02/20/93 Singapore Lecture

Back-Propagation

uses LMS error methods

$$\mathcal{E}(W) = (YD - Y)^T (YD - Y)$$

where YD = desired output

Y = actual output

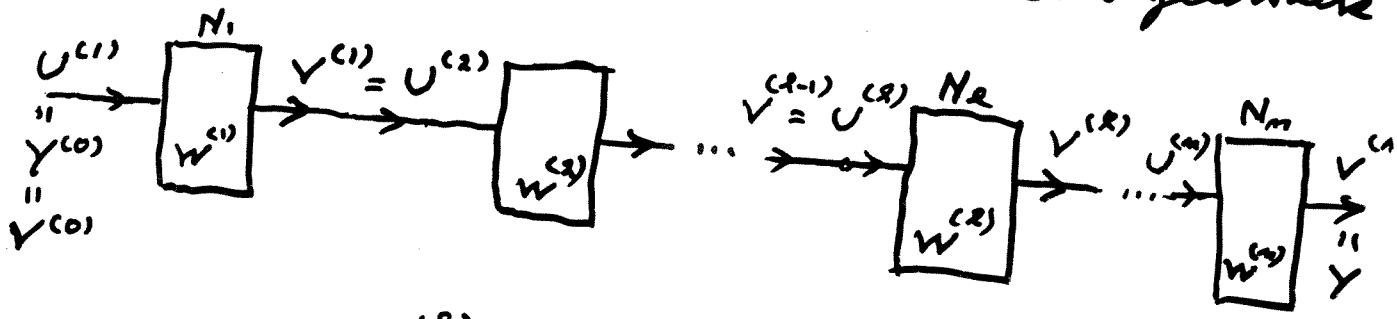
Update weights by

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

where $\lambda = N\,\mathcal{E}(W_0) / \| \nabla_W \mathcal{E}(W_0) \|^2$

can be taken constant (by choice of N)

For a layered structure with no feedback



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

$l = 1, \ldots, m$ (includes bias)

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

$$\Delta w_{ij}^{(l)} = -\lambda \frac{\partial \mathcal{E}}{\partial w_{ij}^{(l)}} = \lambda \left(-\frac{\partial \mathcal{E}}{\partial x_i^{(l)}}\right) \cdot \frac{\partial x_i^{(l)}}{\partial w_{ij}^{(l)}}$$

$$= \lambda\, \delta_i^{(l)} \cdot u_j^{(l)}$$

(the delta rule)

in matrix form

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

$l = m$

$$\delta_i = -\frac{\partial E}{\partial x_i^{(m)}} = -\frac{\partial E}{\partial v_i^{(m)}} \cdot \frac{\partial v_i^{(m)}}{\partial x_i^{(m)}} = -f_i^{(m)\prime}(x_i^{(m)}) \cdot \frac{\partial E}{\partial v_i^{(m)}}$$

$$= -f_i^{(m)\prime}(x_i^{(m)}) \cdot \{-2(yD_i - v_i^{(m)})\}$$

$$= 2 f_i^{(m)\prime}(x_i^{(m)}) \{yD_i - y_i\}$$

$l \neq m$

$$\delta_i^{(l)} = -\frac{\partial E}{\partial x_i^{(l)}} = -\frac{\partial E}{\partial v_i^{(l)}} \cdot \frac{\partial v_i^{(l)}}{\partial x_i^{(l)}}$$

$$= -f_i^{(l)\prime}(x_i^{(l)}) \cdot \frac{\partial E}{\partial v_i^{(l)}}$$

$$= f_i^{(l)\prime}(x_i^{(l)}) \cdot \sum_{k \text{ for } N_{l+1}} \underbrace{\frac{-\partial E}{\partial x_k^{(l+1)}}}_{\delta_k^{(l+1)}} \cdot \underbrace{\frac{\partial x_k^{(l+1)}}{\partial (v_i^{(l)} = u_i^{(l+1)})}}_{w_{ki}^{(l+1)}}$$

$$= f_i^{(l)\prime}(x_i^{(l)}) \cdot \sum_{k \text{ for } N_{l+1}} \delta_k \cdot w_{ki}^{(l+1)}$$

In matrix form

$$\delta^{(l)} = \text{diag}\{f_i^{(l)\prime}(x_i^{(l)})\} \cdot W^{(l+1)T} \cdot \delta^{(l+1)}$$

$$= F^{(l)\prime} \cdot W^{(l+1)T} \delta^{(l+1)}$$

and $l = 1, \ldots, m-1$

$$\delta^{(m)} = 2F^{(m)\prime} \cdot (yD - y)$$

Summary

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

$$\delta^{(l)} = \begin{cases} F^{(l)\prime} \cdot W^{(l+1)T} \delta^{(l+1)} & l = 1, \dots, n-1 \\ F^{(n)\prime} \cdot \{2(YD - Y)\} & l = n \end{cases}$$

and $F^{(l)\prime} = \operatorname{diag}\{ f_i^{(l)\prime}(x_i^{(l)}) \}$

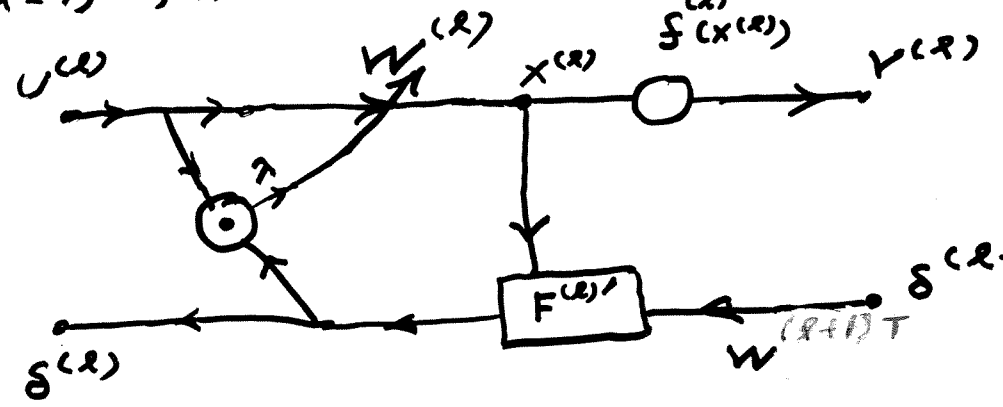
if $f(x) = \tanh x = 1 - \frac{2}{1+e^{2x}}$

$f'(x) = 1 - f^2(x) = \frac{4e^{2x}}{(1+e^{2x})^2}$

kth to $(k+1)$th iteration $W^{(l)}(k+1) = W^{(l)}(k) + \Delta W^{(l)}(k+1)$

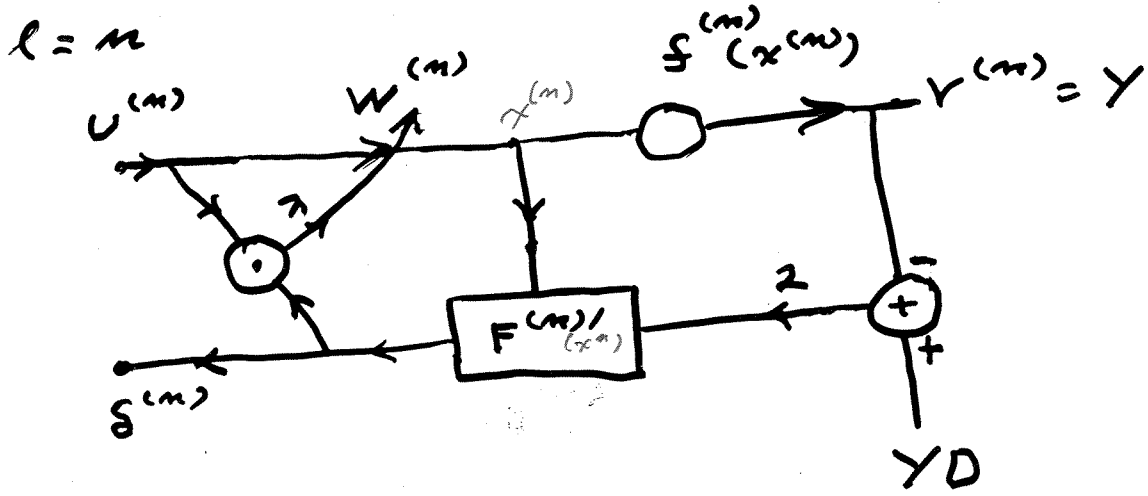
Signal-flow graph, lth section

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



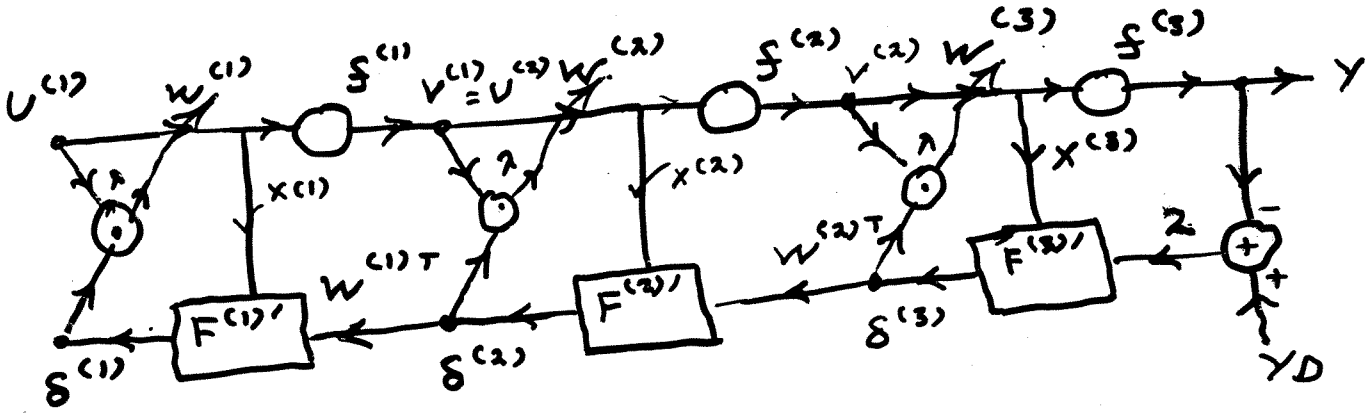
} adjoint-type of structure

$l = n$

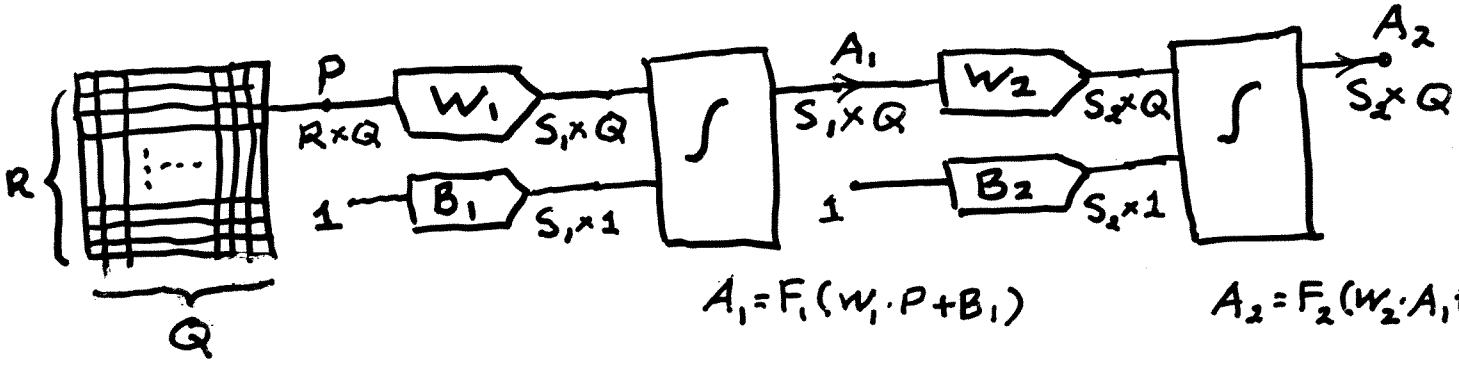


Note: No derivatives of signals need to be taken!

$m = 3$ picture


U(1)
W(1)
S(1)
V(1) = U(2)
W(2)
S(2)
V(2)
W(3)
S(3)
Y
λ
X(1)
λ
X(2)
λ
X(3)
W(1)T
W(2)T
F(1)'
F(2)'
F(3)'
2
−
+
+
δ(1)
δ(2)
δ(3)
YD

Using MATLAB Back-Propagation for Two-Layer ANN



R = size of input vector
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]

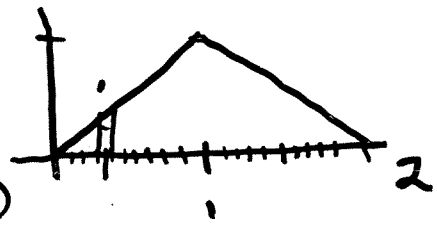
input data to trainbp.

P = training patterns, input
T = target training patterns, output
TP = training parameters

[disp_freq max_epoch err_goal lr]

values after training
TE = # of training epochs
TR = training record

Example: approximating a triangular wave




```
P = 0:0.1:2;   (21 x-axis sample points at 0.1 interval)

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] = size(P);
S1 = 5;   (# of layer 1 neurons)
[S2,Q] = size(T);   (gives # of layer 2 neurons)
[W1,B1] = rands(S1,R);   } randomly initialize weights
[W2,B2] = rands(S2,S1);  }
A1 = tansig(W1*P,B1);
A2 = purelin(W2*A1,B2);
disp_freq = 10;
max_epoch = 500;
err_goal = 0.01;
lr = 0.02;
TP = [disp_freq max_epoch err_goal lr];
[W1,B1,W2,B2,TE,TR] =
    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 = tansig(W1*0.45,B1);   % omit; returns
```