direction of the steepest descent is called gradient and can be computed
- Any function increases most rapidly when the direction of the movement is in the direction of the gradient
- Any function decreases most rapidly when the direction of movement is in the direction of the negative of the gradient
- Change the weights so that we move a short distance in the direction of the **greatest rate of decrease** of the error, i.e., in the direction of –ve gradient.



$$\Delta w = - \eta * \partial E / \partial w$$

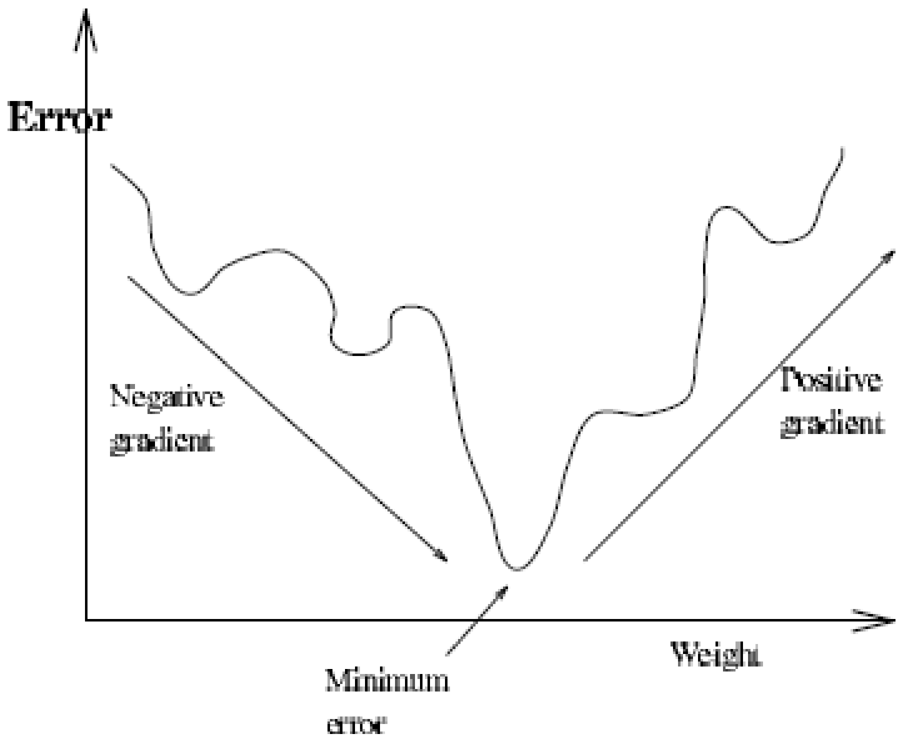


Figure 3.5: A schematic diagram showing error descent. In the negative gradient section, we wish to increase the weight; in the positive gradient section, we wish to decrease the weight

where the fraction is included due to inspired hindsight. Now, if our Adaline is to be as accurate as possible, we wish to minimise the squared error. To minimise the error, we can find the gradient of the error with respect to the weights and move the weights in the opposite direction. If the gradient is positive, the error would be increased by changing the weights in a positive direction and therefore we change the weights in a negative direction. If the gradient is negative, in order to decrease the error we must change the weights in a positive direction. This is shown diagrammatically in Figure 3.5. Formally $\Delta_P w_j = -\gamma \frac{\partial E^P}{\partial w_j}$.

We say that we are searching for the Least Mean Square error and so the rule is called the LMS or Delta rule or Widrow-Hoff rule. Now, for an Adaline with a single output, o,

$$\frac{\partial E^P}{\partial w_j} = \frac{\partial E^P}{\partial o^P}.\frac{\partial o^P}{\partial w_j} \tag{3.6}$$

The Gradient Descent Rule

- It consists of computing the **gradient** of the error function, then taking a **small step** in the **direction of negative gradient**, which hopefully corresponds to decrease function value, then repeating for the new value of the dependent variable.
- In order to do that, we calculate the partial derivative of the error with respect to each weight.
- The change in the weight proportional to the derivative of the error with respect to each weight, and additional proportional constant (learning rate) is tied to adjust the weights.

$$\Delta w = - \eta * \partial E/\partial w$$

LMS Algorithm - Derivation

- Steepest gradient descent rule for change of the weights:

Given

- $x_k(n)$: an input value for a neuron k at iteration n,
- $d_k(n)$: the desired response or the target response for neuron k.

Let:

- $y_k(n)$: the actual response of neuron k.
- $e_k(n)$: error signal = $d_k(n)- y_k(n)$

Train the w_i's such that they minimize the squared error after each iteration

$$E(n) = \frac{1}{2} e_k^2(n)$$

LMS Algorithm – Derivation, cont.

- The derivative of the error with respect to each weight $\dfrac{\partial E}{\partial w_{ij}}$ can be written as:

$$\nabla E(w_{ij}) = \frac{\partial E}{\partial w_{ij}} = \frac{\frac{1}{2}e_i^2}{\partial w_{ij}} = \frac{\partial \frac{1}{2}(d_i - y_i)^2}{\partial w_{ij}}$$

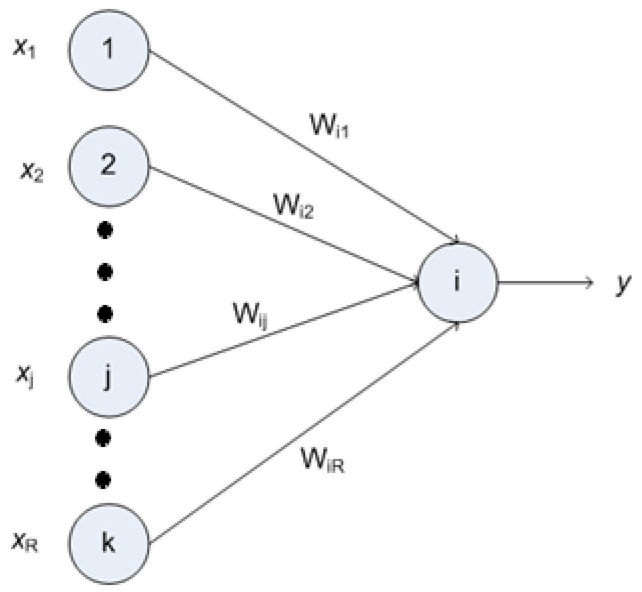
$$\begin{tikzcd} j \arrow[r, "w_{ij}"] & i \end{tikzcd}$$

- Next we use the chain rule to split this into two derivatives:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial \frac{1}{2}(d_i - y_i)^2}{\partial y_i} \frac{\partial y_i}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = \left(\frac{1}{2} * 2(d_i - y_i) * -1\right) * \frac{\partial f\left(\sum_{j=1}^{R} w_{ij} x_j\right)}{\partial w_{ij}}$$

$$\frac{\partial E}{\partial w_{ij}} = -(d_i - y_i) * x_j * f'\left(\sum_{j=1}^{R} w_{ij} x_j\right)$$



LMS Algorithm – Derivation, cont.

$$\nabla E(w_{ij}) = \frac{\partial E}{\partial w_{ij}} = -(d_i - y_i) * x_j * f^{'}(net_i)$$

$$\boxed{\Delta w_{ij} = -\eta \nabla E(w_{ij}) = \eta (d_i - y_i) f'(net_i) x_j}$$

- This is called the **Delta Learning rule**.
- Then
  - The Delta Learning rule can therefore be used Neurons with differentiable activation functions like the sigmoid function.

LMS Algorithm – Derivation, cont.

- The widrow-Hoff learning rule is a special case of Delta learning rule. Since the Adaline's transfer function is **linear function:**
- then $f\left(net_i\right) = net_i \qquad and \qquad f^{'}\left(net_i\right) = 1$

$$\nabla E\left(w_{ij}\right) = \frac{\partial E}{\partial w_{ij}} = -\left(d_i - y_i\right) * x_j$$

- The widrow-Hoff learning rule is:

$$\boxed{\Delta w_{ij} = -\eta \nabla E\left(w_{ij}\right) = \eta (d_i - y_i) x_j}$$

Adaline Training Algorithm

1- initialize the weights to small random values and select a learning rate, (η)

2- Repeat

3- **for m** training patterns

select input vector **X** , with target output, t,

compute the output: **$y = f(v)$, $v = b + w^T x$**

Compute the output error **$e=t-y$**

update the bias and weights

$$w_i(new) = w_i(old) + \eta\,(t - y)\,x_i$$

4- end for

5- **until** the stopping criteria is reached by find the Mean square error across all the training samples

$$mse(n) = \frac{1}{m}\sum_{k=1}^{m} E(n)^2$$

stopping criteria: if the Mean Squared Error across all the training samples **is less than a specified value, stop the training.**

Otherwise , cycle through the training set again (go to step 2)

Convergence Phenomenon

- The performance of an ADALINE neuron depends heavily on the choice of the learning rate η.
- How to choose it?
- Too big
  - the system will oscillate and the system will not converge
- Too small
  - the system will take a long time to converge
- Typically, η is selected by trial and error
  - typical range: $0.01 < \eta < 1.0$
  - often start at 0.1
  - sometimes it is suggested that:
    $0.1/m < \eta < 1.0/m$
    where m is the number of inputs
- Choose of η depends on trial and error.

Example

- The input/target pairs for our test problem are

$$\left\{ p_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, t_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \quad \left\{ p_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, t_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

- Learning rate: $\eta = 0.4$
- Stopping criteria: mse < 0.03

Show how the learning proceeds using the LMS algorithm?

Example Iteration One

- First iteration – p_1

$$y = w(0) \ * P_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0$$

e = t – y = -1 – 0 = -1

$$w(1) \ = w(0) \ + \ \eta * e \ * \ P_1$$

$$w(1) \ = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - 0.4 \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.4 \\ -0.4 \\ 0.4 \end{bmatrix}$$

Example Iteration Two

- Second iteration – p_2

$$y = w(1) \; * P_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$$

e = t – y = 1 – (-0.4) = 1.4

$$w(2) \; = w(1) \; + \; \eta * e \; * \; P_2$$

$$w(2) \; = \begin{bmatrix} 0.4 \\ -0.4 \\ 0.4 \end{bmatrix} + 0.4(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 0.96 \\ 0.16 \\ -0.16 \end{bmatrix}$$

End of epoch 1, check the stopping criteria

Example – Check Stopping Criteria

For input P_1

$$e_1 = t_1 - y_1 = t_1 - w(2)^T * P_1$$

$$= -1 - \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -1 - (-0.64) = -0.36$$

For input P_2

$$e_2 = t_2 - y_2 = t_2 - w(2)^T * P_2$$

$$= 1 - \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1 - (1.28) = -0.28$$

$$mse(1) = \frac{(-0.36)^2 + (-0.28)^2}{2} = 0.1037 > 0.03$$

Stopping criteria is not satisfied, continue with epoch 2

Example – Next Epoch (epoch 2)

- Third iteration – p_1

$$y = w(2)\ * P_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

e = t – y = -1 – 0.64 = -0.36

$$w(3)\ = w(2)\ +\ \eta * e\ *\ P_1$$

$$w(3)\ = \begin{bmatrix} 0.96 \\ 0.16 \\ -0.16 \end{bmatrix} + 0.4(-0.36) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1.104 \\ -0.016 \\ -0.016 \end{bmatrix}$$

if we continue this procedure, the algorithm converges to:
W(…) = [1 0 0]

ADALINE Networks - Capability and Limitations

- Both ADALINE and perceptron suffer from the same inherent limitation - can only solve linearly separable problems
- LMS, however, is more powerful than the perceptron’s learning rule:
  - Perceptron’s rule is guaranteed to converge to a solution that correctly categorizes the training patterns but the resulting network can be sensitive to noise as patterns often lie close to the decision boundary
  - LMS minimizes mean square error and therefore tries to move the decision boundary as far from the training patterns as possible
  - In other words, if the patterns are not linearly separable, i.e. the perfect solution does not exist, an ADALINE will find the best solution possible by minimizing the error (given the learning rate is small enough)

Comparison with Perceptron

- Both use updating rule changing with each input
- One fixes binary error; the other minimizes continuous error
- Adaline always converges; see what happens with XOR
- Both can REPRESENT Linearly separable functions
- **The Adaline,** is similar to the **perceptron**, but their transfer function is **linear** rather than **hard limiting**. This allows their output to take on any value.

Summary

- ADALINE Like perceptrons:
  - ADALINE can be used to classify objects into 2 categories
  - it can do so only if the training patterns are linearly separable
- **Gradient descent** is an optimization algorithm that approaches a local minimum of a function by taking steps proportional to the *negative* of the gradient (or the approximate gradient) of the function at the current point. If instead one takes steps proportional to the gradient, one approaches a local maximum of that function; the procedure is then known as **gradient ascent**.
- Gradient descent is also known as *steepest descent*, or the *method of steepest descent.*

Thank You!