30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation

Widrow et. al.

Carissa Bush, Vidya Srinivas, Po-Chun Huang, Ming Hung Chen

Introduction

- Machine learning concepts have been around for a while
- Many ideas were discovered, and re-discovered independently again
- Many ideas were biologically-inspired
- Everything started with simple concepts

The Adaptive Linear Combiner

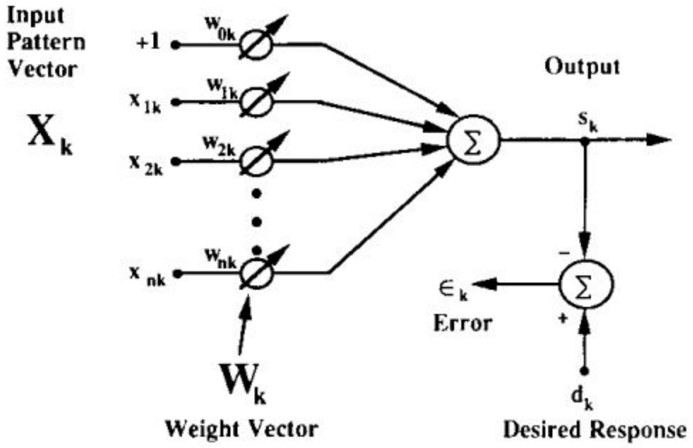
- Outputs a weighted sum of inputs
- Each input has an associated weight
- Weights can be discrete or continuous
- Weights can be adapted

Inputs: $\mathbf{X_k} = \begin{bmatrix} x_0 & x_{1k} & x_{2k} & \dots & x_{nk} \end{bmatrix}^\intercal$

Weights: $\mathbf{W_k} = \begin{bmatrix} w_0 & w_{1k} & w_{2k} & \dots & w_{nk} \end{bmatrix}^\intercal$

Output: $s_k = \mathbf{X_k^\intercal W_k} = w_0x_0 + w_{1k}x_{1k} + w_{2k}x_{2k} \cdots + w_{nk}x_{nk}$

Desired Response: d_k, Error: ϵ_k



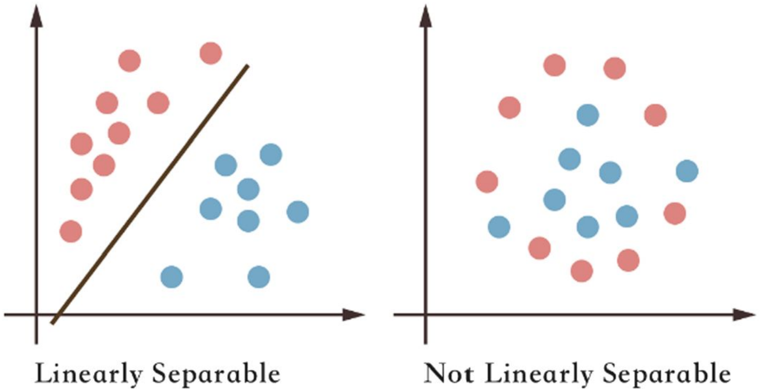
Binary Classification and Linear Separability

Binary Classification

- Target task is to classify input patterns into two groups: positive and negative

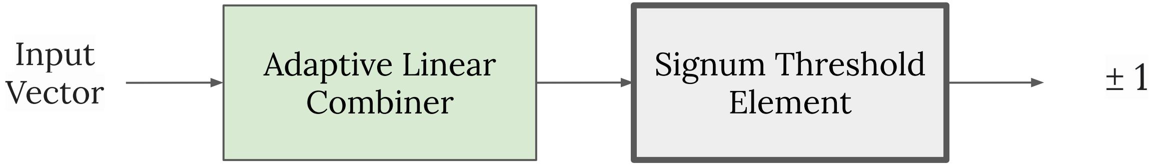
Linear Separability

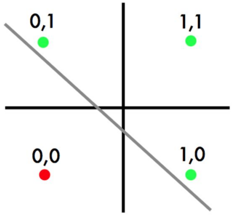
- A set of input patterns is linearly separable if there exists a line that can separate points with positive labels from points with negative labels



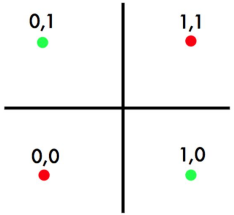
An Adaptive Linear Classifier: Adaline

- Performs binary classification task
- Is able to realize linear separating boundaries
- These boundaries perfectly classify linearly separable datasets





OR

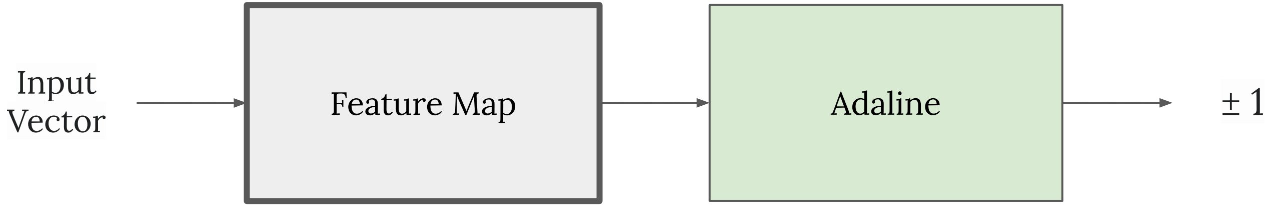


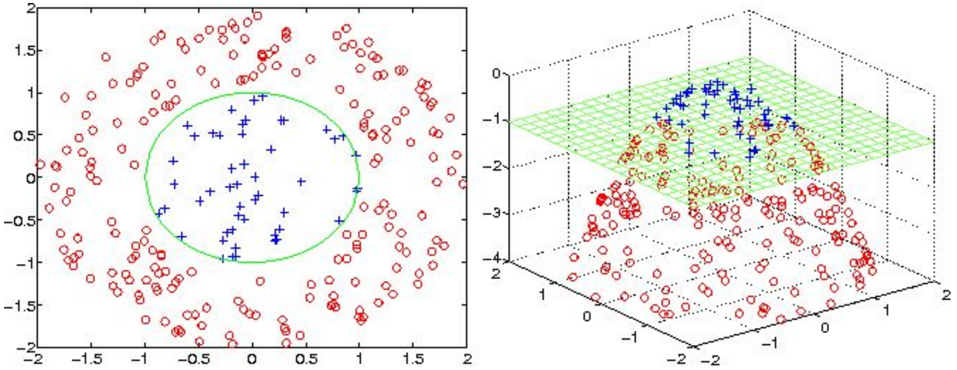
XOR

Realizing Nonlinear Separating Boundaries

Nonlinear Classifiers: Feature Maps

- If the input patterns are not separable in the original input space, map them to a higher-dimensional space where they are linearly separable

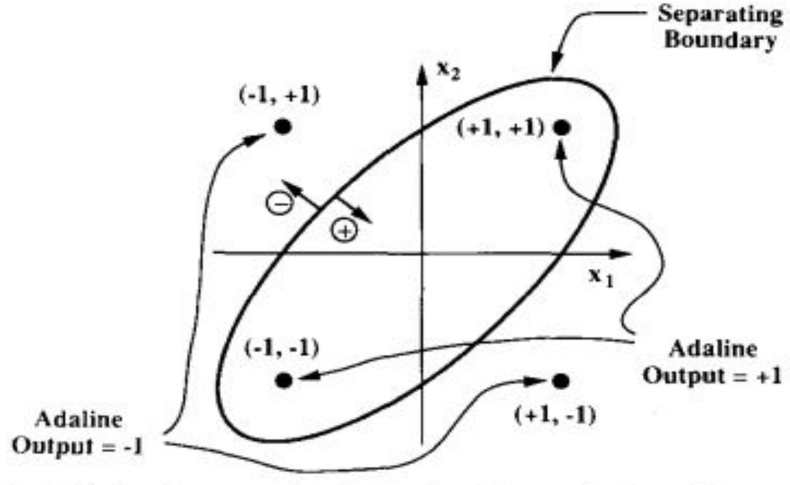




(a) Input Space (b) Projected Space

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^\intercal \mapsto \phi(\mathbf{x}) = \begin{bmatrix} x_1 & x_1^2 & x_1 x_2 & x_2^2 & x_2 \end{bmatrix}^\intercal$$

ϕ: Our feature map



Nonlinear Classifiers: Feature Maps

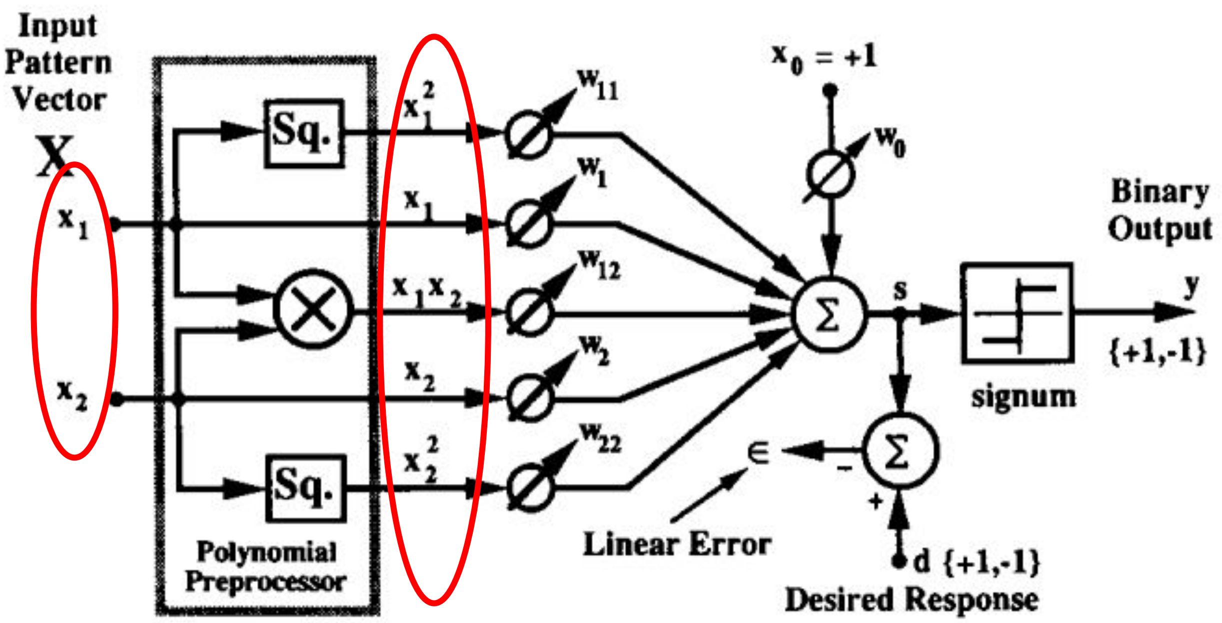
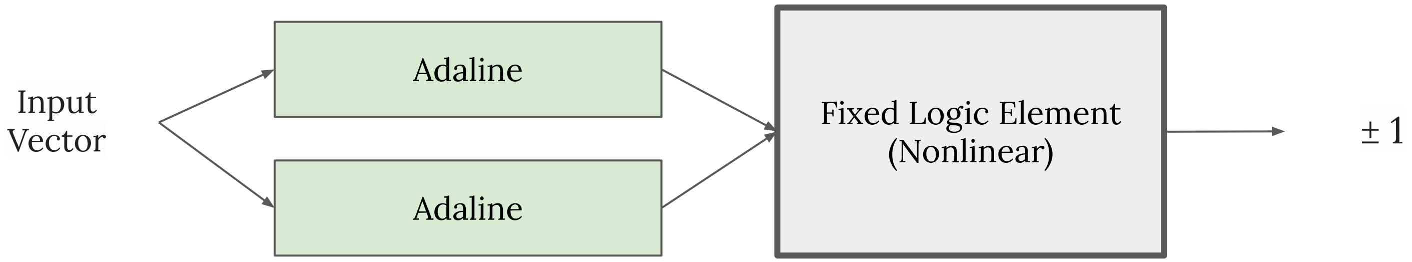
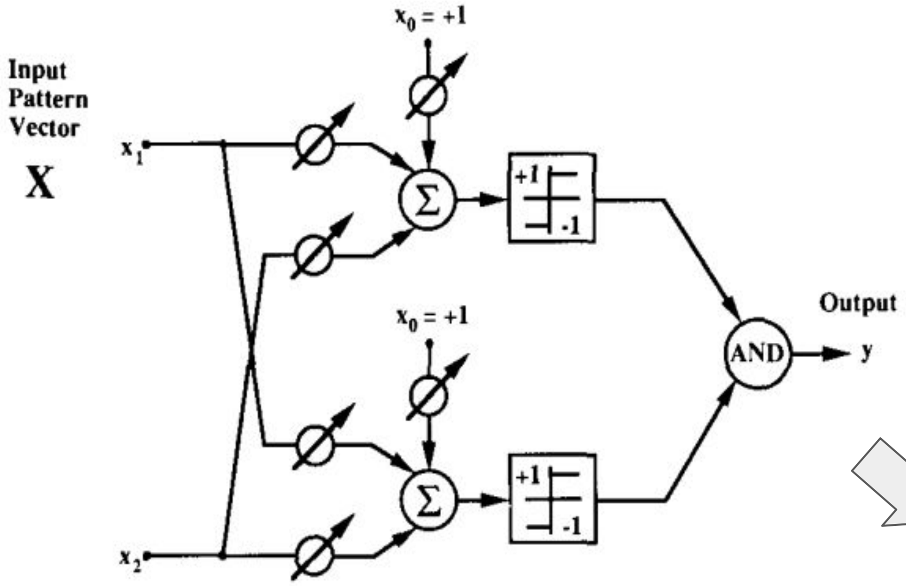


Fig. 6. Adaline with inputs mapped through nonlinearities.

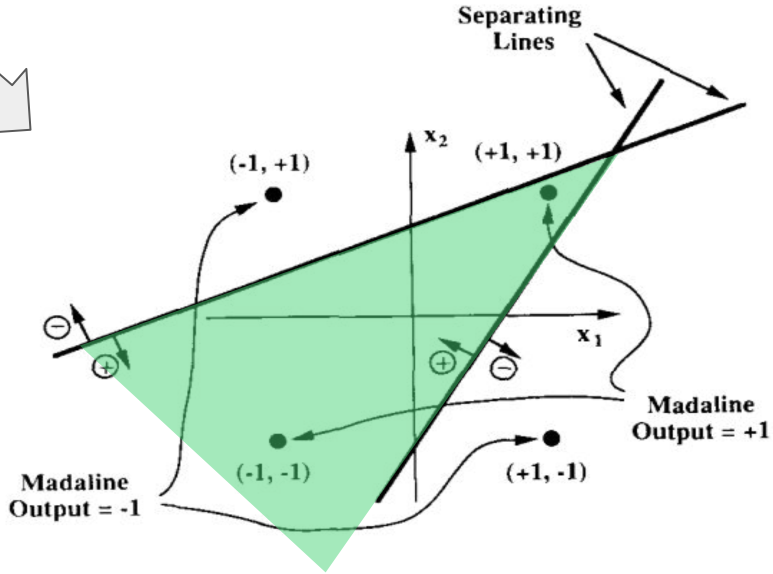
Nonlinear Classifiers: Madaline I

- If the input patterns are not separable in the original input space, use a nonlinear combination of linear separating boundaries to realize a nonlinear separating boundary





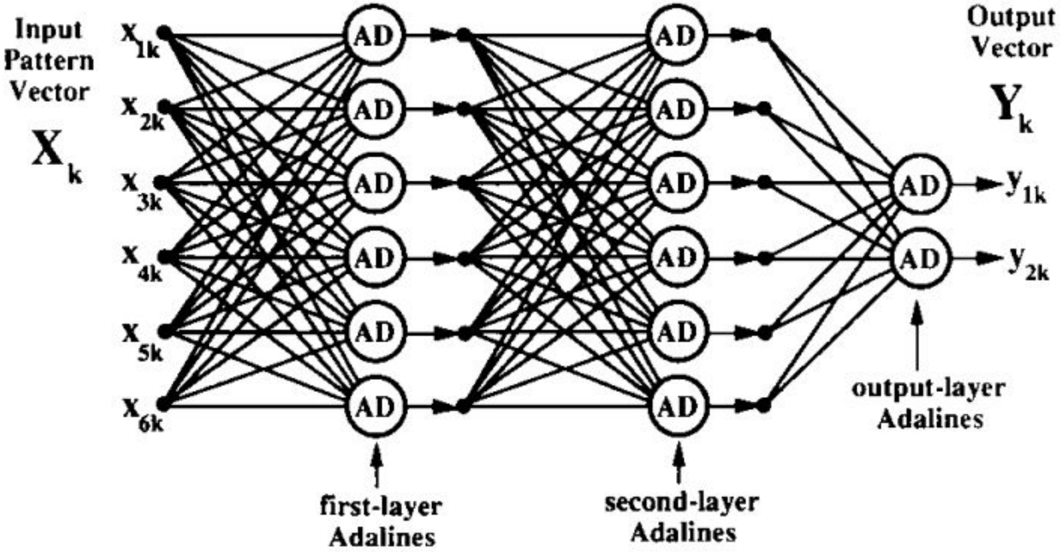
Input
Pattern
Vector
X
x1
x2
x0 = +1
x0 = +1
+1
-1
+1
-1
AND
Output
y



Separating
Lines
x2
x1
(-1, +1)
(+1, +1)
(-1, -1)
(+1, -1)
Madaline
Output = +1
Madaline
Output = -1

Nonlinear Classifiers: Feedforward Networks

- Combines multiple Adalines in layers and feeds outputs of one layer to inputs of the next layer



Adaptation - the Minimal Disturbance Principle

Error-correction (EC) rules

- Alter the weights of a network to correct error in the output response to the present input pattern

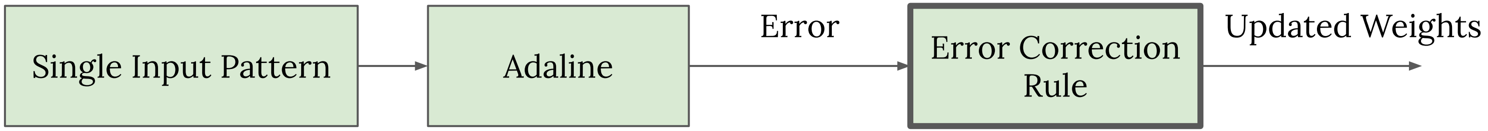
Steepest descent (SD) rules

- Alter the weights of a network by gradient descent
- Reduce error averaged over all input patterns

Error Correction Rules: Single Threshold Element

EC Single Threshold Element

- Every new input pattern starts a new adaptation cycle
- Goal is to update weights to reduce error
- Linear rules will make corrections directly proportional to the error
- Nonlinear rules will make corrections that are not directly proportional to the error



EC Single Threshold Element: α-LMS

- Weight update is independent of input pattern magnitude
- Choice of α controls stability and speed of convergence

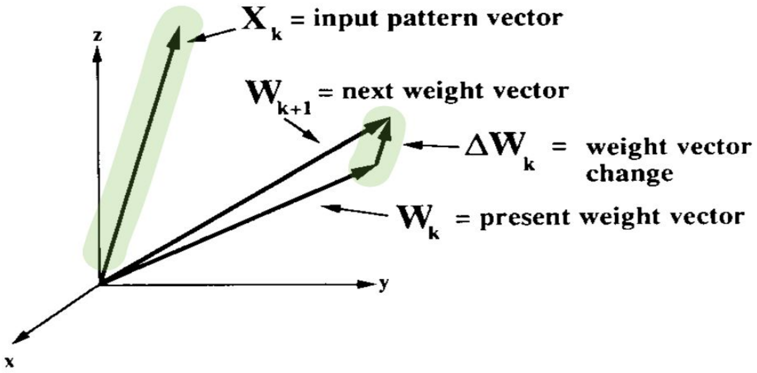
$$\boxed{\boldsymbol{W}_{k+1} = \boldsymbol{W}_k + \alpha \frac{\epsilon_k \boldsymbol{X}_k}{|\boldsymbol{X}_k|^2}.}$$

α: Error correction constant

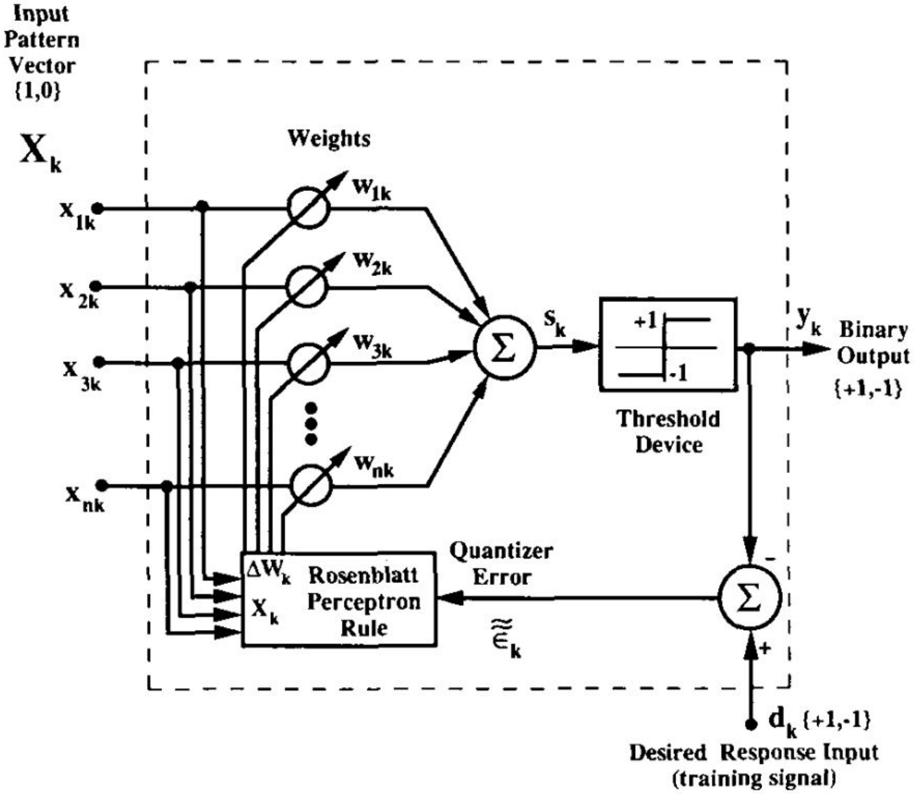
$\mathbf{X_k}$: Inputs

$\mathbf{W_k}$: Weights

ϵ_k: Error



EC Single Threshold Element: Perceptron Rule



$\tilde{\tilde{\epsilon}}$: quantizer error

d_k: desired response

y_k: output of quantizer

EC Single Threshold Element: Perceptron Rule

- Adds or subtracts input patterns to weights to correct for error
- Guaranteed to converge for any linearly separable input patterns
- No guarantees of convergence for input patterns that aren't linearly separable

$$\mathbf{W_k} = \mathbf{W_{k+1}} = \begin{cases} \mathbf{W_k} + \mathbf{x_i} & \tilde{\tilde{\epsilon}} > 0 \\ \mathbf{W_k} - \mathbf{x_i} & \tilde{\tilde{\epsilon}} < 0 \\ \mathbf{W_k} & \tilde{\tilde{\epsilon}} = 0 \end{cases}$$

$$\bar{\bar{\epsilon}}_k \triangleq d_k - y_k .$$

$\tilde{\tilde{\epsilon}}$: quantizer error

d_k: desired response

y_k: output of quantizer

$\mathbf{x_i}$: ith input pattern

EC Single Threshold Element: May's Rules

- All desired responses are +1 or -1
- Defines a "dead zone" or margin around 0, denoted by $\pm\gamma$

Increment Adaptation Rule

- Input pattern falls outside margin: weight vector is adapted by Perceptron rule
- Input pattern falls within margin: weight vector is adapted by normalized variant of fixed increment Perceptron rule

α: Error correction constant
$\mathbf{X_k}$: Inputs
$\mathbf{W_k}$: Weights
s_k: Linear output of system

$\tilde{\tilde{\epsilon}}$: quantizer error
d_k: desired response

$$\boldsymbol{W}_{k+1} = \begin{cases} \boldsymbol{W}_k + \alpha \tilde{\tilde{\epsilon}}_k \dfrac{X_k}{2|X_k|^2} & \text{if } |s_k| \geq \gamma \\ \boldsymbol{W}_k + \alpha d_k \dfrac{X_k}{|X_k|^2} & \text{if } |s_k| < \gamma \end{cases}$$

EC Single Threshold Element: May's Rules

Modified Relaxation Adaptation Rule

- Input pattern falls outside margin and correctly classified: weight vector is <u>not modified</u>
- Input pattern falls within margin or is misclassified: weight vector is adapted by normalized variant of fixed increment Perceptron rule
- If γ (dead-zone) is set to ∞, this turns into α-LMS

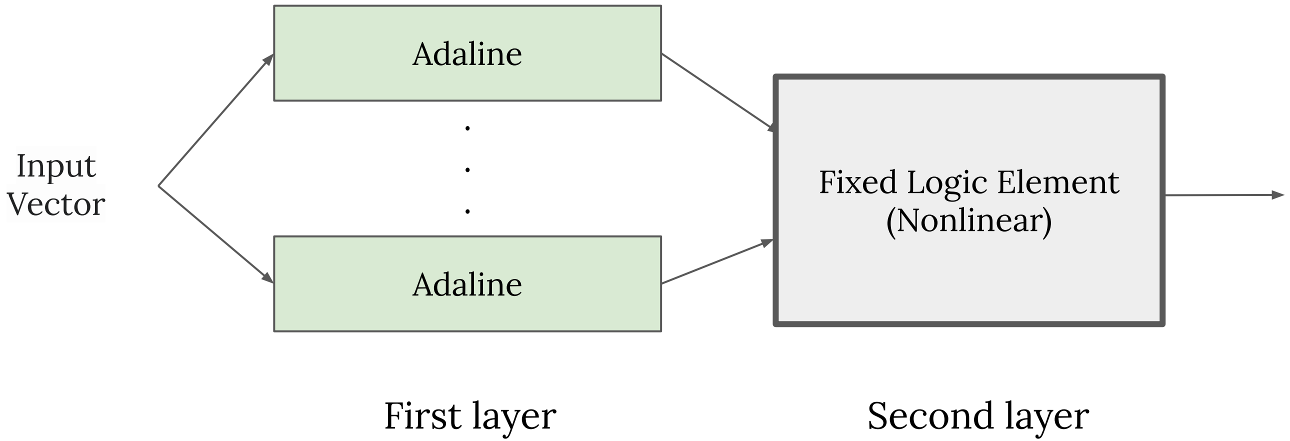
$$
W_{k+1} = \begin{cases} W_k & \text{if } \tilde{\tilde{\epsilon}}_k = 0 \text{ and } |s_k| \geq \gamma \\ W_k + \alpha\epsilon_k \dfrac{X_k}{|X_k|^2} & \text{otherwise} \end{cases}
$$

α: Error correction constant $\mathbf{X_k}$: Inputs $\mathbf{W_k}$: Weights
$\tilde{\tilde{\epsilon}}$: quantizer error
s_k: Linear output of system

Error Correction Rules - Multi-Element Networks

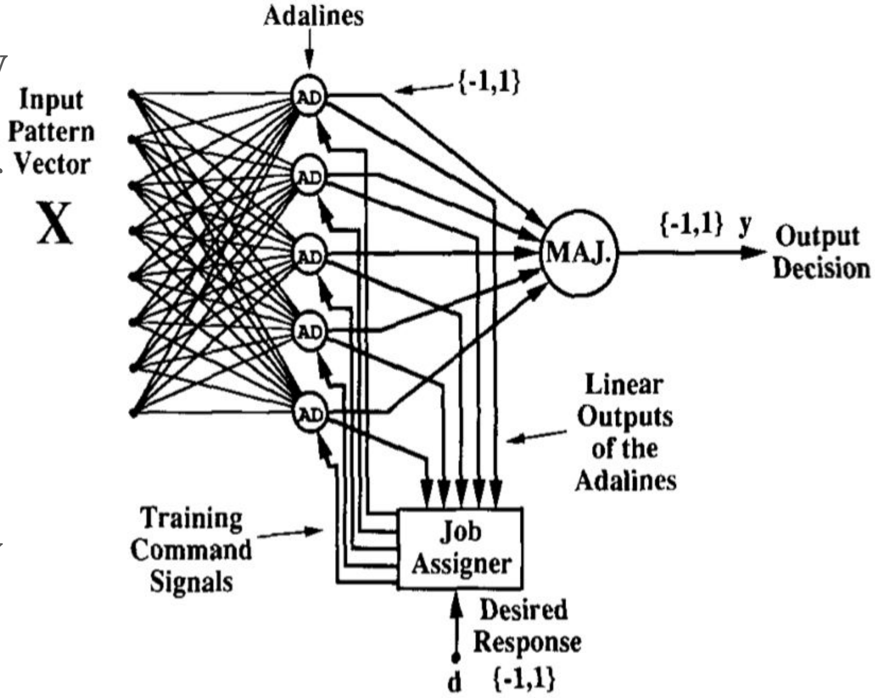
Madaline Rule I (MRI)

- Allows the adaptation of a first-layer element because the logic element is fixed
- The second layer consists of a single fixed-threshold-logic element which may be OR gate, AND gate, majority-vote-taker, etc.



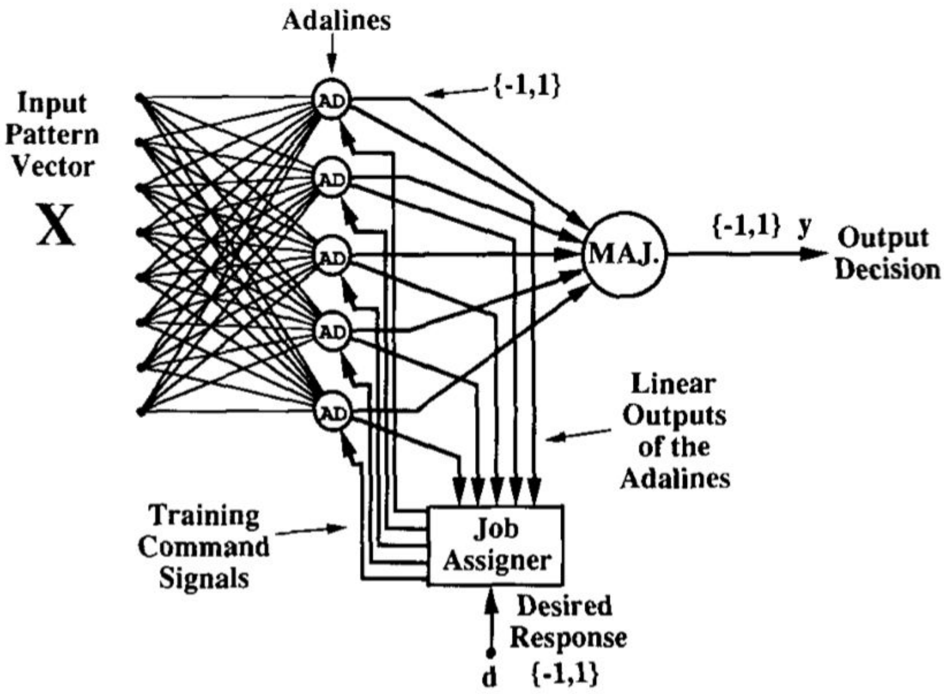
Madaline Rule I: Adaptation

- The weights of the Adalines are initially set to small random values
- Weight vector can be adapted by any of the single error-correction rules
  - Reverse the Adaline's output
  - **α**-LMS
  - Perceptron
- **Main idea:** Assign responsibility to the Adaline or Adalines that can most easily assume it



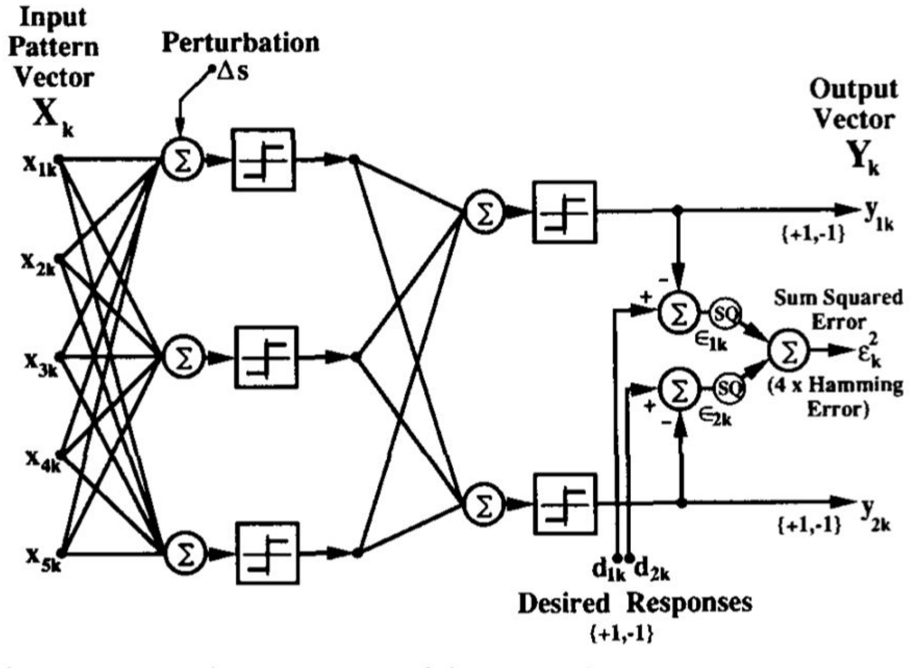
Madaline Rule I

- Job assigner assigns responsibility
- Load sharing is important
- Pattern presentation sequence should be random
- The adaptation process could hang up in local optima
- Obeys the minimal disturbance rule



Madaline Rule II (MRII)

- Extended from MRI
- Used for multilayer binary networks
- Weights are initially set to small random values
- Training patterns are presented in a random sequence
- Could hang up in local optima

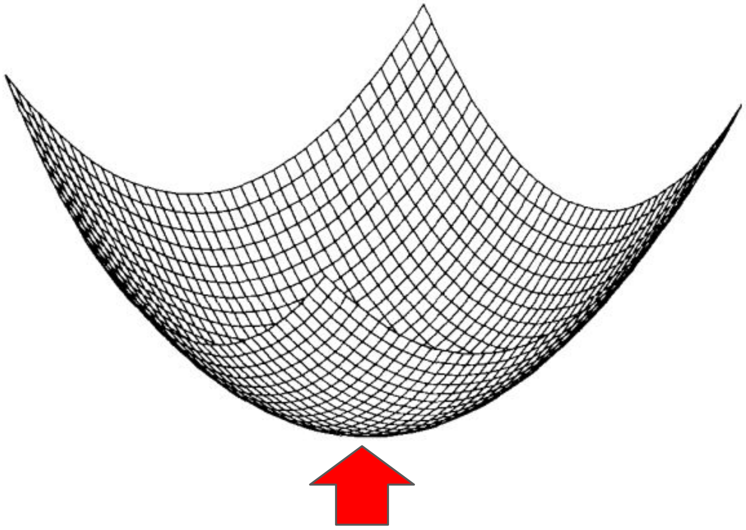


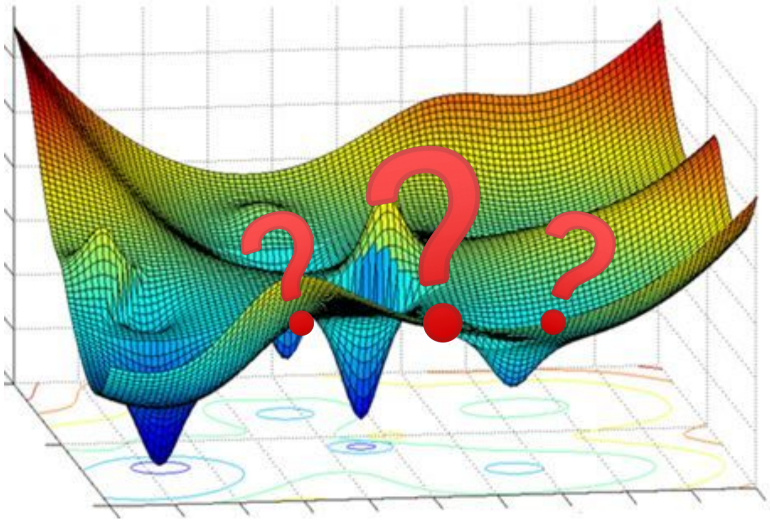
Madaline Rule II: Adaptation

- The goal is to reduce Hamming error
- If the network produces an error, adapt first layer
  - Trial adaptation: inverting its binary output
  - Done without adaptation: add a perturbation Δs
  - If output error is reduced, remove perturbation Δs and adapt selected Adalines by **α**-LMS
  - If the error is not reduces, no weight adaptation
- Exhaust all Adalines in first layer
- Repeat for all layers

Steepest Descent Rules: Single Threshold Element

Steepest Descent Rules - Motivation





Steepest Descent Rules - The μ-LMS algorithm

- Crude gradient used in place of the actual gradient
- Instantaneous gradient determined from a single input pattern

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k - \mu \boxed{\frac{\partial \epsilon_k^2}{\partial \mathbf{W}_k}}.$$

- μ is the learning constant that determines stability and convergence rate
- For the mean-squared error, this equation is:

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial \epsilon_k}{\partial \mathbf{W}_k}$$

Steepest Descent Rules - α-LMS and μ-LMS Comparison

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \alpha \frac{\epsilon_k \mathbf{X}_k}{|\mathbf{X}_k|^2}.$$

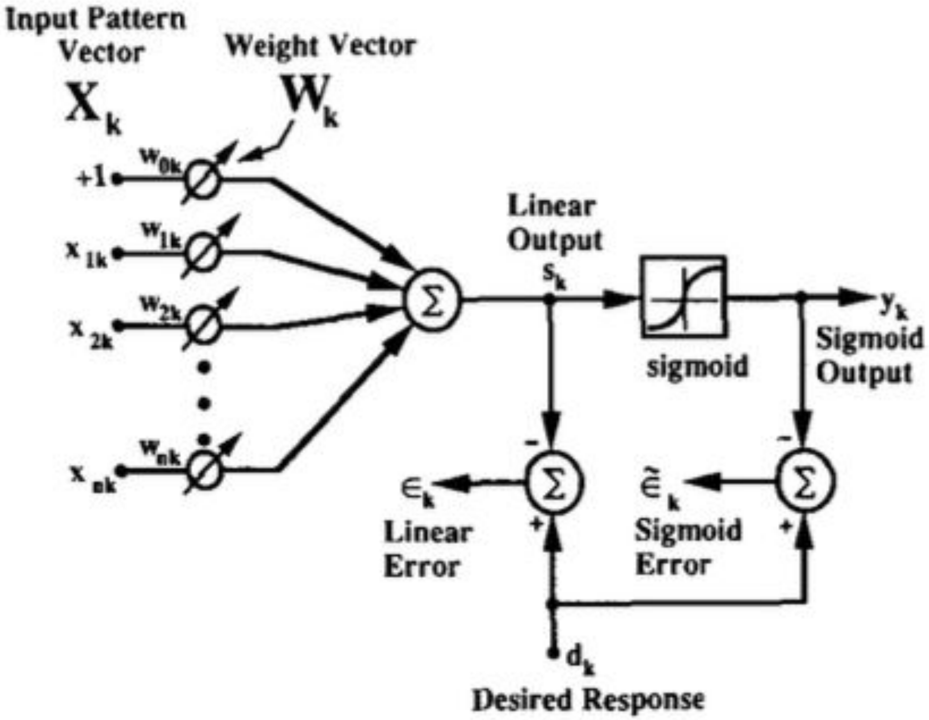
α-LMS

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\epsilon_k \mathbf{X}_k.$$

μ-LMS

LMS instantaneous gradient	LMS instantaneous gradient
self-normalizing	not self-normalizing
more difficult to analyze	easier to analyze
faster convergence	slower convergence
may converge to biased point	always converges in mean to minimum

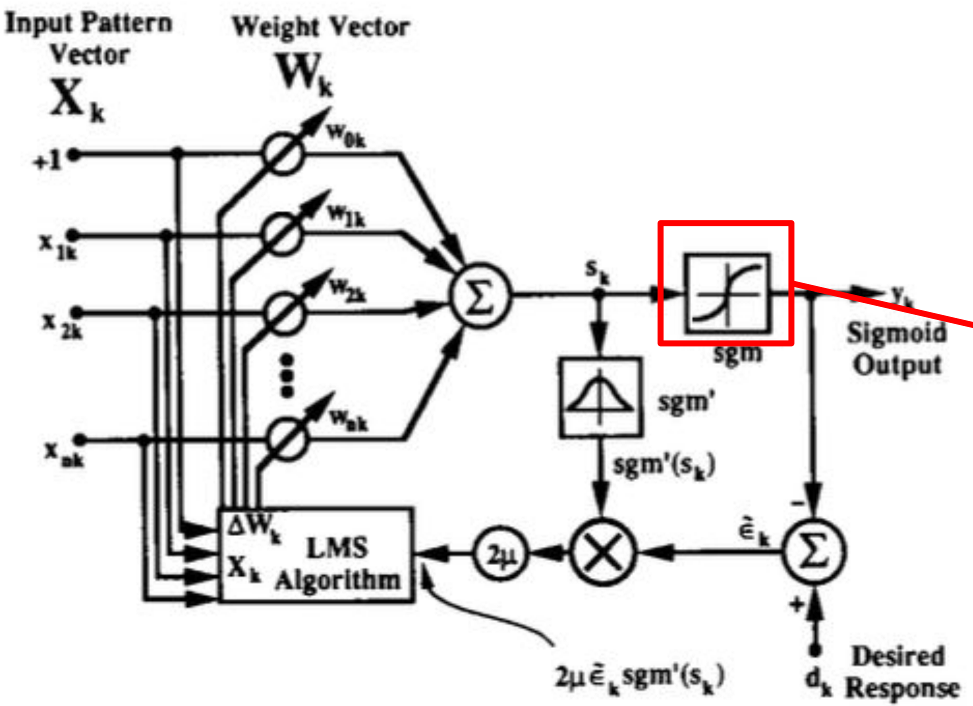
Steepest Descent Rules - Adaline with Sigmoid



$$y_k = \tanh(s_k) = \left(\frac{1 - e^{-2s_k}}{1 + e^{-2s_k}}\right).$$

$$\tilde{\epsilon}_k \triangleq d_k - y_k = d_k - \mathrm{sgm}(s_k).$$

Steepest Descent Rules - Backpropagation

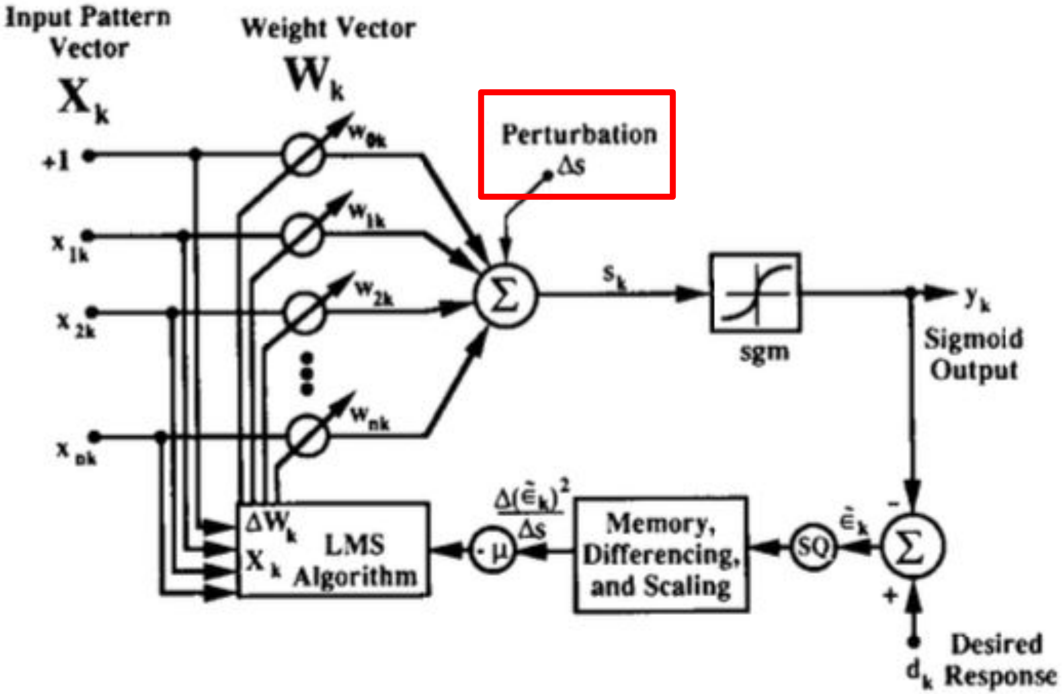


$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k + 2\mu\tilde{\epsilon}_k \,\mathrm{sgm}'(s_k)\,\mathbf{X}_k.$$

$$\mathrm{sgm}'(s_k) = \frac{\partial(\tanh(s_k))}{\partial s_k} = 1 - (\tanh(s_k))^2 = 1 - y_k^2.$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\tilde{\epsilon}_k(1 - y_k^2)\,\mathbf{X}_k.$$

Steepest Descent Rules - MRIII Algorithm



$$
\begin{aligned}
W_{k+1} &= W_k \boxed{+ \mu(-\hat{\nabla}_k)} \\
&= W_k + 2\mu \tilde{\epsilon}_k \,\mathrm{sgm}'(s_k)\, X_k.
\end{aligned}
$$

$$
W_{k+1} = W_k \boxed{- \mu \left(\frac{\Delta(\tilde{\epsilon}_k)^2}{\Delta s} \right)} X_k
$$

Steepest Descent Rules - Nonlinear Elements

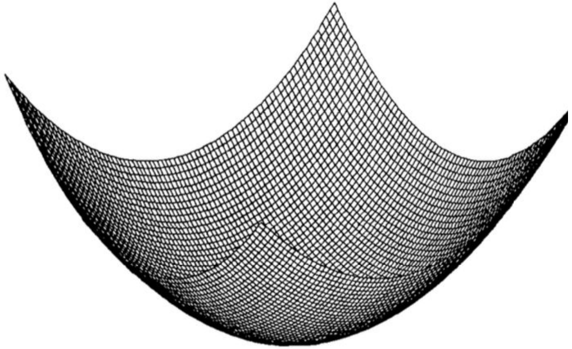


Fig. 22. Example MSE surface of linear error.

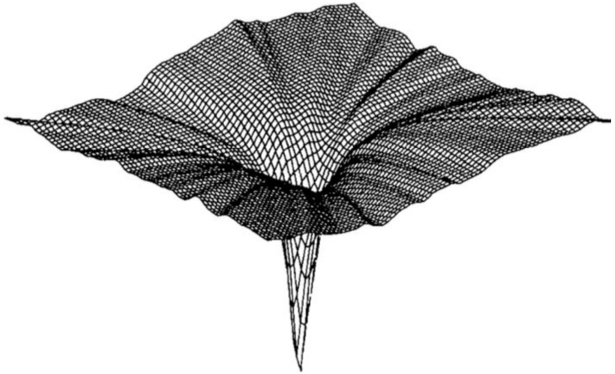


Fig. 23. Example MSE surface of sigmoid error.

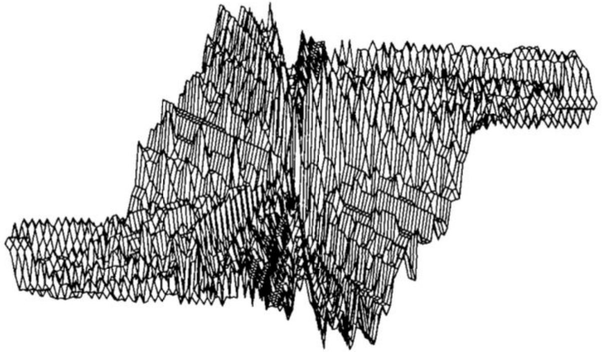


Fig. 24. Example MSE surface of signum error.

$$\text{linear error} = \epsilon = d - s$$

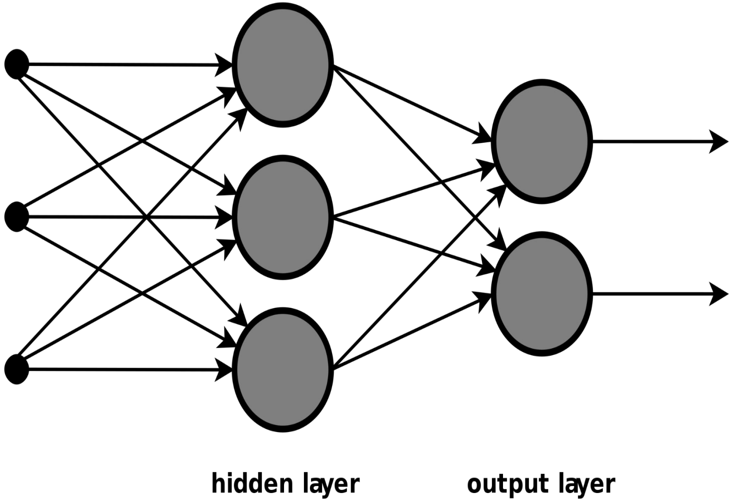
$$\text{sigmoid error} = \tilde{\epsilon} = d - \mathrm{sgm}\,(s)$$

$$\begin{aligned}\text{signum error} = \tilde{\tilde{\epsilon}} &= d - \mathrm{sgn}\,(\mathrm{sgm}\,(s)) \\ &= d - \mathrm{sgn}\,(s).\end{aligned}$$

Steepest Descent Rules - Multi-Element Networks

Backpropagation

- Process:
  - Input → Output
  - Find errors of the output
  - Sweep the effects of the errors backwards through the network to associate a “squared error derivative” (δ) with each Adaline
  - Use the δ to determine the gradient
  - Update the weights based on the gradient
  - Repeat for all layers



Multi-Element Networks: Madaline Rule III

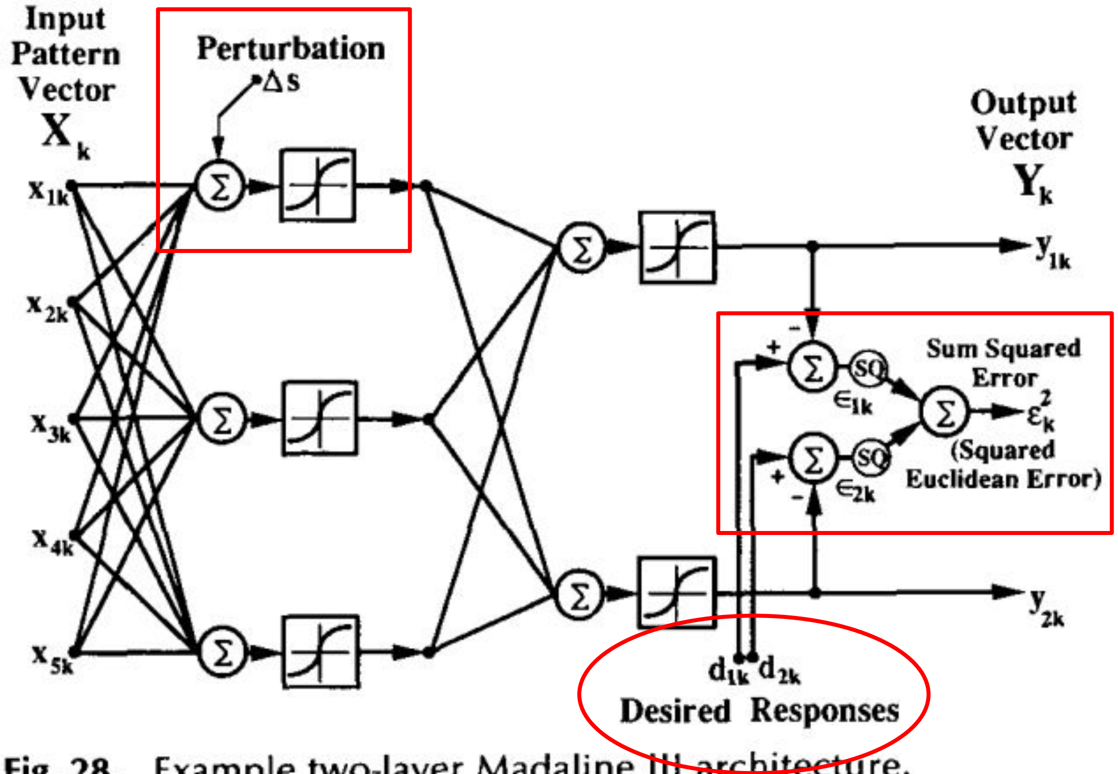


Fig. 28. Example two-layer Madaline III architecture.

MRIII vs Backpropagation

- MRIII essentially equivalent to Backpropagation
  - Same arguments for each of the Adaline elements
  - The Δs, perturbation, is small
  - Adaption is applied to all elements in the network at once

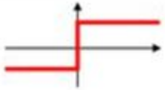
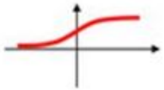
MRII vs. MRIII

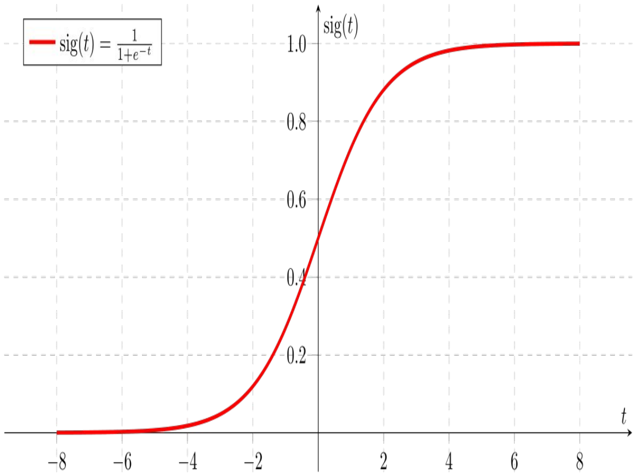
MRII:

- Discontinuous and nonlinear
- Not possible to use instantaneous gradients to update weights
- Problems with running into local minima

MRIII:

- All Adalines are adapted
    - Those with analog sums closest to zero are usually adapted stronger

Activation function	Equation	Example	1D Graph
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	



What is still being used today

- Different methods are best in different applications
- Used Less:
  - Linear Classification
  - Single elements
  - Single-layer
- Used More:
  - Multi-layer
  - Multi-element
  - Backpropagation

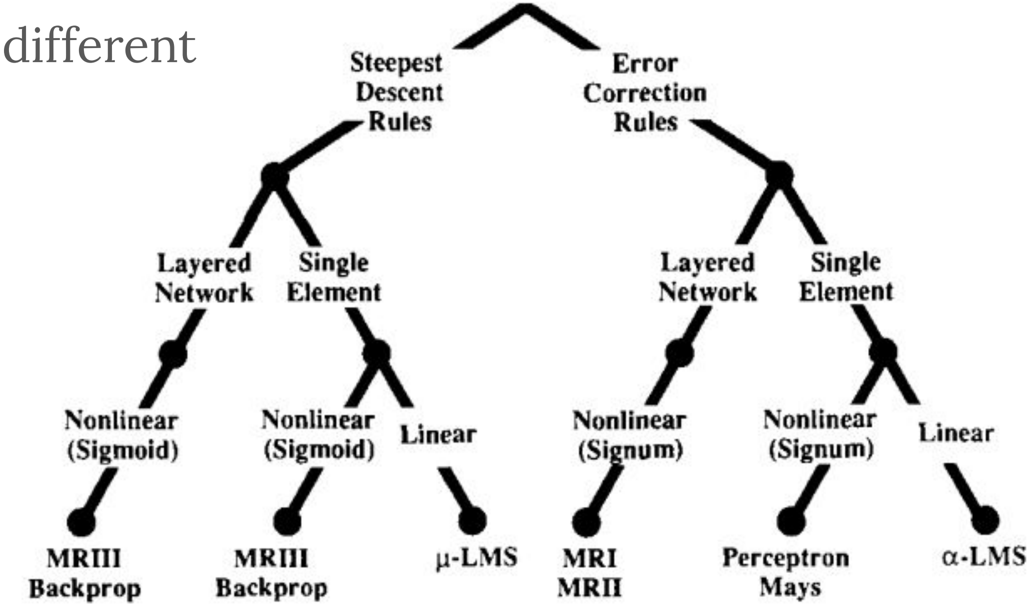


Fig. 33. Learning rules.

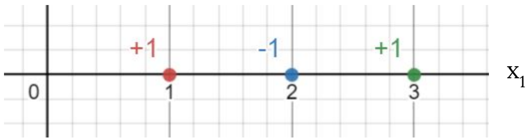
Questions?

Appendix

Nonlinear Classifiers: Feature Maps

- Consider the following example

Original Space: 1D

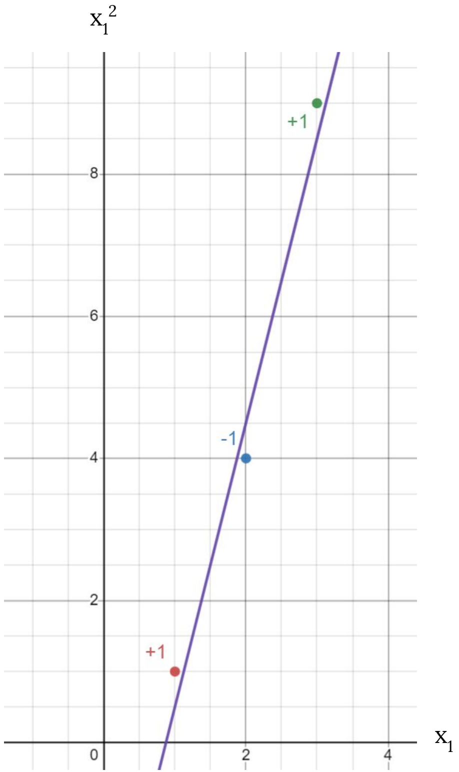


Points: 1, 2, 3

Mapped Space: 2D

Mapped Points:
- $1 \rightarrow (1, 1)$
- $2 \rightarrow (2,4)$
- $3 \rightarrow (3,9)$

Linear boundary: $x_1^2 - 4x_1 + 3.5$



Feature Map: $\mathbf{x} = [x_1] \mapsto \phi(\mathbf{x}) = \mathbf{x} = [x_1, x_1^2]$

Multi-Element Networks: Backpropagation

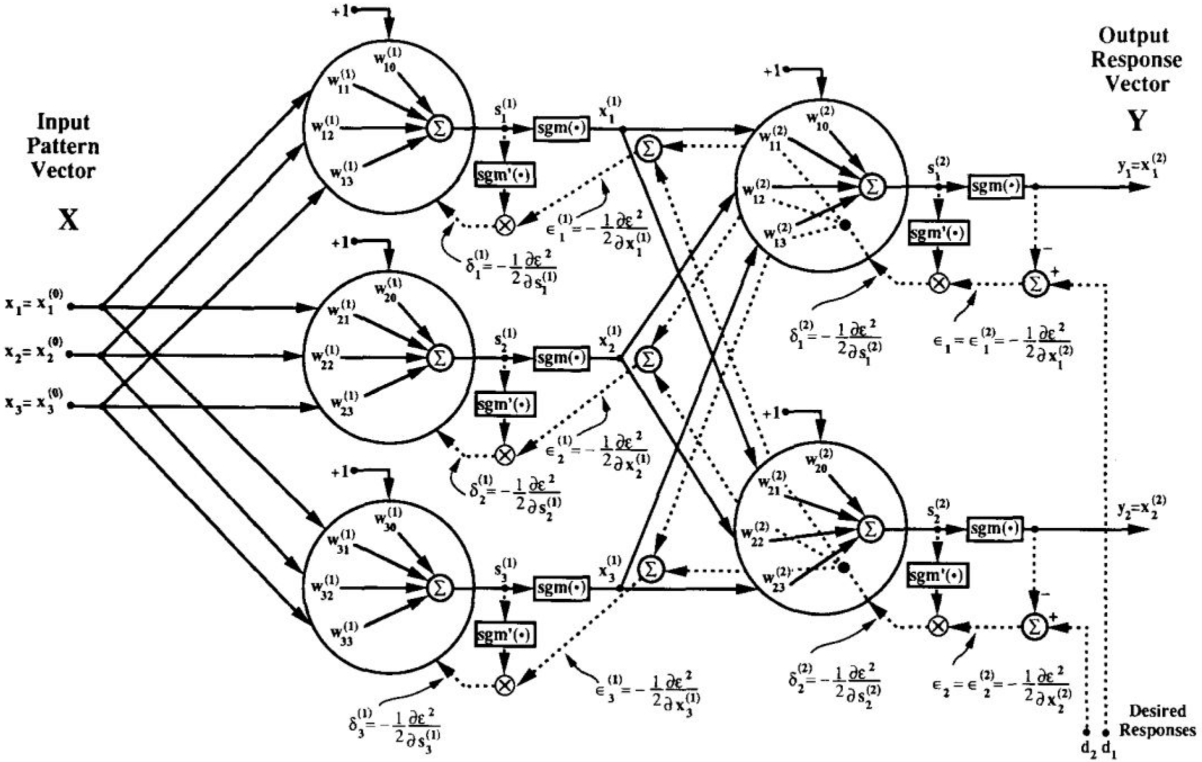
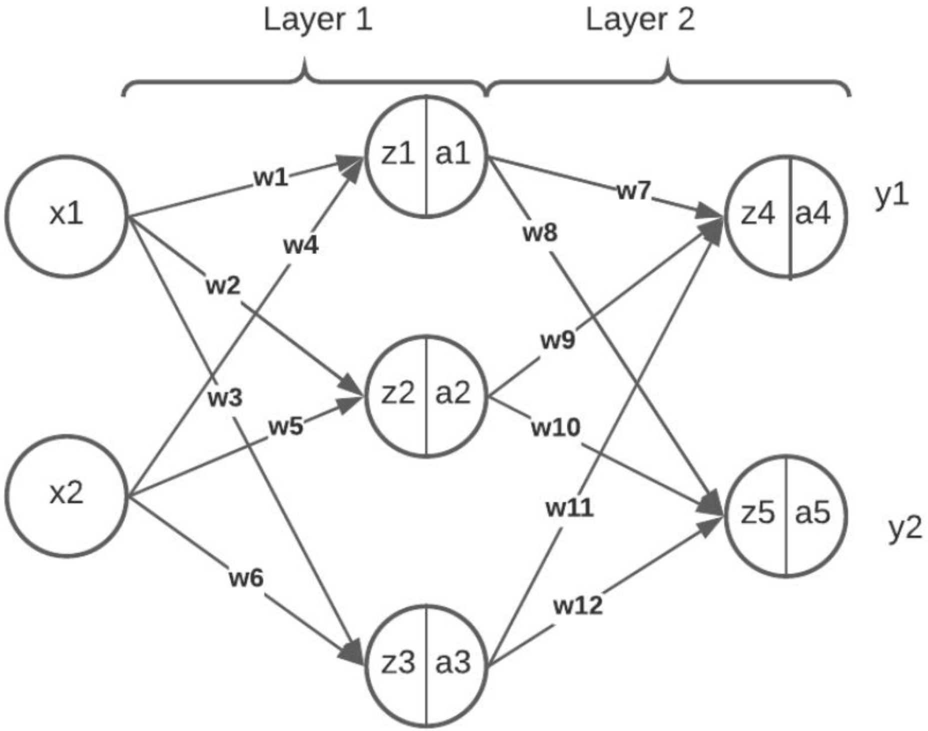


Fig. 25. Example two-layer backpropagation network architecture.

Backpropagation Example

$$
\begin{array}{cccccc}
w^{[1]} & z^{[1]} & a^{[1]} & w^{[2]} & z^{[2]} & a^{[2]} \\
\begin{bmatrix} w1 & w4 \\ w2 & w5 \\ w3 & w6 \end{bmatrix} &
\begin{bmatrix} z1 \\ z2 \\ z3 \end{bmatrix} &
\begin{bmatrix} a1 \\ a2 \\ a3 \end{bmatrix} &
\begin{bmatrix} w7 & w9 & w11 \\ w8 & w10 & w12 \end{bmatrix} &
\begin{bmatrix} z4 \\ z5 \end{bmatrix} &
\begin{bmatrix} a4 \\ a5 \end{bmatrix}
\end{array}
$$



$$\frac{\partial z_4}{\partial w_7} = \frac{\partial z_5}{\partial w_8} = a_1$$

$$dz^{[2]} = da^{[2]} \odot g'(z^{[2]})$$

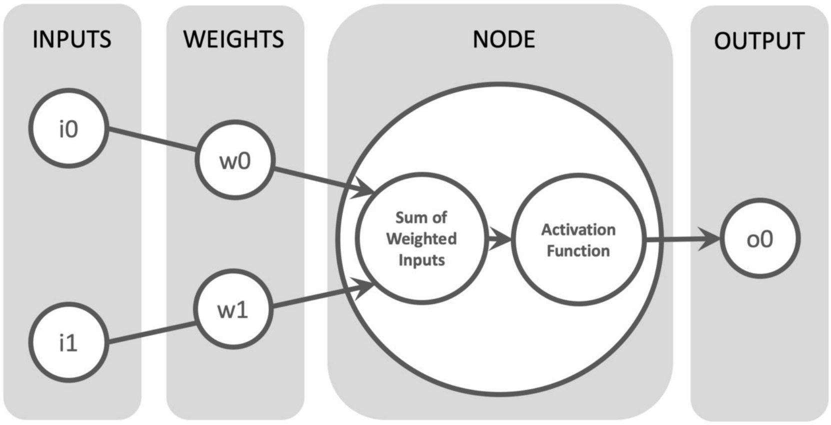
$$dw^{[2]} = dz^{[2]} \cdot (a^{[1]})^T$$

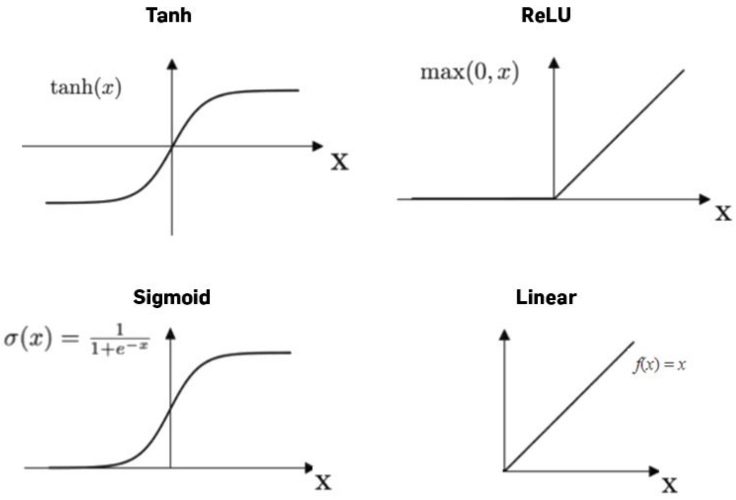
$$da^{[1]} = (w^{[2]})^T \cdot dz^{[2]}$$

$$\boxed{dz^{[1]} = da^{[1]} \odot g'(z^{[1]})}$$

Nonlinear Classifiers: Feedforward Networks

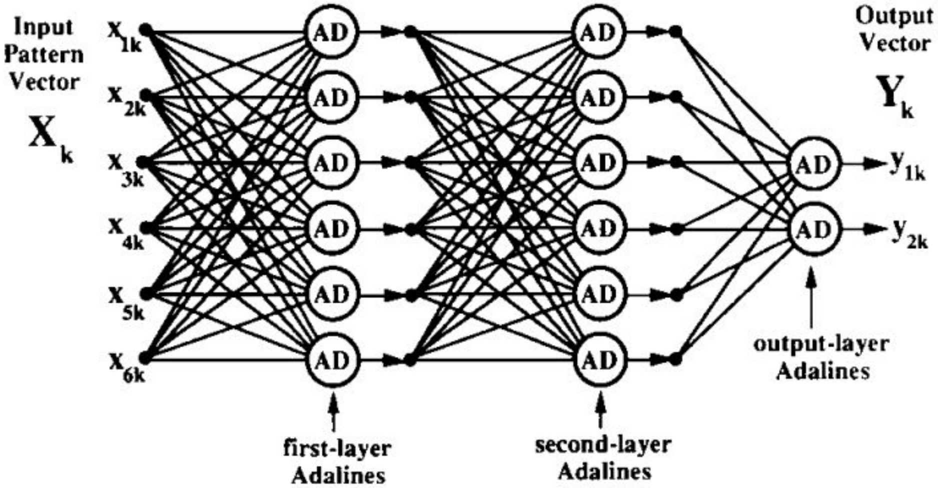
- Different layers can have different activation functions



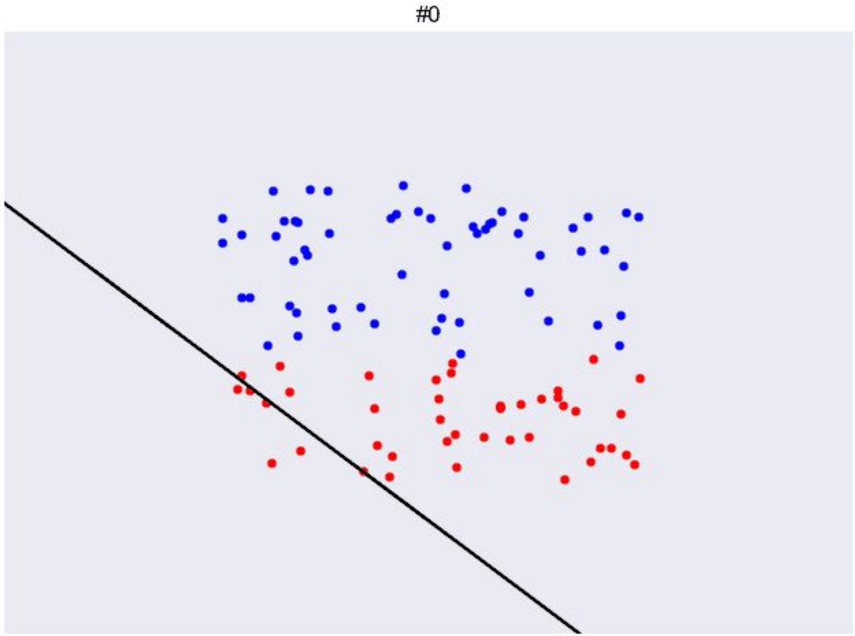


Nonlinear Classifiers: Adaptation

- Can compute error signal at output vector
- Adaptation of output layer possible
- What do we do about adaptation of hidden layers?



EC Single Threshold Element: Perceptron Rule



Steepest Descent Rules - The μ-LMS algorithm

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\nabla_k)$$

$$= (d_k - \mathbf{X}_k^T\mathbf{W}_k)^2$$

$$= d_k^2 - 2d_k\mathbf{X}_k^T\mathbf{W}_k + \mathbf{W}_k^T\mathbf{X}_k\mathbf{X}_k^T\mathbf{W}_k.$$

$$E[\epsilon_k^2]_{W=W_k} = E[d_k^2] - 2\overbrace{\boxed{E[d_k\mathbf{X}_k^T]}}^{P^T}\mathbf{W}_k + \mathbf{W}_k^T\underbrace{\boxed{E[\mathbf{X}_k\mathbf{X}_k^T]}}_{R}\mathbf{W}_k.$$

$$\nabla_k \triangleq \left\{\begin{matrix} \dfrac{\partial E[\epsilon_k^2]}{\partial w_{0k}} \\ \vdots \\ \dfrac{\partial E[\epsilon_k^2]}{\partial w_{nk}} \end{matrix}\right\}_{W=W_k} = -2\mathbf{P} + 2\mathbf{R}\mathbf{W}_k. \quad \Rightarrow \quad \mathbf{W}^* = \mathbf{R}^{-1}\mathbf{P}.$$

Steepest Descent Rules - The μ-LMS algorithm

$$\mathbf{W}_{k+1} = \mathbf{W}_k + \mu(-\hat{\nabla}_k) = \mathbf{W}_k - \mu \frac{\partial \epsilon_k^2}{\partial \mathbf{W}_k}.$$

$$\mathbf{W}_{k+1} = \mathbf{W}_k - 2\mu\epsilon_k \frac{\partial \epsilon_k}{\partial \mathbf{W}_k}$$

$$= \mathbf{W}_k - 2\mu\epsilon_k \overset{X_k}{\boxed{\frac{\partial(d_k - \mathbf{W}_k^T\mathbf{X}_k)}{\partial \mathbf{W}_k}}}.$$

$$\hat{\nabla}_k = \frac{\partial \epsilon_k^2}{\partial \mathbf{W}_k} = \begin{Bmatrix} \dfrac{\partial \epsilon_k^2}{\partial w_{0k}} \\ \vdots \\ \dfrac{\epsilon_k^2}{\partial w_{nk}} \end{Bmatrix}.$$

- The learning constant μ determines stability and convergence rate
- $0 < \mu < \dfrac{1}{\text{trace } [\mathbf{R}]}$, the algorithm converges mean to W^*

$$\Rightarrow \quad \mathbf{W}_{k+1} = \mathbf{W}_k + 2\mu\epsilon_k\mathbf{X}_k.$$