

Back-Propagation

uses LMS error methods

$$E(W) = (YD - Y)^T (YD - Y)$$

where YD = desired output

Y = actual output

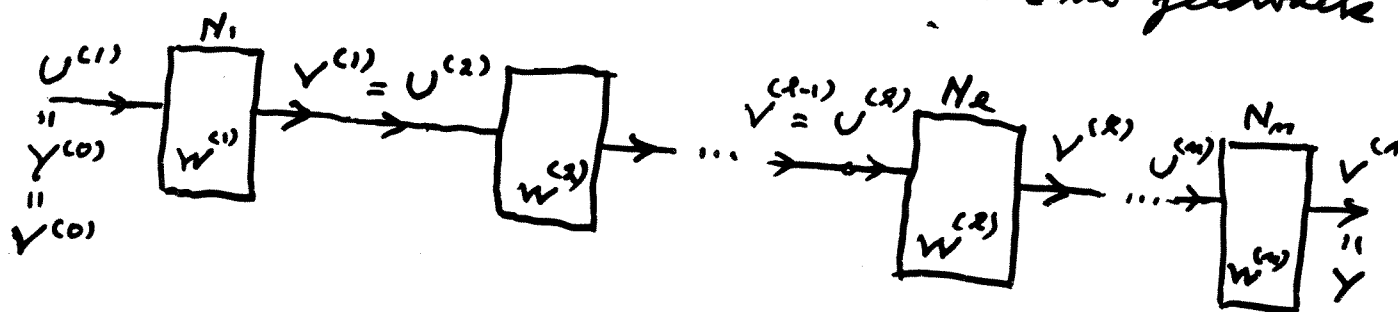
update weights by

$$W = W_0 - \lambda \nabla_W E(W_0) \quad ; \quad 0 < \lambda < 1$$

where $\lambda = \mu E(W_0) / \|\nabla_W E(W_0)\|^2$

can be taken constant (by choice of μ)

For a layered structure with no feedback



$$x_i^{(l)} = \sum_{j \in I(l)} w_{ij}^{(l)} u_j^{(l)}$$

$l = 1, \dots, n$

$$v_i^{(l)} = f_i^{(l)}(x_i^{(l)})$$

$$\begin{aligned} \Delta w_{ij}^{(l)} &= -\lambda \frac{\partial E}{\partial w_{ij}^{(l)}} = \lambda \left(-\frac{\partial E}{\partial x_i^{(l)}} \right) \cdot \frac{\partial x_i^{(l)}}{\partial w_{ij}^{(l)}} \\ &= \lambda \delta_i^{(l)} \cdot u_j^{(l)} \end{aligned}$$

(the delta rule)

in matrix form

$$\Delta W = \lambda \delta^{(l)} U^{(l)T}$$

Feb. 27/93 v.2

BP-2

$l = n$

$$\begin{aligned}\delta_i &= -\frac{\partial \mathcal{E}}{\partial x_i^{(n)}} = -\frac{\partial \mathcal{E}}{\partial v_i^{(n)}} \cdot \frac{\partial v_i^{(n)}}{\partial x_i^{(n)}} = -f_i^{(n)'}(x_i^{(n)}) \cdot \frac{\partial \mathcal{E}}{\partial v_i^{(n)}} \\ &= -f_i^{(n)'}(x_i^{(n)}) \cdot \{-2(yD_i - v_i^{(n)})\} \\ &= 2 f_i^{(n)'}(x_i^{(n)}) \{yD_i - y_i\}\end{aligned}$$

$l \neq n$

$$\begin{aligned}\delta_i^{(l)} &= -\frac{\partial \mathcal{E}}{\partial x_i^{(l)}} = -\frac{\partial \mathcal{E}}{\partial v_i^{(l)}} \cdot \frac{\partial v_i^{(l)}}{\partial x_i^{(l)}} \\ &= -f_i^{(l)'}(x_i^{(l)}) \cdot \frac{\partial \mathcal{E}}{\partial v_i^{(l)}} \\ &= f_i^{(l)'}(x_i^{(l)}) \cdot \sum_{\substack{k \text{ for} \\ N_{l+1}}} \underbrace{\frac{\partial \mathcal{E}}{\partial x_k^{(l+1)}}}_{\delta_k^{(l+1)}} \cdot \underbrace{\frac{\partial x_k^{(l+1)}}{\partial (v_i^{(l)} = u_i^{(l+1)})}}_{w_{ki}^{(l+1)}} \\ &= f_i^{(l)'}(x_i^{(l)}) \cdot \sum_{\substack{k \text{ for} \\ N_{l+1}}} \delta_k^{(l+1)} \cdot w_{ki}^{(l+1)}\end{aligned}$$

In matrix form

$$\begin{aligned}\delta^{(l)} &= \text{diag}\{f_i^{(l)'}(x_i^{(l)})\} \cdot W^{(l+1)T} \cdot \delta^{(l+1)} \\ &= F^{(l)'} \cdot W^{(l+1)T} \delta^{(l+1)}\end{aligned}$$

and

$l = 1, \dots, n-1$

$$\delta^{(n)} = 2F^{(n)'} \cdot (yD - y)$$

summary

Test. 27/93 v.3
BP-3

$$\Delta W = \lambda \delta^{(L)} U^{(L)T}$$

$$\delta^{(L)} = \begin{cases} F^{(L)'} \cdot W^{(L+1)T} \delta^{(L+1)} \\ F^{(n)'} \cdot \{2(YD - Y)\} \end{cases}$$

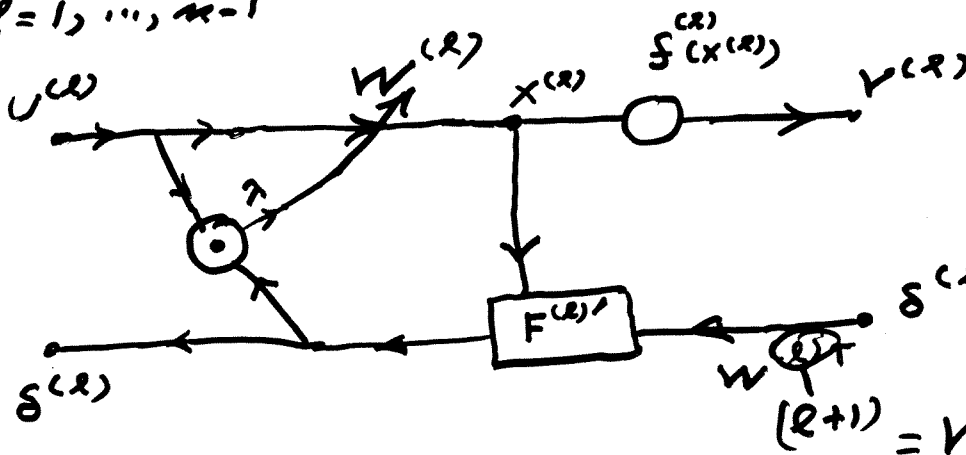
$$L = 1, \dots, n-1$$

$$L = n$$

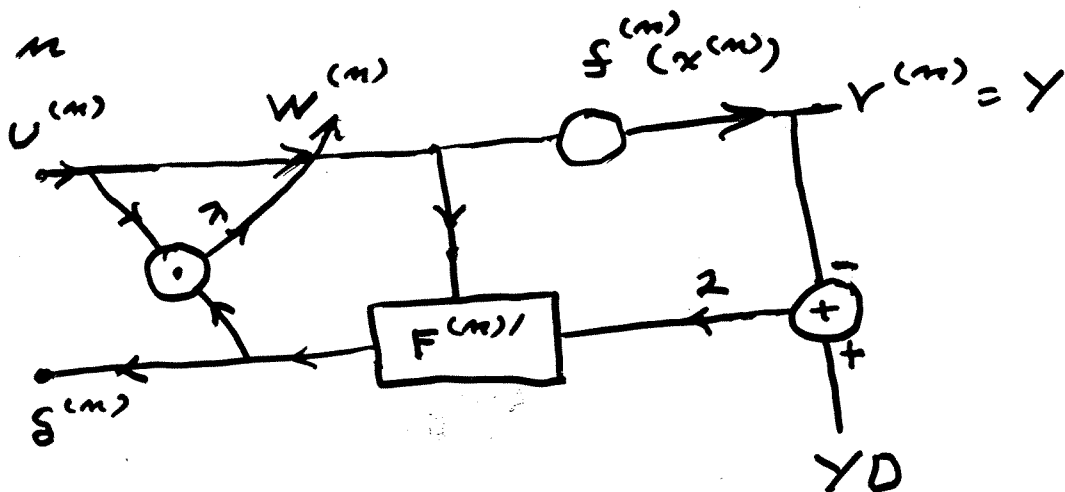
$$\text{and } F^{(L)'} = \text{diag} \{ f_i'^{(L)}(x_i^{(L)}) \}$$

signal-flow graph, l th section

$$L = 1, \dots, n-1$$



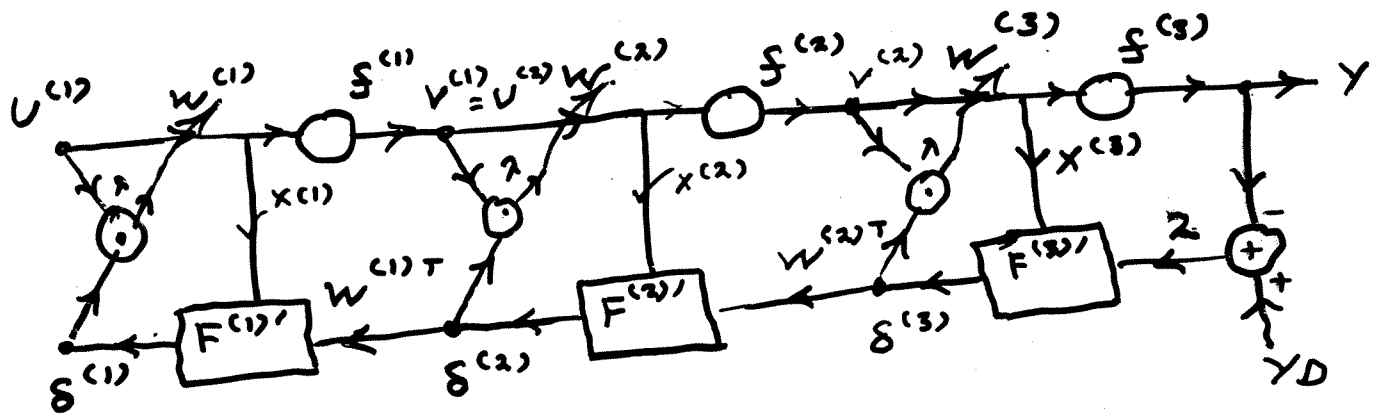
$$L = n$$



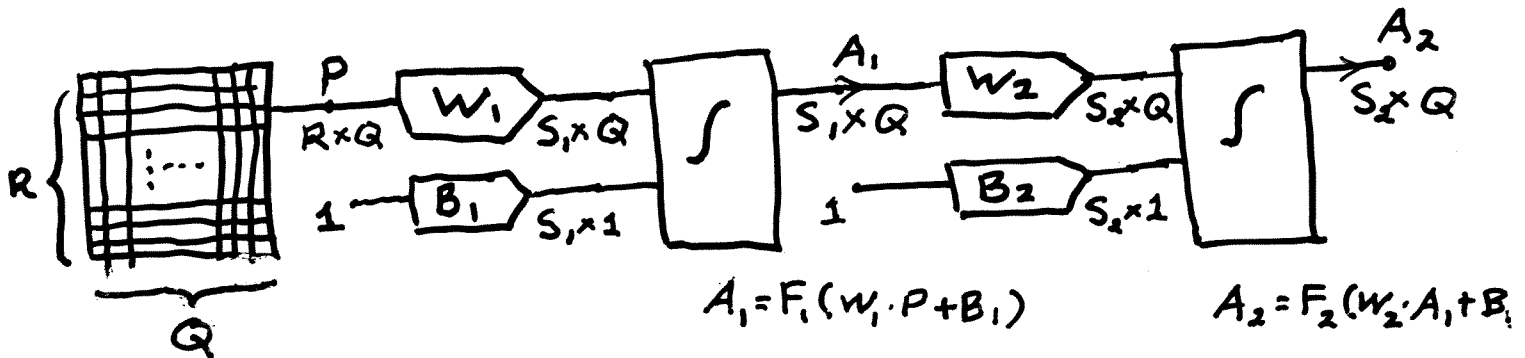
Note: No derivatives of signals need to be taken!

Feb. 27/93 P. 4
BP-4

$M = 3$ picture



Using MATLAB Back-Propagation for Two-Layer ANN



R = size of input vectors

Q = # of training (R -vector) patterns

S_1 = # of layer 1 neurons

S_2 = # of layer 2 neurons

F_1 & F_2 = layer 1 & 2 activation functions.

Uses primarily the call to `trainbp`

`[W1, B1, W2, B2, TE, TR]` (result)

`= trainbp [W1, B1, 'F1', W2, B2, 'F2', P, T, TP]`

input data to `trainbp`.

P = training patterns, input

T = target training patterns, output

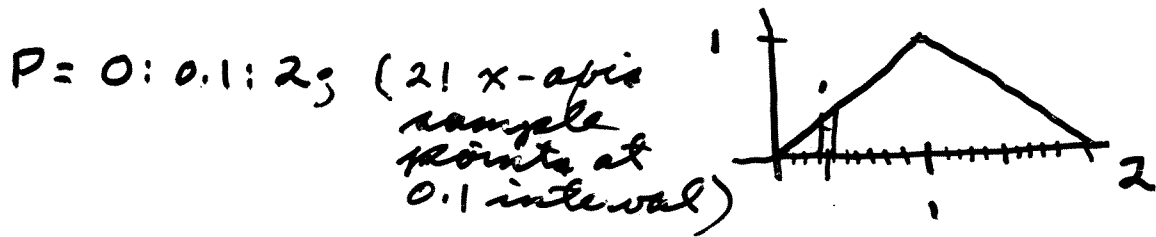
TP = training parameters

values after training

TE = # of training epochs
 TR = training record

[disp - freq max-epoch
err-goal lr]

Example: approximating a triangular wave



$T = [0 \ 0.1 \ 0.2 \ 0.3 \ 0.4 \ 0.5 \ 0.6 \ 0.7 \ 0.8 \ 0.9 \ 1 \ 0.9 \ 0.8 \ 0.7 \ 0.6 \ 0.5 \ 0.4 \ 0.3 \ 0.2 \ 0.1 \ 0];$
 (21 y-axis output at sample x-points)

$[R, Q] = \text{size}(P);$

$S1 = 5;$ (# of layer 1 neurons)

$[S2, Q] = \text{size}(T);$ (fixes # of layer 2 neurons)

$[W1, B1] = \text{randi}(S1, R);$

$[W2, B2] = \text{randi}(S2, S1);$

} randomly
initializing
weights

$A1 = \text{tansig}(W1 * P, B1);$

$A2 = \text{purelin}(W2 * A1, B2);$

$\text{disp_freq} = 10;$

$\text{max_epoch} = 500;$

$\text{err_goal} = 0.01;$

$\text{lr} = 0.02;$

$TP = [\text{disp_freq} \ \text{max_epoch} \ \text{err_goal} \ \text{lr}];$

$[W1, B1, W2, B2, TE, TR] =$

$\text{trainbp}(W1, B1, 'tansig', W2, B2, 'purelin', P, T, TP);$

plots sum-squared error vs epoch as it trains

To get output given input @ $x = 0.45$:

$A1 = \text{tansig}(W1 * 0.45, B1);$ %omit; returns