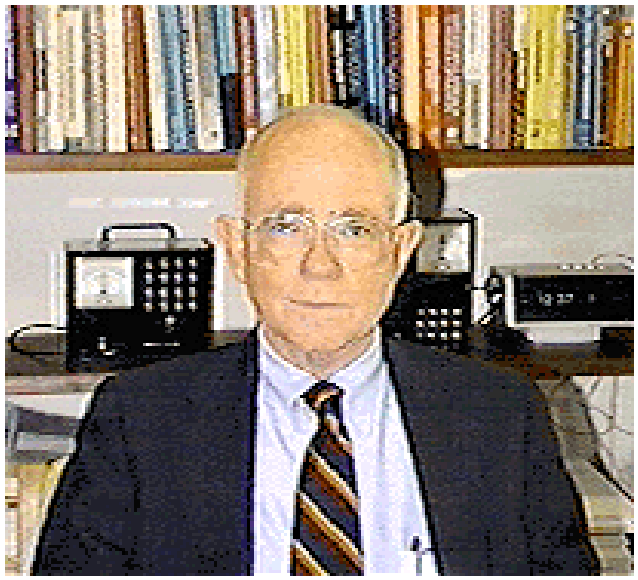
Adalines

Primitive Artificial Neurons

Sweet Adaline

- The Adaline (Adaptive Linear Neuron or “Adaptive Linear Element”) is a model similar to the Perceptron. There are several variations:
  - One has the **threshold** function similar to a perceptron.
  - Another uses a **pure linear** function with no threshold.
  - We can devise more.

Adaline Inventors
Bernard Widrow and Marcian (Ted) Hoff

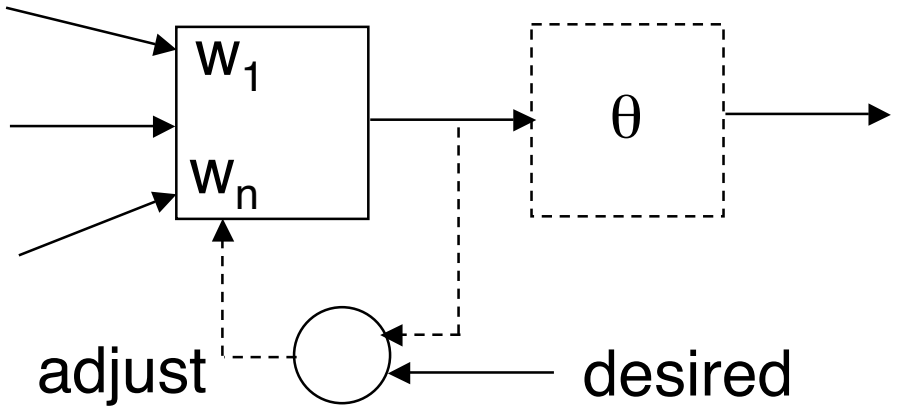


Bernard Widrow,
Professor Emeritus of E.E.,
Stanford University

Marcian Hoff
Co-inventor of Patent 3,821,715
Microprocessor Concept and Architecture

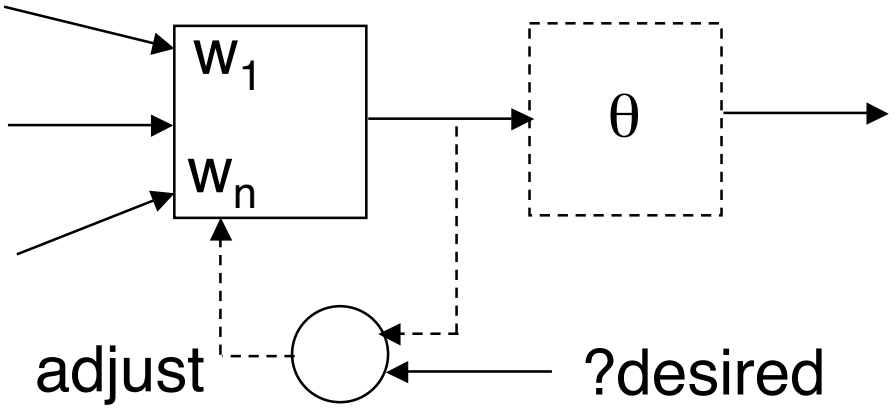
Adaline Training (1)

- With or without the threshold, the Adaline is **trained** based on the output of the *linear* function rather than the final output.



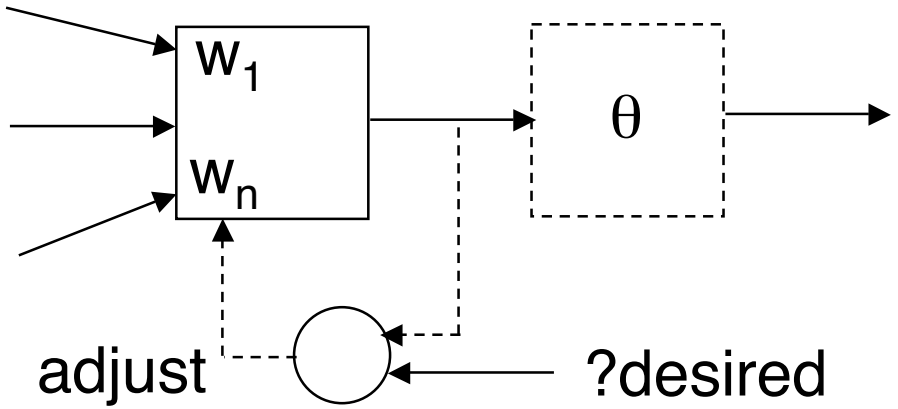
Adaline Training (2)

- The catch here is that we have to state the **desired** value in terms of the output of the linear part, rather than the output after the threshold.
- What is this for a classifier?



Adaline Training (2)

- A reasonable approach is to use any *nominal* target value such as -1 as desired for a “no” classification and a +1 for a “yes” classification.



Adaline Training (3)

- The formula for Adaline weight updating is very similar to the Perceptron:

Add to the weights Δw where

$$\Delta w = \varepsilon \eta [-1, x_1, x_2, \ldots, x_n]$$

Adaline learning rule

only now the ***error*** **is not limited to 1, -1, 0 as before**; it can have a fractional value, since it is based on the output of the linear part of the device.

Adaline Training (4)

- The weight update formula for the Adaline will be justified eventually.
- One major difference from this vs. the Perceptron is that a learning rate of 1 won't generally be acceptable. It will need to be smaller, say 0.01. There is a theory that tells us how large we can make it.

Adaline Convergence (2)

- The Adaline admits a more refined stopping criterion: The **Mean-Squared Error (MSE)** is the average of the squares of the error taken over all samples. Squaring makes the measure insensitive to the sign of the error. It also provides certain analytic properties.

- This quantity ideally converges toward a specific minimum (which might never be exactly attained). The algorithm can be set to stop when the MSE reaches a desired value.

Adaline Example

- We'll use the same example as before. But now we'll train on the output of the linear portion and target for +1 for a "yes" answer and -1 for a "no" answer.
  - (4, 5) +1
  - (6, 1) +1
  - (4, 1) -1
  - (1, 2) -1
- Try a learning rate of 0.01.

Adaline Training Example

- Start with initial weights all 0.
- (In general, can use random weights.)
- Progress:
  - trying sample desired: 1, inputs: -1 4 5 output: -1
  - diff is 2
  - error is 1
  - new weights: -0.01 0.04 0.05

Adaline Training Example

- weights: -0.01 0.04 0.05
- trying sample desired: 1, inputs: -1 6 1 output: 1
- diff is 0
- error is 0.7
- new weights: -0.017 0.082 0.057

- trying sample desired: -1, inputs: -1 4 1 output: 1
- diff is -2
- error is -1.402
- new weights: -0.00298 0.02592 0.04298

Adaline Training Example

- trying sample desired: -1, inputs: -1 4 1 output: 1
- diff is -2
- error is -1.402
- new weights: -0.00298 0.02592 0.04298

- trying sample desired: -1, inputs: -1 1 2 output: 1
- diff is -2
- error is -1.11486
- new weights: 0.0081686 0.0147714 0.0206828

- **epoch 1**: wrong = 3, mse = 1.17463

Adaline Training Example

- epoch 1: wrong = 3, mse = 1.17463
- epoch 2: wrong = 2, mse = 1.11176
- epoch 3: wrong = 2, mse = 1.08817
- epoch 4: wrong = 2, mse = 1.07521
- epoch 5: wrong = 2, mse = 1.06563

  . . .

- epoch 30: wrong = 2, mse = 0.888344
- epoch 31: wrong = 1, mse = 0.881999

  . . .

- epoch 197: wrong = 1, mse = 0.363726
- epoch 198: wrong = 0, mse = 0.362493

- Final weights: 1.54436 0.273645 0.252003

Adaline Training Example

- Final weights: 1.54436 0.273645 0.252003
  - input desired weighted sum actual
  - (4, 5) +1 0.810235 1
  - (6, 1) +1 0.359513 1
  - (4, 1) -1 -0.197777 -1
  - (1, 2) -1 -0.766709 -1

Alternate Rule Names

- Because the Adaline rule minimizes MSE, it is sometimes called the "**LMS rule**" [LMS = "least mean square"].
- The term "**Delta rule**" is also sometimes used, although this will be seen to be a rule for a more general class of networks.
- The "**Widrow-Hoff**" rule is also used.

Minimizing Error Iteratively

- Consider the task of finding the minimum of a function of one variable. Could try to extend this to functions of multiple variables.

- One standard method for doing this, if the derivative of the function is known, is **Newton's method**, which entails constructing a tangent line from a current estimate, then using the intersection of the line with the origin for the next.

  This method applies to finding the **zeroes** of the function, so it would be applied to the ***derivative*** of the function to be minimized.

What function is minimized?

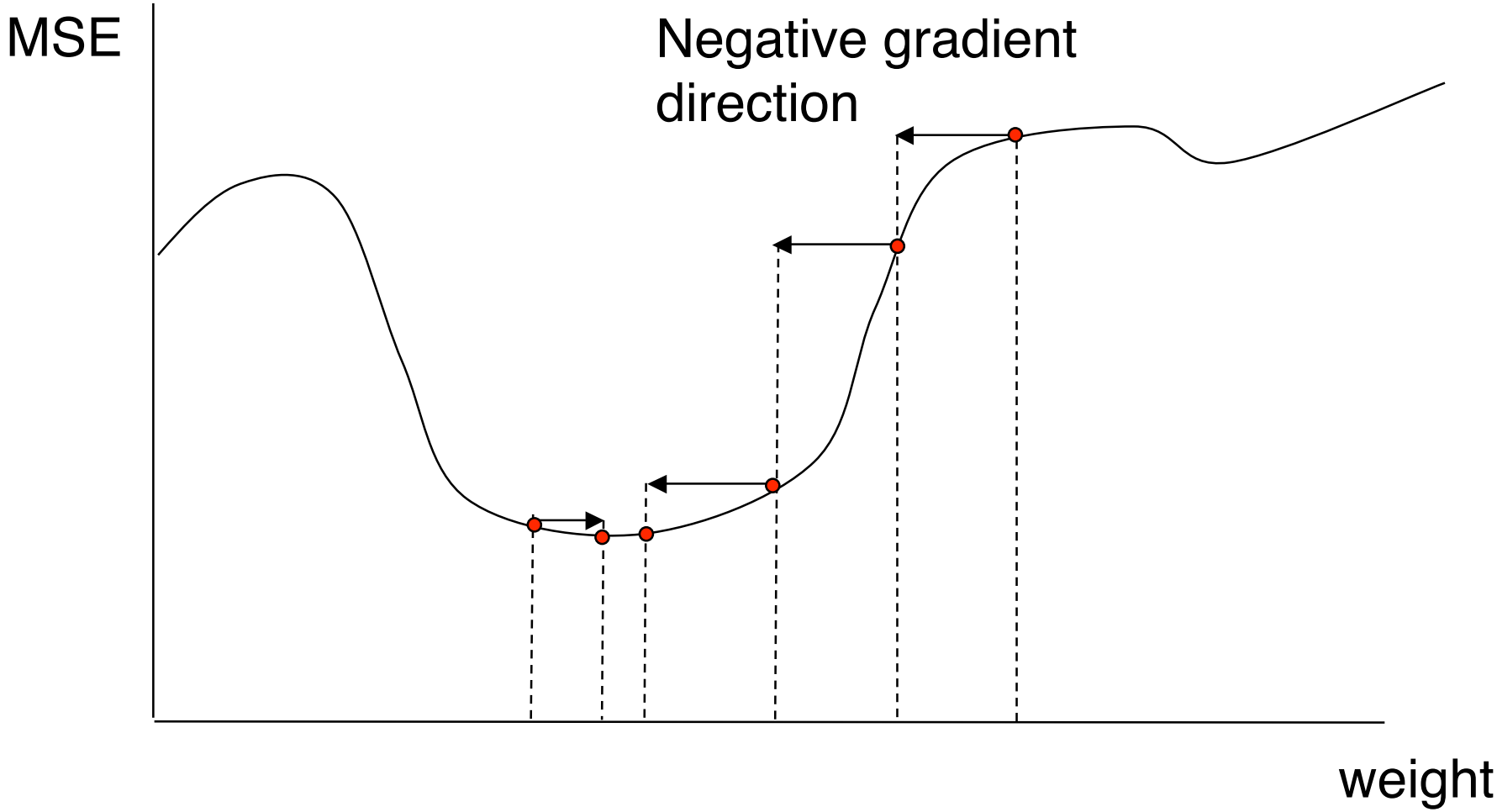
- Think of the inputs as being **constants**.
- The weights are the variables.
- We want to find the weights that minimize the **MSE** as a function of the weights.
- The fact that the MSE is defined analytically is a big help.

Gradient Descent

- Gradient descent is another method for finding the minimum.

- It consists of computing the **gradient** of the function, then taking a small **step** in the **direction of negative gradient**, which hopefully corresponds to decreased function value, then repeating for the new value of the dependent variable.

Gradient Descent

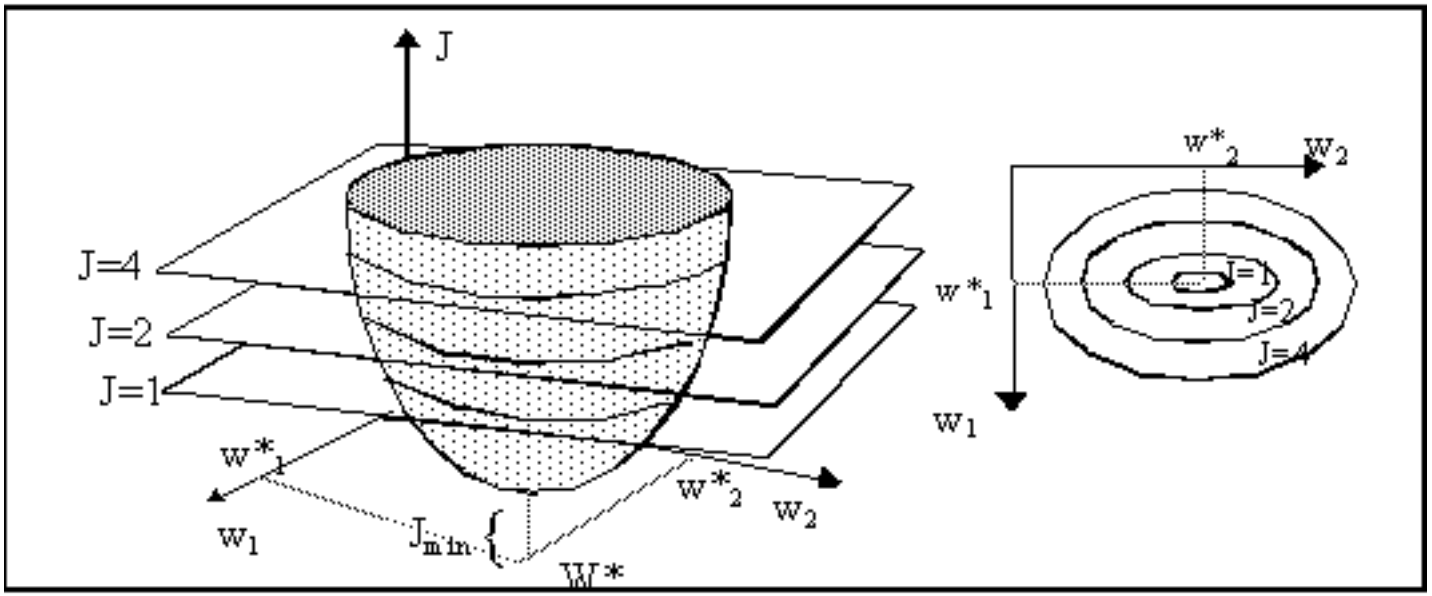


Gradient Descent

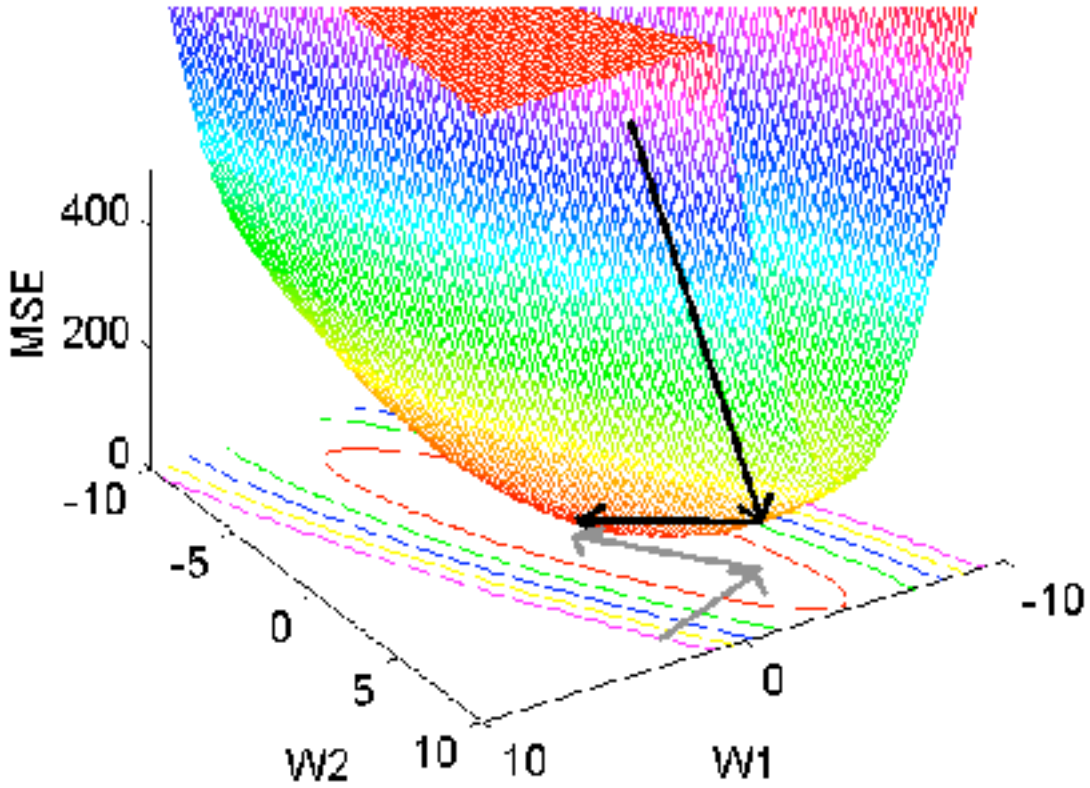
- The previous diagram is mainly to enhance our intuition.
- A single dimension for weights (including bias) is atypical.
- For the general case, the gradient is a **vector** of gradient components, one for each weight (including bias).

Error vs. 2-D Weight Space

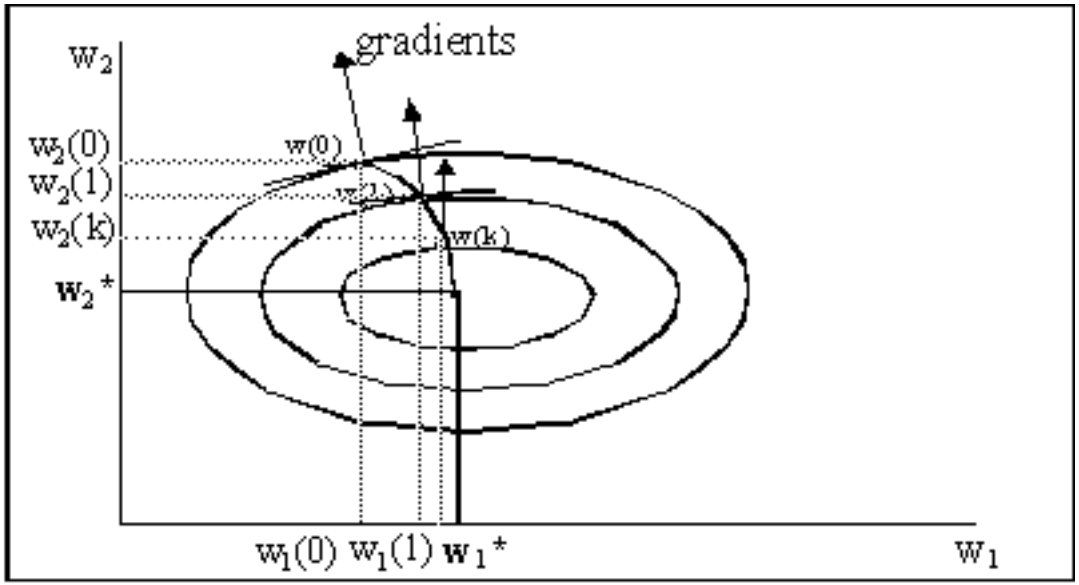
(one weight is threshold or -bias)



2-D Gradient Descent



2-D Gradient Descent



Computing Gradients

Note: squared

- MSE = J(w) = Σ(desired-actual)^2/n where Σ is over n samples.
- desired is a fixed constant for each sample.
- actual = $\Sigma w_j x_j$ (Σ over input lines)
- So J(w) = Σ(desired- $\Sigma w_j x_j$)^2/n

On-Line Approximation to Gradient

- "On-line" means based on a **single sample**, vs. "batch", which means using all samples
- $J \approx (d - \Sigma w_j x_j)^2$ ← (note: no outer sum) (d = desired)
- i^{th} gradient component = $\partial J/\partial w_i$
  $= \partial/\partial w_i (d - \Sigma w_j x_j)^2$
  $= 2 (d - \Sigma w_j x_j) \; \partial/\partial w_i (d - \Sigma w_j x_j) = -2\varepsilon x_i$

  $= 2 (d - \Sigma w_j x_j)$ = error, ε

  $\partial/\partial w_i (d - \Sigma w_j x_j) = -x_i$

Computing Gradients

- i^{th} gradient component = $-2\ \varepsilon\ x_i$
- However we want to move in the *direction* of ***negative*** gradient, tempered by the **learning rate** η, so:

  Amount to *add* to weight is

  $$\Delta w_i = 2\ \varepsilon\ \eta\ x_i$$

  which we recognize as the LMS (Adaline) rule (2 can be folded into η).

Vector Version of the Analysis of Gradient Descent for Adaline

- $MSE = J(w) = E[(d - w^T x)^2]$ (E = **expectation** or mean averaged over samples)

$= E[d^2 - 2dw^T x + w^T x\, x^T w]$

$= E[d^2] - E[2dw^T x] + E[\, w^T x\, x^T w]$

$= E[d^2] - 2w^T E[dx] + w^T E[x\, x^T]\, w$

$= c - 2w^T h + w^T R\, w$, for appropriate const. c, h, R

Vector Analysis of Gradient Descent for Adaline

- $J(w) = c - 2w^T h + w^T R\, w$

  where $c = E[d^2]$, $h = E[d\, x]$, $R = E[x\, x^T]$

- This is a **quadratic form** in w with coefficients derived from the data vectors x.

- R is called the (auto-)**correlation matrix.**

Standard Quadratic Form

- $J(w) = c + w^T b + (1/2) w^T A\, w$

  where $c = E[d^2]$,
  $b = -2E[d\, x]$,
  $A = 2E[x\, x^T]$

- A is called the **Hessian** matrix. It is the matrix of **2nd partial derivatives** of the surface.

Analytic Minimization by Gradient

- $\nabla_w J(w) = \nabla_w(c + w^T b + (1/2) w^T A\, w) =$
  $b + A\, w$

- It can be shown that if J has a **minimum**, it will be at a point w* where $\nabla J(w^*) = 0$,
  i.e. $b + A\, w^* = 0$
  i.e. $\boxed{w^* = A^{-1} b}$

  recalling $A = 2E[x\, x^T]$, $b = -2E[dx]$

Stable Points

- In general, $w^* = A^{-1} b$ is just a stable point.
- It may correspond to a minimum, maximum, or saddle.

Outline of Convergence of Gradient Descent for Adaline

- $\Delta w = 2\,\varepsilon\,\eta\,x$
  $w(k+1) = w(k) + 2\,\eta\,\varepsilon(k)\,x(k)$ (at k^{th} step)

- $E[w(k+1)] = E[w(k)] + 2\eta\,E[\varepsilon(k)\,x(k)]$
  … math ...
  $= (I-2\eta R)\,E[w(k)] + 2\eta h$
  (I is the identity matrix, and h a constant vector)

- For convergence, the eigenvalues of the matrix $I–2\eta R$ must be within the unit circle.

Convergence of Gradient Descent for Adaline

- If λ_i is an eigenvalue of R, convergence requires $|1 - 2 \eta \lambda_i| < 1$.
- which simplifies to $\eta < 1/\lambda_i$ for all eigenvalues λ_i, in particular for the maximum one.

$$\eta < 1/\lambda_{max}$$

Bound on learning rate for convergence of Adaline training by gradient descent

Example
(from NND)

$$\text{Sample1} \quad \left\{ \mathbf{x}_1 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{d}_1 = \begin{bmatrix} -1 \end{bmatrix} \right\} \qquad \text{Sample 2} \quad \left\{ \mathbf{x}_2 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, \mathbf{d}_2 = \begin{bmatrix} 1 \end{bmatrix} \right\}$$

$$\mathbf{R} = E[\mathbf{x}\mathbf{x}^T] = \frac{1}{2}\mathbf{x}_1\mathbf{x}_1^T + \frac{1}{2}\mathbf{x}_2\mathbf{x}_2^T$$

$$\mathbf{R} = \frac{1}{2}\begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} -1 & 1 & -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}\begin{bmatrix} 1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

$$\lambda_1 = 1.0, \qquad \lambda_2 = 0.0, \qquad \lambda_3 = 2.0$$

$$\eta < \frac{1}{\lambda_{max}} = \frac{1}{2.0} = 0.5$$

Training, 1st Epoch

Sample 1 $$a(0) = \mathbf{W}(0)\mathbf{p}(0) = \mathbf{W}(0)\mathbf{x}_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = 0$$

$$\varepsilon(0) = d(0) - a(0) = d_1 - a(0) = -1 - 0 = -1$$

$$\mathbf{W}(1) = \mathbf{W}(0) + 2\eta\varepsilon(0)\mathbf{x}^T(0)$$

$$\mathbf{W}(1) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} + 2(0.2)(-1) \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix}$$

2nd Epoch

Sample 2 $$a(1) = \mathbf{W}(1)\mathbf{p}(1) = \mathbf{W}(1)\mathbf{p}_2 = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = -0.4$$

$$\varepsilon(1) = d(1) - a(1) = d_2 - a(1) = 1 - (-0.4) = 1.4$$

$$\mathbf{W}(2) = \begin{bmatrix} 0.4 & -0.4 & 0.4 \end{bmatrix} + 2(0.2)(1.4) \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}^T = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix}$$

3rd Epoch

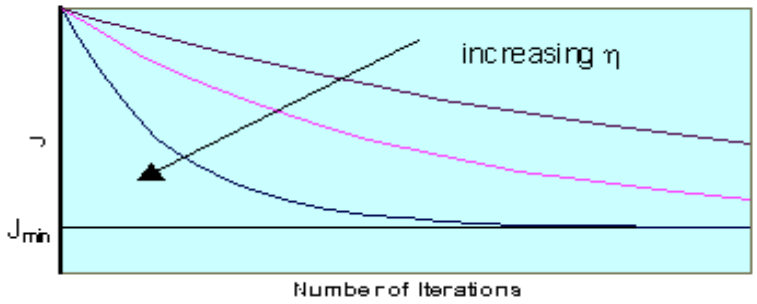
$$a(2) = \mathbf{W}(2)\mathbf{p}(2) = \mathbf{W}(2)\mathbf{p}_1 = \begin{bmatrix} 0.96 & 0.16 & -0.16 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} = -0.64$$

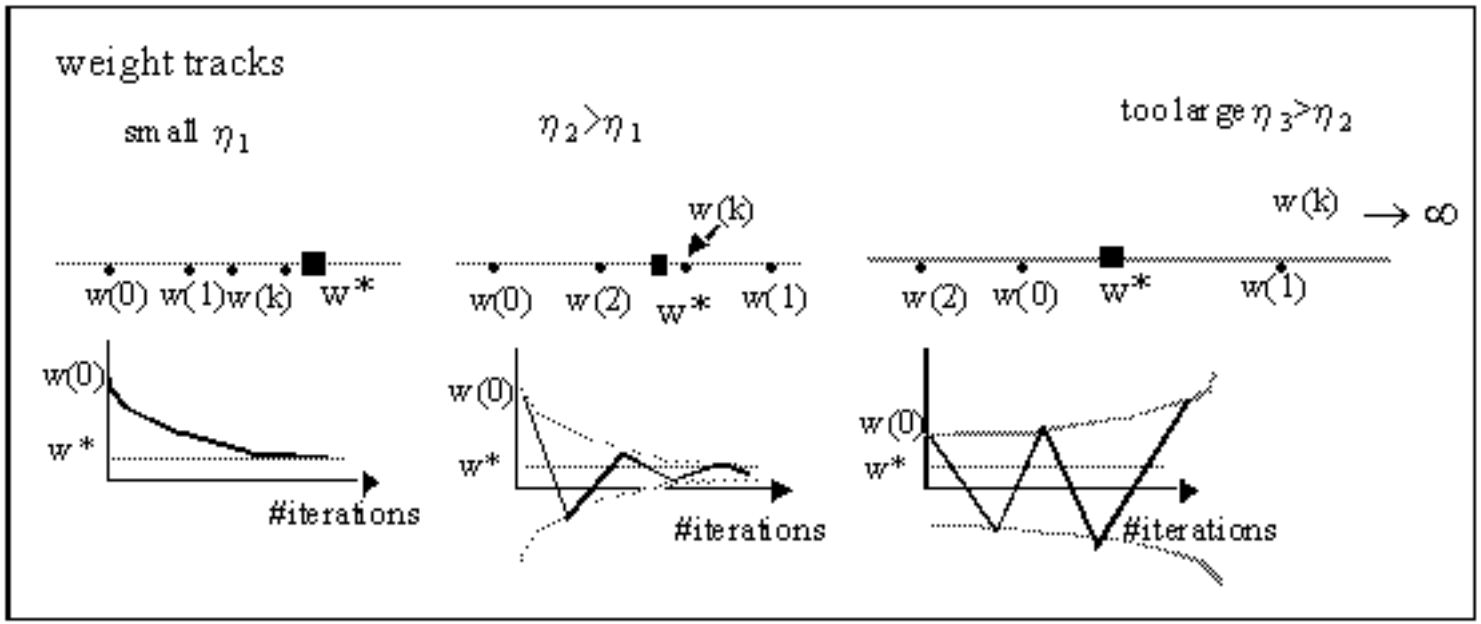
$$\varepsilon(2) = d(2) - a(2) = d_1 - a(2) = -1 - (-0.64) = -0.36$$

$$\mathbf{W}(3) = \mathbf{W}(2) + 2\eta\varepsilon(2)\mathbf{x}^T(2) = \begin{bmatrix} 1.1040 & 0.0160 & -0.0160 \end{bmatrix}$$

$$\mathbf{W}(\infty) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

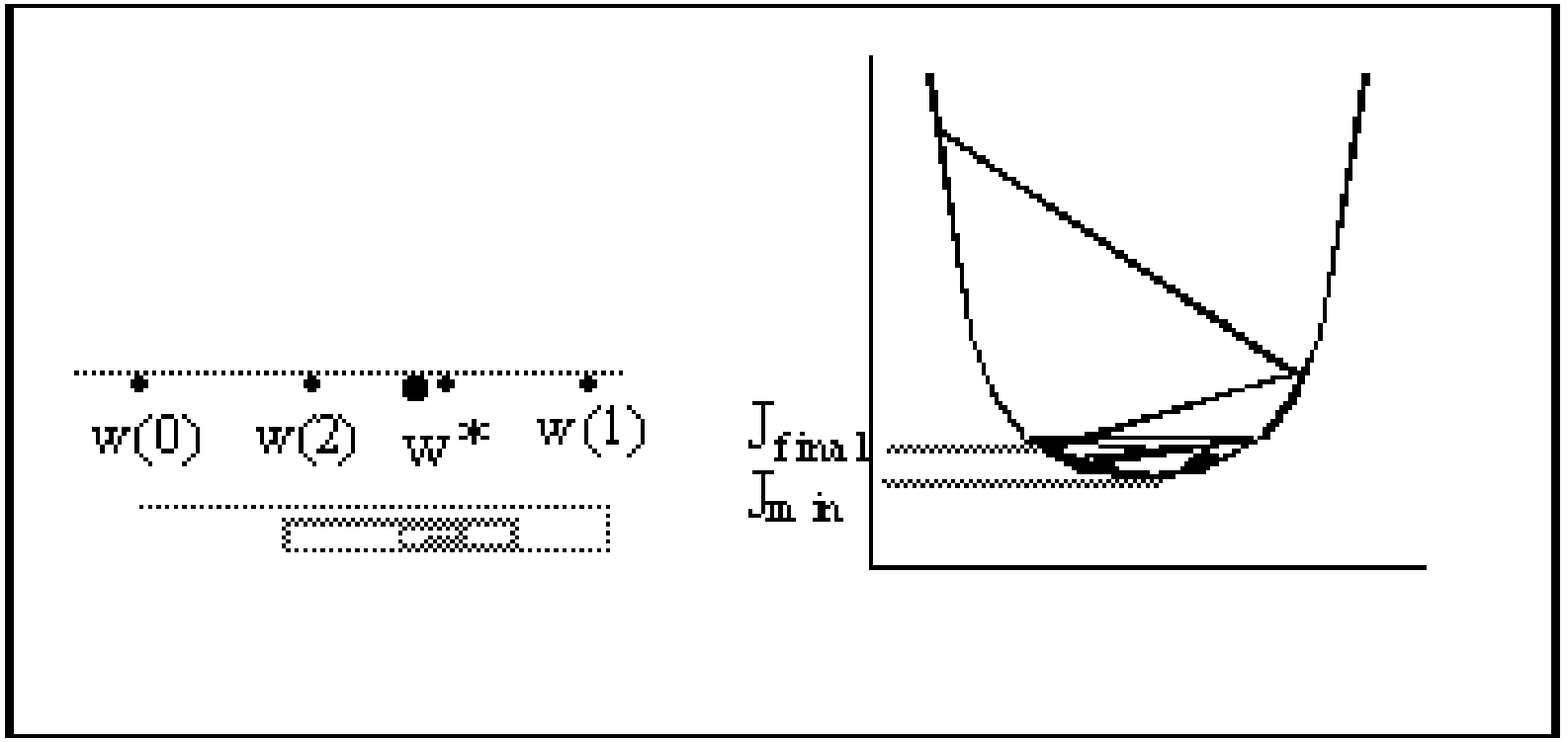
“Learning Curve”





“Rattling”

If learning rate too high, minimum solution won’t be achieved, even if there is no divergence.

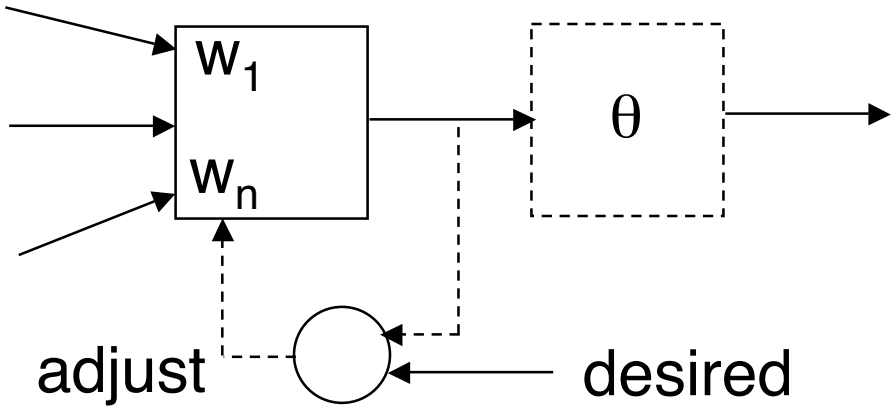


Solutions to Rattling

- Rule of thumb: Use a learning rate 0.1 times the theoretical maximum, **or**
- Gradually decrease the learning rate, e.g. according to a pre-set schedule, or
- Adaptively set the learning rate:
  - If the MSE is taking big steps, make it smaller.
  - If the MSE is taking small steps, make it larger.

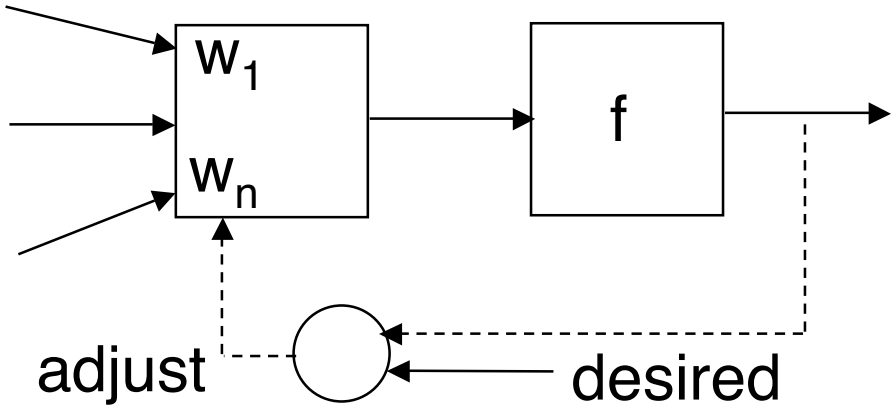
Generalizing the Adaline

- We have already discussed the problem of a threshold output in this picture



Generalizing the Adaline

- Suppose that we replace the threshold stage with a general analytic function f and revert to expressing desired in terms of its output:



Generalizing the Adaline

- Consider again the derivation of the LMS rule:
  $J(w) = \Sigma(\text{desired} - f(\Sigma w_j x_j))^2/n$
- $J \approx (d - f(\Sigma w_j x_j))^2$ (d = desired)

- i^{th} gradient component = $\partial J/\partial w_i$
  $= \partial/\partial w_i (d - f(\Sigma w_j x_j))^2$
  $= 2\,(d - f(\Sigma w_j x_j))\ \partial/\partial w_i\ (d - f(\Sigma w_j x_j))$
  $= -2\,\varepsilon\ \partial/\partial w_i\ f(\Sigma w_j x_j)$
  $= -2\,\varepsilon\ f'(\Sigma w_j x_j)\ \partial/\partial w_i\ \Sigma w_j x_j = -2\,\varepsilon\ x_i f'(\Sigma w_j x_j)$

Generalized LMS Rule (or Delta Rule)

- $\Delta w = 2\ \varepsilon\ \eta\ f'(\Sigma w_j x_j)\ x_i$
  assuming that f has a derivative f'.

- $\Sigma w_j x_j$ is often called the **"net" value** or **"activation" value**, and f the **activation function**.

- For the special case of f being the **identity** function, this reduces to the LMS rule we had before.

Generalized LMS Rule (or Delta Rule)

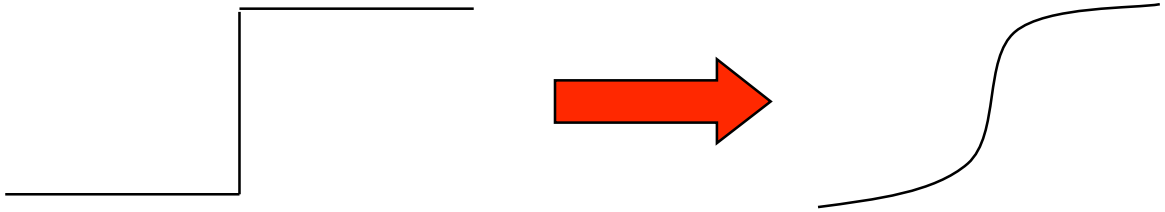
- Why worry about this generalization?

$$\Delta w = 2\ \varepsilon\ \eta\ f'(\Sigma w_j x_j)\ x_i$$

- It will have a number important uses.

Use #1

- In the Adaline with **threshold**, we can't very well treat the model analytically, due to the fact that we have a non-continuous function at the output.
- But we can *approximate* the non-continuous function with a continuous one:



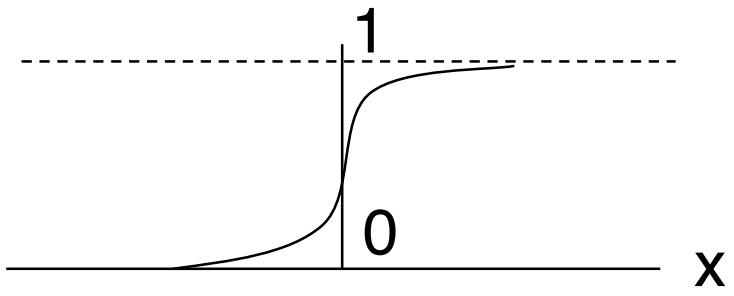
Sigmoids

- The “S” shape on the right of the previous slide is called a sigmoid curve.
- This is a generic term and there are several different analytic functions that behave this way.

Logisitic Sigmoid

- Logisitic function (“logsig”–Matlab):
  $f(x) = 1/(1+\exp(-ax))$

- $f'(x) = f(x)(1-f(x))$



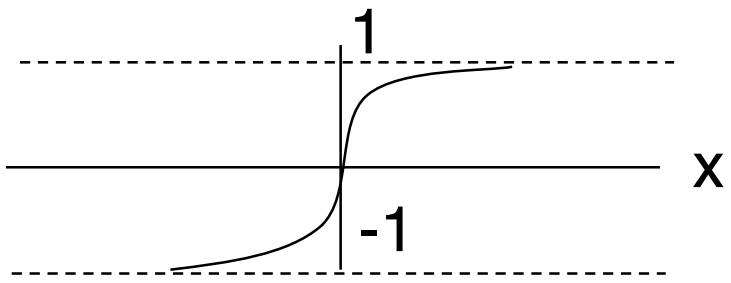
Hyperbolic Sigmoid

- Hyperbolic tangent function (“tansig”):

  $f(x) = \tanh(x)$

  $=(\exp(x)-\exp(-x))/(\exp(x) + \exp(-x))$

- $f'(x) = 1-f^2(x)$



Squashing Functions

- Sigmoids, step functions, and other functions that force their results to be in a limited range are called “squashing functions”.

- It is generally accepted that biological neural system is based on such functions, since there are physical limits to the response level.