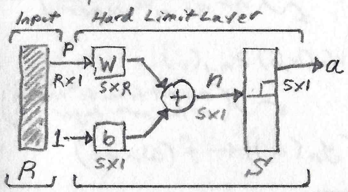
Perceptron:



$$W = \begin{bmatrix} w_1^T \\ w_2^T \\ \vdots \\ w_S^T \end{bmatrix}$$

$a = \text{hardlim}(Wp+b)$

$\left(a_i = \text{hardlim}(n_i) = \text{hardlim}(w_i^T p + b_i) \right)$

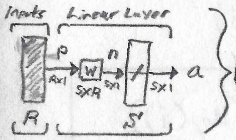
LEARNING RULES: $W^{new} = W^{old} + ep^T$, $b^{new} = b^{old} + e$ (where $e = t - a$)

Decision Boundary: $w_i^T p + b_i = \varnothing$

* Always orthogonal to weight vector
* Single-layer perceptron can only classify linearly seperable vectors

Supervised Hebbian Learning:

$$w_{ij}^{new} = w_{ij}^{old} + t_{qi} p_{qj}$$



(Linear Associator)

(Hebb Rule)

$a = \text{purelin}(Wp)$

$$W = t_1 p_1^T + t_2 p_2^T + \ldots + t_Q p_Q^T$$

$$\Rightarrow W = [t_1\, t_2 \ldots t_Q] \begin{bmatrix} p_1^T \\ p_2^T \\ \vdots \\ p_Q^T \end{bmatrix} \equiv TP^T$$

Pseudoinverse Rule: $W = TP^+$

* If R (# of rows of P) $> Q$ (# of columns of P) & all columns of P are independent

$$\Rightarrow P^+ = (P^T P)^{-1} P^T$$

Hebbian Learning Variations:

* Filtered Learning

$$\Rightarrow W^{new} = (1+\gamma) W^{old} + \alpha t_q p_q^T$$

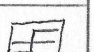
* Delta Rule

$$\Rightarrow W^{new} = W^{old} + \alpha (t_q - a_q) p_q^T$$

* Unsupervised Hebb

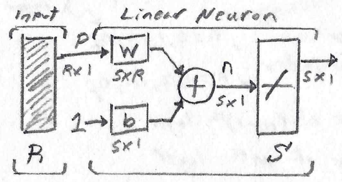
$$\Rightarrow W^{new} = W^{old} + \alpha a_q p_q^T$$

Pattern Classification: (bias = −1)



* use Symmetrical Hard Limit $\begin{cases} a = -1, & n < 0 \\ a = +1, & n \geq 0 \end{cases}$
* Not linearly seperable if classes on 2D plane (hyperplane) cannot be seperated by linear decision boundary

ADALINE:



$a = \text{purelin}(Wp+b)$

Mean Square Error (MSE):

If no bias $\Rightarrow w^T p + b \Rightarrow w^T p$

$$F(x) = E[e^2] = E[(t-a)^2]$$

$$\Rightarrow F(x) = E[(t - x^T z)^2] \text{ where } \begin{cases} x = \begin{bmatrix} w_1 \\ b \end{bmatrix} \\ z = \begin{bmatrix} p \\ 1 \end{bmatrix} \end{cases}$$

$$F(x) = c - 2x^T h + x^T R x$$

$(c = E[t^2],\ h = E[tz],\ R = E[zz^T])$

* If a unique minimum exists, it is $x^* = R^{-1} h$

Hessian $= 2R$

LMS Algorithm:

$$W(k+1) = W(k) + 2\alpha e(k) p^T(k)$$

$$b(k+1) = b(k) + 2\alpha e(k)$$

* Convergence Point

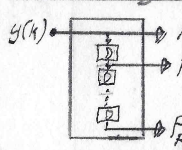
$$\Rightarrow x^* = R^{-1} h$$

* Stable Learning Rate

$$\Rightarrow 0 < \alpha < 1/\lambda_{max}$$

(where λ_{max} is maximum eigenvalue of R (find det))

Tapped Delay Line:



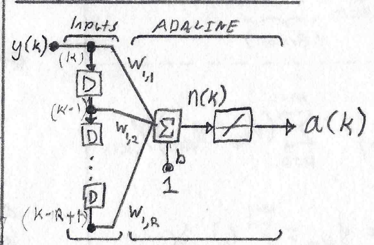
$p_1(k) = y(k)$

$p_2(k) = y(k-1)$

$\vdots$

$p_R(k) = y(k-R+1)$

Adaptive Filter ADALINE:

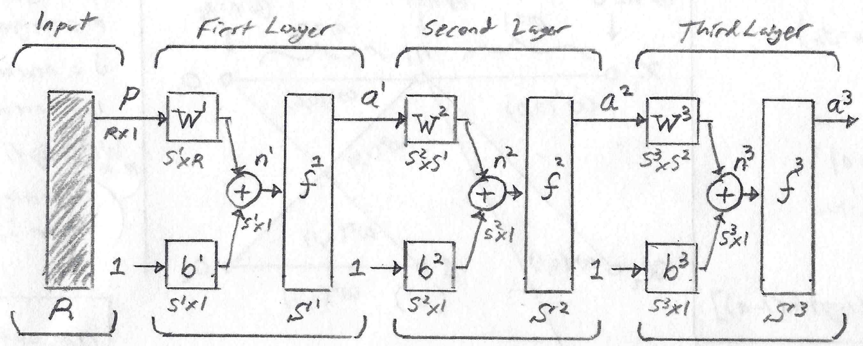


$a(k) = \text{purelin}(Wp+b)$

$$a(k) = \sum_{i=1}^{R} w_{1,i}\, y(k-i+1) + b$$

Multilayer Network:

(if classification problem, P_1, P_2 are coordinate inputs, 1st layer represents linear decision boundaries, 2nd layer "groups" them into regions, 3rd classifies)



$a^1 = f^1(W^1 p + b^1)$ $\quad a^2 = f^2(W^2 p + b^2)$ $\quad a^3 = f^3(W^3 p + b^3)$

$$\Rightarrow a^3 = f^3(W^3 f^2(W^2 f^1(W^1 p + b^1) + b^2) + b^3)$$

Backpropogation:

* Performance Index $\Rightarrow F(x) = E[e^T e] = E[(t-a)^T (t-a)]$
* Approximate Performance Index

$$\Rightarrow \hat{F}(x) = e^T(k) e(k) = (t(k) - a(k))^T (t(k) - a(k))$$

* Sensitivity $\Rightarrow$

$$S^m \equiv \frac{\partial \hat{F}}{\partial n^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n_1^m} \\ \frac{\partial \hat{F}}{\partial n_2^m} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n_{S^m}^m} \end{bmatrix}$$

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = \frac{\partial \hat{F}}{\partial n_i^m} \frac{\partial n_i^m}{\partial w_{i,j}^m} = s_i^m a_j^{m-1}$$

$\dot{f} \rightarrow$ derivative

Fwd Propagation:

$a^0 = p$ ← first layer neurons

$a^{m+1} = f^{m+1}(W^{m+1} a^m + b^{m+1})$, for $m = 0, 1, \ldots, M-1$

$a = a^M$ ← last layer outputs

Bwd Propagation:

$$S^M = -2\dot{F}^M(n^M)(t-a)$$

$$S^m = \dot{F}^m(n^m)(W^{m+1})^T S^{m+1}, \text{ for } m = M-1, \ldots, 2, 1$$

$$\dot{F}^m(n^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & \varnothing & \ldots & \varnothing \\ \varnothing & \dot{f}^m(n_2^m) & & \varnothing \\ \vdots & \vdots & & \vdots \\ \varnothing & \varnothing & & \dot{f}^m(n_{S^m}^m) \end{bmatrix}$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

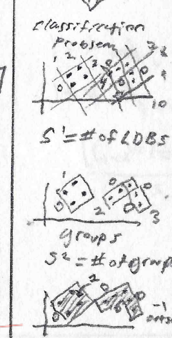
Weight Update (Approximate Steepest Descent):

$$W^m(k+1) = W^m(k) - \alpha S^m (a^{m-1})^T$$

$$b^m(k+1) = b^m(k) - \alpha S^m$$

For bypass connection: $\frac{\partial \hat{F}}{\partial w_{i,j}^{A,B}}$; $r(x) = \frac{\partial}{\partial y(x)} y(x)$; $r(k+1) = \ldots = \dot{f}(y(k)) r(k)$; A = centerlayer, B = input layer; $s^A(a^B)$; initialized at $k=0$, $r(0) = \frac{\partial y(0)}{\partial y(0)}$ for $k = 1, \ldots, k-1$

Classification Problem



S^1 = # of LDBs

groups

S^2 = # of groups

Final decision regions

S^3 = output classification

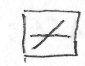
For Part 2, a neuron (group) is sum of LDBs that enclose a...

Hard Limit

$a = 0,\ n < 0$

$a = 1,\ n \geq 0$

Linear (purelin)

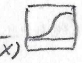


$a = n$

$f(x) = x,\ f'(x) = 1$

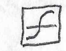
Log-Sig

$$f = \frac{1}{1+\exp(-x)}$$



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

Hyp. Tangent (tansig)



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

Bipolar Sigmoid ($\tanh(\frac{x}{2})$)

$$f = \frac{2}{1+e^{-x}} - 1 = \frac{1-e^{-x}}{1+e^{-x}}$$

$$f' = \frac{1}{2}((1+f)(1-f))$$

Netword tries to fit params. to target d

$$[w\ b] \begin{bmatrix} x \\ 1 \end{bmatrix} \equiv [x\ 1] \begin{bmatrix} w \\ b \end{bmatrix} = d$$

$A_{(2\times 2)} = \begin{vmatrix} a & b \\ c & d \end{vmatrix}$, find det

$\Rightarrow \det(A - \lambda I)$ — matrix, eigen values, identity mat

$\Rightarrow$ set $\det(\cdot) = \varnothing$ & solve

$\det(A - \lambda I) = \varnothing$

$(a-\lambda)(d-\lambda) - bc = \varnothing$

$\det \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$

Derivative Rules:

$$\frac{d}{dx} f(g(x)) = f'(g(x)) g'(x)$$

$$\frac{d}{dx} f(x) g(x) = fg' + f'g$$

$$\frac{d}{dx} \frac{f(x)}{g(x)} = \frac{gf' - fg'}{g^2}$$

Exponent Rules:

$b^n \cdot b^m = b^{n+m}$

$(b^n)^m = b^{n \cdot m}$

$\ln(x^p) = p \ln(x)$

$E(W) = E(\omega_1, \omega_2)$

$$\nabla E(W) = \left\langle \frac{\partial E}{\partial \omega_1}, \frac{\partial E}{\partial \omega_2} \right\rangle$$

Cost Functions:

MSE: $E_m = \frac{1}{2}\sum_{k=1}^{M}(t_k - y_k)$

Quadratic:

$C(\omega,b) = \frac{1}{2n}\sum_x |\underset{\text{desired}}{y(x)} - \underset{\text{actual}}{o}|^2$

Cross-Entropy:

$C = -\frac{1}{n}\sum_x [y\ln(a) + (1-y)\ln(1-a)]$

GD:

$\omega_{k+1} = \omega_k + \lambda_k D_k$ (direction vector)

CG:

$D_1 = -G(w_1)$, for $k=1$

$D_k = -G(w_k) + b_k D_{k-1}$, for $k>1$

$b_k = \dfrac{G(w_k)^t[G(w_k) - G(w_{k-1})]}{|G(w_{k-1})|^2}$

$\lambda_k = \dfrac{|G(w_k)|^2}{D_k^t A D_k}$

Softmax Activation: (similar to probability)

$a_j = \dfrac{e^{z_j}}{\sum_k e^{z_k}}$ & $\sum_j a_j = 1$

Fast Computations: (Matlab)

$D_{ij} = \sum(x_i - y_j)^2$

$\Rightarrow$ D = bsxfun(@plus, dot(x,x,1)', dot(y,y,1)) - 2(x'y) (1st dimension)

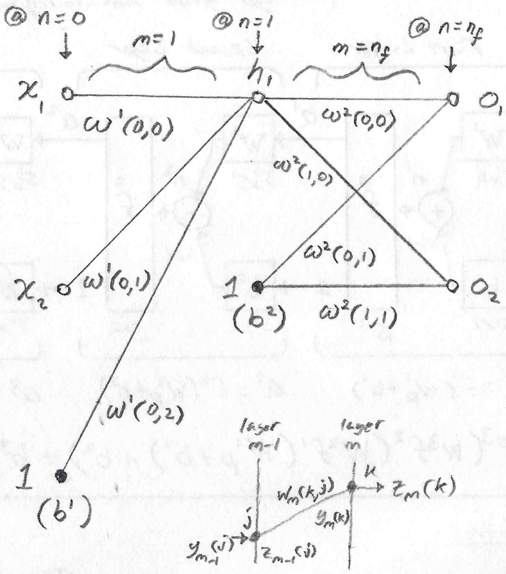
(Python)

```
def vector():
    x = numpy.arange(a, b, deltx)
    return (f(x)*delta), sum()

np.where(condition, r_true, r_false)
```

return when c is true

3-2-2 Backpropagation Network



n = layer index, $n = 0,1,\dots,n_f$ (output layer)

m = weight stage index, $m = 1,2,\dots,n_f$

j = neuron index of $(m-1)^{th}$ layer

i = neuron index of m^{th} layer

$W_m(i,j) \Rightarrow (i,j)^{th}$ element of W_m (connecting j^{th} node from $(m-1)^{th}$ layer to i^{th} node of layer m)

$y_m(k) = \sum_{j=0}^{N_{m+1}-1} W_m(k,j)\, z_{m-1}(j)$ (input to k^{th} neuron at the m^{th} layer)

$z_m(k) = f(y_m(k)) \leftarrow f(\omega x + b)$ (output from k^{th} neuron at the m^{th} layer)

$E = \frac{1}{2}\sum_{k=0}^{N_{nf}-1}(d(k) - o(k))^2$

LMS Algorithm as Spatial Filter Summary:

1) initialization $\Rightarrow$ set $\hat{w}_k(1) = \emptyset$, for $k = 1,2,\dots,p$

2) filtering

$\hat{y}(n) = \sum_{j=1}^{p}\hat{w}_j(n)x_j(n)$

$e(n) = d(n) - y(n) \leftarrow$ err signal $e(n)$

(Estimation)

$\Rightarrow \hat{w}_k(n+1) = \hat{w}_k(n) + \eta e(n) x_k(n)$ for $k=1,2,\dots,p$

Future Prediction from input vector $\hat{x}(n-1)$: recursive

$\hat{x}(n) = \hat{w}^T(n)\hat{x}(n-1)$, $\hat{\omega}(n+1) = \hat{w}(n) + \eta e(n)\hat{x}(n-1)$

$e(n) = x(n) - \hat{x}(n)$

Fast input: $X(n-1) = [X(n-1) \dots X(n-M)]^T$
Tap weights: $W = [W_1 W_2 \dots W_M]^T$
Prediction Order: M (data points)
Future Prediction: $\hat{x}(n|X_{n-1})$
$W(n+1) = W(n) + 2e(n)x_i(n), i = 1\dots M$

Weight Update $\Rightarrow \Delta W_m(i,j) = -\alpha \dfrac{\partial E}{\partial W_m(i,j)}$ (*Note that δ'_* is not needed if it does not connect to prior layer)

$\dfrac{\partial E}{\partial W_m(i,j)} = \dfrac{\partial E}{\partial y_m(i)}\dfrac{\partial y_m(i)}{\partial W_m(i,j)}$

$\dfrac{\partial y_m(i)}{\partial W_m(i,j)} = z_{m-1}(j)$

$*$ let $-\delta_m(i) = \dfrac{\partial E}{\partial y_m(i)}$

$\Rightarrow \Delta W_m(i,j) = \alpha\,\delta_m(i)\, z_{m-1}(j)$

$$\boxed{\begin{aligned}\delta_{n_f}(i) &= (d(i) - o(i))f'(y_{n_f}(i)) \\ \delta_m(i) &= f'(y_m(i))\sum_{k=0}^{N_{m+1}-1}\delta_{m+1}(k)\,\omega_{m+1}(k,i)\end{aligned}}$$

i) Output Layer ($m = n_f$)

$-\delta_m(i) = -\delta_{n_f}(i) = \dfrac{\partial E}{\partial y_{n_f}(i)} = \dfrac{\partial E}{\partial o(i)}\dfrac{\partial o(i)}{\partial y_{n_f}(i)} = -(d(i) - o(i))f'(y_{n_f}(i))$

ii) Hidden Layer ($m=1$)

$\delta_m(i) = \dfrac{\partial E}{\partial y_m(i)} = \dfrac{\partial E}{\partial z_m(i)}\dfrac{\partial z_m(i)}{\partial y_m(i)} = \dfrac{\partial E}{\partial z_m(i)} f'(y_m(i))$

$= \sum_{k=0}^{N_{m+1}-1}\dfrac{\partial E}{\partial y_{m+1}(k)}\dfrac{\partial y_{m+1}(k)}{\partial z_m(i)} = \sum_{k=0}^{N_{m+1}-1}\dfrac{\partial E}{\partial y_{m+1}(k)}\omega_{m+1}(k,i)$

BP (Non linear delta rule):

$g_m(i,j) = -\delta_m(i)\, z_{m-1}(j)$

$z_{m-1}(j)$ = output of node j in layer $(m-1)$

$\delta_{n_f}(i) = (d(i) - o(i))f'(y(i))$, for $m = n_f$ (f' in terms of f, y_0 after activation value)

$\delta_m(i) = f'(y_m(i))\sum_{k=0}^{N_{m+1}-1}\delta_{m+1}(k)\,W_{m+1}(k,i)$, for $m < n_f$

$\Rightarrow$ BP w/ Momentum: (@ iteration $k=1$, first epoch, new term is zero)

$\Delta W_m^{k+1}(i,j) = \alpha\,\delta_m(i)\, z_{m-1}(j) + \beta\,\Delta W_m^{k}(i,j)$ (momentum term)

Linear Delta Rule:

$\omega X_k = D_k$ (input) (outputs)

$E(\omega) = \sum_{k=0}^{m-1}|D_k - \omega X_k|^2$ (# of I/O pairs) (Error)

$\dfrac{\partial E}{\partial w(m,n)} = -\sum_{k=0}^{m-1}(d_{km} - y_{km})X_{kn}$

where $y_{km} = \sum_{j=0}^{N-1}\omega_{mj}X_{kj}$

Ex: $y(t) = w^t(t)X(t) + b(t)$

$y_o(t) = f(w(t)X(t) + b(t))$ (output after activation)

$E(\omega) = \frac{1}{2}(\underset{\text{desired}}{d(t)} - \underset{\text{actual}}{y_o(t)})^2$

$G(W) = \nabla_W E = -\underbrace{(d(t) - y_o(t))f'(y(t))}_{z(t)}X(t)$

$\Delta w(t) = -\alpha G(W) = \alpha z(t)X(t)$

$\Delta b(t) = -\alpha z(t)$

$\Rightarrow \Delta w(t) = \alpha(d(t) - y_o(t))X^t(t)$ (for $f(\cdot)$ = identity fn (LMS))

RK