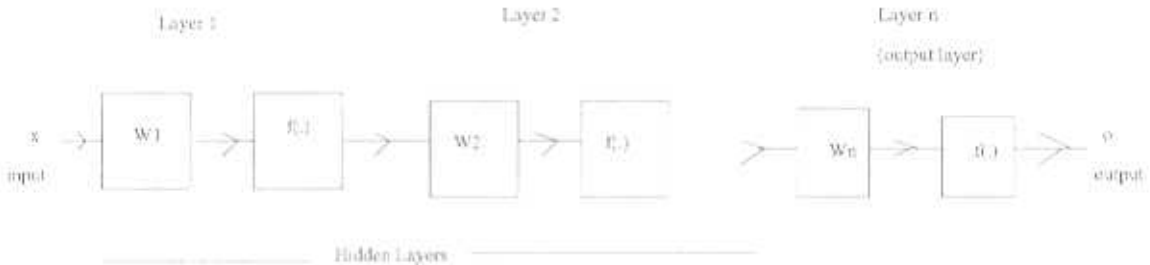
• Back Propagation Algorithm

• If we cascade single layer structures we obtain multilayer structures...



• $o = \Gamma(W_n(\Gamma(W_{n-1}(\ldots\Gamma(W_2(\Gamma(W_1 x))\ldots)$

• Usually we have a training set set $\mathcal{S}_{Tr} = \{x^1, x^2, \ldots x^N\}$ and we have a set of desired outputs $d^1, \ldots, d^N$. Note that these are **vectors**.

• Assume that at the output layer we have R neurons. Hence, we have $d, o \in \mathbf{R}^R$, i.e. they have R components.

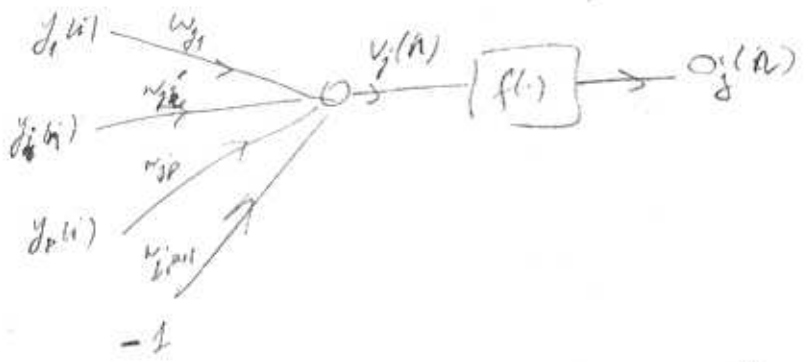
• Suppose that we applied nth training set pattern, we obtain an output $o(n)$, and the corresponding desired output is $d(n) = d^n$. Componentwise we have :

• $d(n) = \begin{pmatrix} d_1(n) \\ d_2(n) \\ \vdots \\ d_R(n) \end{pmatrix} \qquad o(n) = \begin{pmatrix} o_1(n) \\ o_2(n) \\ \vdots \\ o_R(n) \end{pmatrix} \qquad n = 1, 2, \ldots, N$

$$\underset{\sim}{d}_n = \begin{pmatrix} d_1(n) \\ d_2(n) \\ d_R(n) \end{pmatrix}, \quad \underset{\sim}{O}_n = \begin{pmatrix} o_1(n) \\ o_2(n) \\ o_R(n) \end{pmatrix}, \quad \underset{\sim}{y}_n = \begin{pmatrix} y_1(n) \\ y_2(n) \\ y_p(n) \\ 1 \end{pmatrix}$$

n: pattern number

j: output neuron



$$v_j(n) = w_{j1}\, y_1(n) + \cdots + w_{j2}\, y_2(n) + \cdots + w_{jp}\, y_p(n) - w_{j,p+1}$$

$$o_j(n) = f(v_j(n))$$

$$E(n) = \frac{1}{2} \| \underset{\sim}{d}_n - \underset{\sim}{O}_n \|^2 = \frac{1}{2} \sum_{i=1}^{R} (d_i(n) - O_i(n))^2$$

$e_i(n) = d_i(n) - o_i(n)$ → error node at i^{th} output neuron for n^{th} input pattern.

$$\rightarrow E(n) = \frac{1}{2} \sum_{j=1}^{R} e_j^2(n)$$

$$E_{ave} = \frac{1}{N} \sum_{n=1}^{N} E(n) = \frac{E(1) + E(2) + \cdots + E(N)}{N}$$

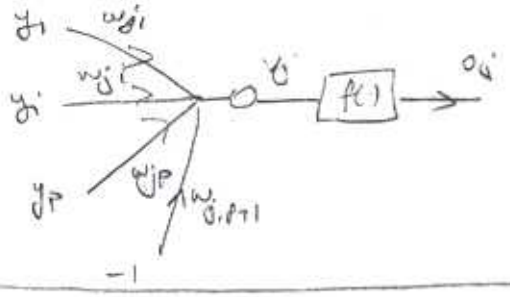
Update law for any weight:

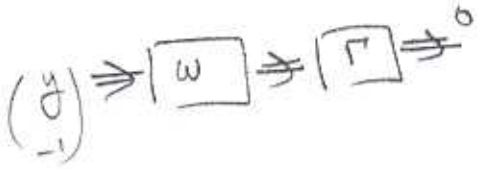
$$w \leftarrow w - \eta \underbrace{\frac{\partial E(n)}{\partial w}}_{\Delta w}$$

FOR FREQUENTLY ASKED QUESTIONS ON NEURAL NETWORKS:
ftp://ftp.sas.com/pub/neural/FAQ.html

How to evaluate $\frac{\partial E(n)}{\partial w}$?

→ Neuron j is an output neuron:





$$\frac{\partial E(n)}{\partial w_{ji}} = -f'(v_j(n)) \cdot e_j(n)\, y_i$$

$j = 1, \dots, R$
$i = 1, \dots, p+1$

Local gradient:

$$\delta_j(n) = f'(v_j(n)) \cdot e_j(n) \qquad j = 1, 2, \dots, R$$

$$\frac{\partial E(n)}{\partial w_{ji}} = -\delta_j(n)\, y_i$$

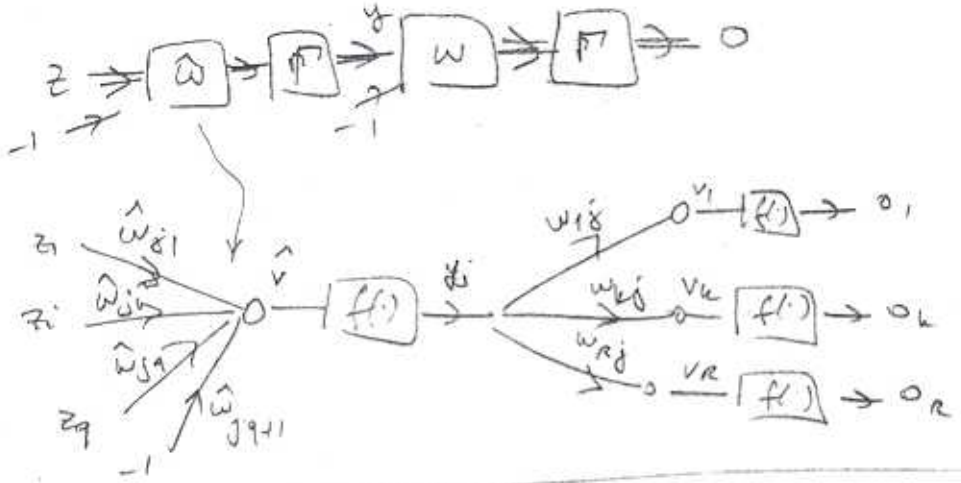
MATRIX FORMULATION:

$$\delta(n) = \begin{pmatrix} \delta_1(n) \\ \vdots \\ \delta_R(n) \end{pmatrix} = \Gamma'(v_j(n)) \cdot e(n) \quad , \quad \Gamma'(v) = \begin{bmatrix} f'(v_1) & & 0 \\ & \ddots & \\ 0 & & f'(v_R) \end{bmatrix}$$

$$w = \begin{pmatrix} w_{11} & \cdots & w_{1p} \\ \vdots & & \vdots \\ w_{R1} & \cdots & w_{Rp} \end{pmatrix} \qquad w^e = (w \mid \text{treshold}) \qquad y^e = \begin{pmatrix} y \\ -1 \end{pmatrix}$$

$$w^e \leftarrow w^e + \eta\, \delta(n)\, y^{eT}$$

→ Neuron j is hidden layer neuron:



$$\frac{\partial E(n)}{\partial \hat{w}_{ji}} = -f'(\hat{v}_j(n)) \left(\sum_{k=1}^{R} f'(v_k(n)) \cdot e_k(n) w_{kj} \right) z_i \qquad \begin{matrix} j=1,\dots,p \\ i=1,\dots,q+1 \end{matrix}$$

Local gradient:

$$\hat{\delta}_j = f'(\hat{v}_j(n)) \sum_{k=1}^{R} f'(v_k(n))\, e_k(n)\, w_{kj} \qquad j=1,\dots,p$$

$$= f'(v_j(n)) \sum_{k=1}^{R} \delta_k(n) \cdot w_{kj}$$

$$\frac{\partial E(n)}{\partial \hat{w}_{kj}} = -\hat{\delta}_j(n) \cdot z_i$$

MATRIX FORMULATION

$$\hat{\delta}(n) = \begin{pmatrix} \hat{\delta}_1(n) \\ \vdots \\ \hat{\delta}_p(n) \end{pmatrix} = \Gamma'(\hat{v}_j(n))\, W^T \cdot \delta(n)$$

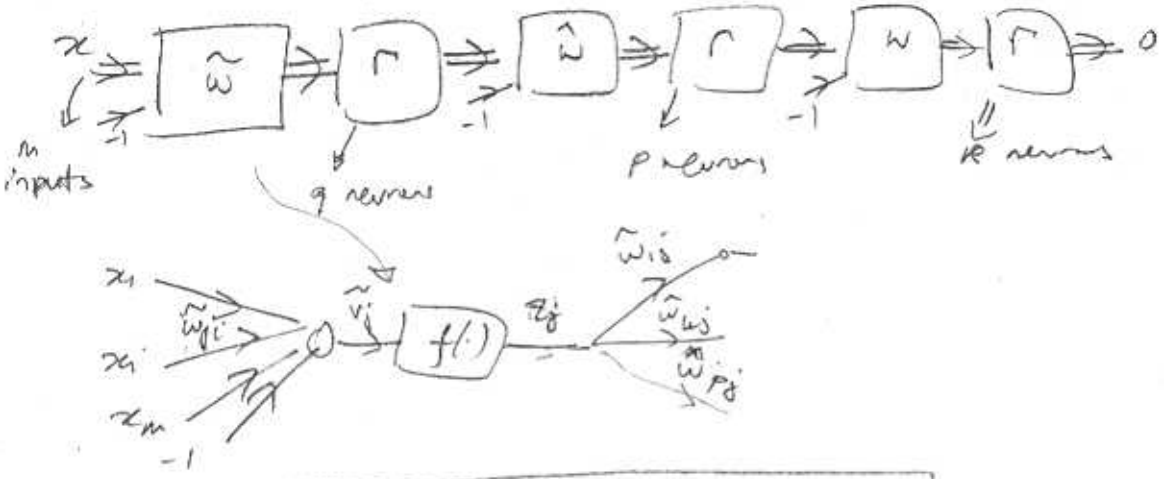
local gradient vector of following layer.

weight matrix of following layer

$$\hat{W} = \begin{pmatrix} \hat{w}_{11} & \cdots & \hat{w}_{1q} \\ \vdots & & \\ \hat{w}_{p1} & \cdots & \hat{w}_{pq} \end{pmatrix} \qquad \hat{W}^e = \left(\hat{W} \mid \text{treshold} \right) \qquad z_e = \begin{pmatrix} z \\ -1 \end{pmatrix}$$

$$\hat{W}^e \leftarrow \hat{W}^e + \eta\, \hat{\delta}(n)\, z_e^T$$

— How do extend to one more layer?



$$\frac{\partial E(n)}{\partial \tilde{w}_{ji}} = -f'(\tilde{v}_j(n)) \left(\sum_{k=1}^{p} \hat{\delta}_k(n)\, \hat{w}_{kj} \right) x_i$$

$\hat{\delta}_k(n)$ → local gradient of following layer.

$j = 1, \dots, q$
$i = 1, \dots, m+1$

LOCAL gradient:

$$\tilde{\delta}_j = f'(\tilde{v}_j(n)) \sum_{k=1}^{p} \hat{\delta}_k(n)\, \hat{w}_{kj} \qquad j = 1, \dots, q$$

$$\frac{\partial E(n)}{\partial \tilde{w}_{ji}} = -\tilde{\delta}_j(n)\, x_i$$

MATRIX FORMULATION:

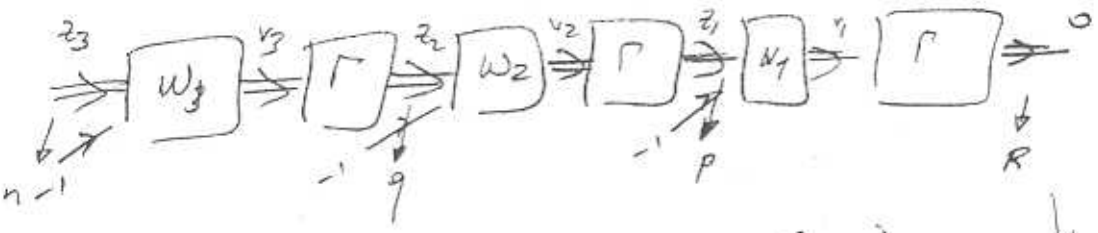
$$\tilde{\delta}(n) = \begin{pmatrix} \tilde{\delta}_1 \\ \vdots \\ \tilde{\delta}_q \end{pmatrix} = \tilde{F}'(\tilde{v})\, \hat{w}^T \cdot \hat{\delta}(n)$$

$\tilde{F}'(\tilde{v})$ ← local gradient of following layer.

$\hat{w}^T$ ← weight matrix of following layer

$$\tilde{w} = \begin{pmatrix} \tilde{w}_{11} & \cdots & \tilde{w}_{1m} \\ \vdots & & \\ \tilde{w}_{q1} & \cdots & \tilde{w}_{qm} \end{pmatrix} \qquad \tilde{w}_e = (\tilde{w} \mid \text{threshold}) \qquad x_e = \begin{pmatrix} x \\ -1 \end{pmatrix}$$

$$\tilde{w}_e \leftarrow \tilde{w}_e + \eta\, \tilde{\delta}(n) \cdot x_e^T$$



$z_3 \in \mathbb{R}^n$ $\quad z_2 \in \mathbb{R}^q$ $\quad z \in \mathbb{R}^p$ $\quad o \in \mathbb{R}^R$ $\quad d \in \mathbb{R}^R$

$z_3^e = \begin{pmatrix} z_3 \\ -1 \end{pmatrix} \in \mathbb{R}^{n+1}$ $\quad z_2^e = \begin{pmatrix} z_2 \\ -1 \end{pmatrix} \in \mathbb{R}^{q+1}$ $\quad z_1^e = \begin{pmatrix} z_1 \\ -1 \end{pmatrix} \in \mathbb{R}^{p+1}$

n: # of input components (excluding threshold)

q: # of neurons in 1st layer.

p: # of neurons in 2nd layer.

R: # of neurons in output layer.

$W_1 \in \mathbb{R}^{R \times p}$ $\quad W_1^e = [W_1 \mid \text{threshold}] \in \mathbb{R}^{R \times (p+1)}$

$W_2 \in \mathbb{R}^{p \times q}$ $\quad W_2^e = [W_2 \mid \text{threshold}] \in \mathbb{R}^{p \times (q+1)}$

$W_3 \in \mathbb{R}^{q \times n}$ $\quad W_3^e = [W_3 \mid \text{threshold}] \in \mathbb{R}^{q \times (n+1)}$

present $x(n) = z_3$ at nth step, use $W_3(n)$, $W_2(n)$, $W_1(n)$

Calculate everything till 0 using current weights.

- $e(n) = d(n) - o(n)$
- $\delta_1(n) = \Gamma'(v_1(n)) e(n) = \begin{bmatrix} f'(v_{11}(n)) & & \\ & f'(v_{12}(n)) & \\ & & f'(v_{1R}(n)) \end{bmatrix} \begin{pmatrix} e_1(n) \\ \vdots \\ e_R(n) \end{pmatrix}$ (R×R) (R×1)
- $W_1^{e(n+1)} \leftarrow W_1^e(n) + \eta\, \delta_1(n) (z_1^e)^T$
- $\delta_2(n) = \Gamma'(v_2(n)) \; W_2^T(n) \; \delta_1(n)$ (p×p) (p×R) (R×1) = p×1
- $W_2^e(n+1) \leftarrow W_2^e(n) + \eta\, \delta_2(n) (z_2^e)^T$
- $\delta_3(n) = \Gamma'(v_3(n)) \cdot W_2^T(n) \cdot \delta_2(n)$ (q×q) (q×p) (p×1) = q×1
- $W_3^e(n+1) \leftarrow W_3^e(n) + \eta\, \delta_3(n) (z_3^e)^T$
- pick up another pattern as x. continue next step.

TRAINING ERRORS:

Training set: $\{z_1, \dots, z_N\}$, $o = \begin{pmatrix} o_1 \\ \vdots \\ o_R \end{pmatrix}$ $d = \begin{pmatrix} d_1 \\ \vdots \\ d_R \end{pmatrix}$

error at n^{th} pattern $\Rightarrow$ $E(n) = \frac{1}{2} \|o(n) - d(n)\|^2 = \frac{1}{2} \sum_{i=1}^{R} (o_i(n) - d_i(n))^2$

cumulative error $\Rightarrow$ $E_{cum} = \sum_{n=1}^{N} E(n)$

average error $\Rightarrow$ $E_{ave} = \frac{1}{N} E_{cum}$.

if # of output neurons (R) is large $\Rightarrow$ Root-Mean-Square error:

$$E_{rms} = \frac{1}{NR} \sqrt{2 E_{cum}}$$

$\Rightarrow$ maybe useful for comparing two NN architectures with different N and R.

In Binary case ($o_i = \mp 1$) / ($o_i = 1, 0$)

$\Rightarrow$ $e_i = \begin{cases} 1 & \text{if output } i \text{ is incorrect (bit error)} \\ 0 & \text{if output } i \text{ is correct} \end{cases}$

$E(n) = \sum_{i=1}^{R} e_i$ (bit error at output)

$E_{cum} = \sum_{n=1}^{N} E(n)$ (cumulative bit error)

$E_{ave} = \frac{1}{N} E_{cum}$ (average error)

$E_{rms} = \frac{1}{NR} \sqrt{2 E_{cum}}$ (RMS error)

$E_{dc} = \frac{E_{cum}}{NR}$ (decision error)

— MOMENTUM METHOD —

$$E(w+\Delta w) = E(w) + \frac{\partial E}{\partial w} \cdot \Delta w + h.o.t$$

$$\Delta w = -\eta \frac{\partial E}{\partial w}$$

- if η is low method works, but error descent may be slow.
- if η is high error descent may be high but method may not be as w

Momentum method: A method to (possibly) accelerate the descent without instability

$$\Delta w(n) = \alpha \Delta w(n-1) - \eta \frac{\partial E(n)}{\partial w}$$

α = momentum constant (forgetting factor)

- Acts like a filter:

$\Rightarrow$ z transform:

$$\Delta W(z) = \alpha z^{-1} \Delta W(z) - \eta Z(\nabla E)$$

↙ z transform of error gradient:

$$\Delta W(z) = \frac{1}{1-\alpha z^{-1}} \left(-\eta Z(\nabla E)\right)$$

↳ acts like a low pass filter

$$-\eta \nabla E \Rightarrow \boxed{\frac{1}{1-\alpha z^{-1}}} \Rightarrow \Delta w$$

For stability: $|\alpha| < 1$

$$\Delta w(1) = -\eta \frac{\partial E(1)}{\partial w}$$

$$\Delta w(2) = \alpha \Delta w(1) - \eta \frac{\partial E(2)}{\partial w} = -\alpha\eta \frac{\partial E(1)}{\partial w} - \eta \frac{\partial E(2)}{\partial w}$$

$$\Delta w(3) = \alpha \Delta w(2) - \eta \frac{\partial E(3)}{\partial w} = -\alpha^2\eta \frac{\partial E(1)}{\partial w} - \alpha\eta \frac{\partial E(2)}{\partial w} - \eta \frac{\partial E(3)}{\partial w}$$

$$\boxed{\Delta w(n) = -\eta \sum_{k=1}^{n} \alpha^{n-k} \frac{\partial E(k)}{\partial w}}$$ ⇒ weighted average of error gradients

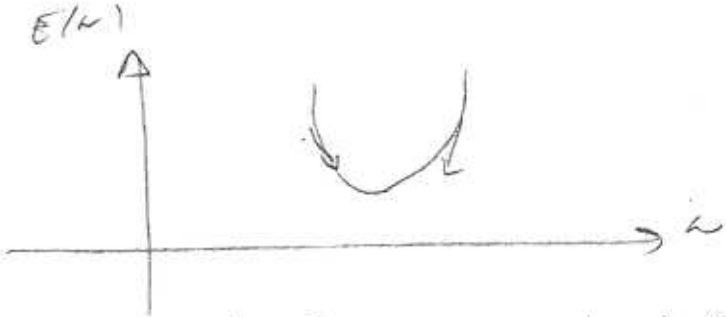
- $0 < \alpha < 1$ for stability and accelerating the descent.
- when $\frac{\partial E(k)}{\partial w}$ has the same algebraic sign over consecutive steps ⇒

← some downhill

⇒ weight increment is increased

→ accelerating the descent.

- when $\frac{\partial E(k)}{\partial w}$ has alternating signs over consecutive steps →

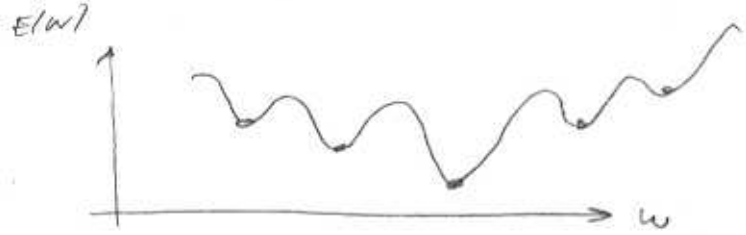


→ danger of instability (oscillations) ⇒ weight change is decreased.

— PROBLEM of LOCAL MINIMUM.

— Convergence is never guaranteed.

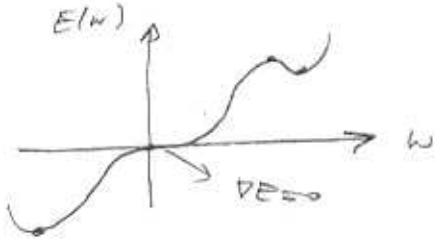
— If η is sufficiently small and if $w(0)$ is sufficiently close to a local minimum, then algorithm converges to the local minimum.



— No guarantee to converge to a global minimum

— "sufficiently small" "sufficiently close"

— Saddle point: extremum ($\nabla E = 0$) but neither minimum nor maximum.



— If is local minimum, impossible to get out by using deterministic approach.

— Start with different initial conditions.

— Introduce some randomness to the algorithm.

— Shuffle the training patterns

— Add zero mean white noise to the training patterns.

- Weight Initialization

- Prevent the neurons from saturation.

$f'(v) = (1-o^2) \Rightarrow$ if $o \sim \pm 1$, $f'(v)$ is small $\Rightarrow$ small update.

- Choose the weights randomly in a small range around the origin (try to keep the output in linear range)

- $\left(-\frac{2.4}{F_j}, \frac{2.4}{F_j}\right)$ a typical range for the initial weights of j^{th} neuron

- Wrong choice of initial conditions may cause "premature saturation"

- Correct saturation, incorrect saturation.

- If a neuron is in incorrect saturation, very difficult to get out of it. $\Rightarrow$ saddle behavior.

- STOPPING CRITERIA

typical: if $Error < \epsilon$ stop
else start a new epoch.

- Error: $\rightarrow E_{cum}$, E_{ave}, E_{rms}, E_{dc} ...

- check ∇E. stop when $\|\nabla E\| < \epsilon$

- Near minimum, error is stationary.

Let q be epoch index.

if $\frac{|E_{cum}(q) - E_{cum}(q-1)|}{E_{cum}(q-1)} < \epsilon$ stop