**General Minimization Algorithm:**

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$ ***or*** $\Delta\mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$

**Steepest Descent Algorithm:**

$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$ **where,** $\mathbf{g}_k = \nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_k}$

**Stable Learning Rate:** $(\alpha_k = \alpha$, **constant**$)$ $\alpha < \frac{2}{\lambda_{max}}$

$\{\lambda_1\ \lambda_2, \ldots, \lambda_n\}$ Eigenvalues of Hessian matrix A

**Learning Rate to Minimize Along the Line:**

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \overset{is}{\Rightarrow} \alpha_k = -\frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A} \mathbf{p}_k}$ (For quadratic fn.)

**After Minimization Along the Line:**

$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k \Rightarrow \mathbf{g}_{k+1}^T \mathbf{p}_k = 0$

---

ADALINE: $\mathbf{a} = purelin(\mathbf{Wp} + \mathbf{b})$

**Mean Square Error:** *(for ADALINE it is a quadratic fn.)*

$F(\mathbf{x}) = E[e^2] = E[(t-a)^2] = E[(t - \mathbf{x}^T\mathbf{z})^2]$

$F(\mathbf{x}) = c - 2\mathbf{x}^T\mathbf{h} + \mathbf{x}^T\mathbf{R}\mathbf{x}$,

$c = E[t^2]$, $\mathbf{h} = E[t\mathbf{z}]$ *and* $\mathbf{R} = E[\mathbf{z}\mathbf{z}^T] \Rightarrow \mathbf{A} = 2\mathbf{R}$, $\mathbf{d} = -2\mathbf{h}$

Unique minimum, if it exists, is $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$,

**where** $\mathbf{x} = \begin{bmatrix} {}_1\mathbf{w} \\ b \end{bmatrix}$ **and** $\mathbf{z} = \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$

**LMS Algorithm:** $\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\, \mathbf{e}(k)\, \mathbf{p}^T(k)$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\, \mathbf{e}(k)$$

**Convergence Point:** $\mathbf{x}^* = \mathbf{R}^{-1}\mathbf{h}$

**Stable Learning Rate:** $0 < \alpha < 1/\boldsymbol{\lambda}_{max}$ where $\boldsymbol{\lambda}_{max}$ is the maximum eigenvalue of R

**Adaptive Filter ADALINE:**

$$a(k) = purelin(\mathbf{Wp}(k) + b) = \sum_{i=1}^{R} \mathbf{w}_{1,i}\, y(k-i+1) + b$$

---

**Backpropagation Algorithm:**

**Performance Index:**

Mean Square error: $F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$

**Approximate Performance Index:** (single sample)

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

**Sensitivity:** $\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial \mathbf{n}_1^m} & \frac{\partial \hat{F}}{\partial \mathbf{n}_2^m} & \cdots & \frac{\partial \hat{F}}{\partial \mathbf{n}_{s^m}^m} \end{bmatrix}^T$

**Forward Propagation:** $\mathbf{a}^0 = \mathbf{p}$,

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

$$\mathbf{a} = \mathbf{a}^M$$

**Backward Propagation:** $\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$,

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

$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \mathrm{diag}([\dot{f}^m(n_1^m)\ \ \dot{f}^m(n_2^m)\ \ \ldots\ \ \dot{f}^m(n_{s^m}^m)])$

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

**Weight Update (Approximate Steepest Descent):**

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

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

---

***Heuristic Variations of Backpropagation:**

**Batching:** The parameters are updated only after the entire training set has been presented. The gradients calculated for each training example are averaged together to produce a more accurate estimate of the gradient.(If the training set is complete, i.e., covers all possible input/output pairs, then the gradient estimate will be exact.)

**Backpropagation with Momentum (MOBP):**

$$\Delta\mathbf{W}^m(k) = \gamma\Delta\mathbf{W}^m(k-1) - (1-\gamma)\alpha\, \mathbf{s}^m(\mathbf{a}^{m-1})^T$$

$$\Delta\mathbf{b}^m(k) = \gamma\Delta\mathbf{b}^m(k-1) - (1-\gamma)\alpha\, \mathbf{s}^m$$

**Variable Learning Rate Backpropagation (VLBP)**

**1.** If the squared error (over the entire training set) increases by more than some set percentage $\boldsymbol{\zeta}$ (typically one to five percent) after a weight update, then the weight update is discarded, the learning rate is multiplied by some factor $\boldsymbol{\rho} < 1$, and the momentum coefficient $\gamma$ (if it is used) is set to zero.

**2.** If the squared error decreases after a weight update, then the weight update is accepted and the learning rate is multiplied by some factor $\boldsymbol{\eta} > 1$. If $\boldsymbol{\gamma}$ has been previously set to zero, it is reset to its original value.

**3.** If the squared error increases by less than $\boldsymbol{\zeta}$, then the weight update is accepted but the learning rate and the momentum coefficient are unchanged.

---

**Association:** $\mathbf{a} = hardlim(\mathrm{W}^0\mathrm{P}^0 + \mathbf{Wp} + b)$

An association is a link between the inputs and outputs of a network so that when a stimulus A is presented to the network, it will output a response B.

**Associative Learning Rules:**

**Unsupervised Hebb Rule:**

$$\mathbf{W}(q) = \mathbf{W}(q-1) + \alpha\, \mathbf{a}(q)\mathbf{p}^T(q)$$

**Hebb with Decay:**

$$\mathbf{W}(q) = (1-\gamma)\mathbf{W}(q-1) + \alpha\, \mathbf{a}(q)\mathbf{p}^T(q)$$

**Instar:** $\mathbf{a} = hardlim(\mathbf{Wp} + b)$, $\mathbf{a} = hardlim({}_1\mathbf{w}^T\mathbf{p} + b)$

The instar is activated for ${}_1\mathbf{w}^T\mathbf{p} = \|{}_1\mathbf{w}\|\|\mathbf{p}\|cos\theta \geq -b$

where $\theta$ is the angle between $\mathbf{p}$ and ${}_1\mathbf{w}$.

**Instar Rule:**

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\, a_i(q)(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$

${}_i\mathbf{w}(q) = (1-\alpha)\ {}_i\mathbf{w}(q-1) + \alpha\, \mathbf{p}(q)$, $if\ (a_i(q) = 1)$

**Kohonen Rule:**

${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\, (\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$ $for\ i \in X(q)$

**Outstar Rule:** $\mathbf{a} = satlins(\mathbf{W}p)$

$$\mathbf{w}_j(q) = \mathbf{w}_j(q-1) + \alpha\, (\mathbf{a}(q) - \mathbf{w}_j(q-1))\, \mathrm{p}_j(q)$$

---

**Competitive Layer:** $\mathbf{a} = compet(\mathbf{Wp}) = compet(\mathbf{n})$

**Competitive Learning with the Kohonen Rule:**

$${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha\, (\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1))$$

$$= (\mathbf{1} - \alpha)\, {}_{i^*}\mathbf{w}(q-1) + \alpha\, \mathbf{p}(q)$$

${}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1)$, $i \neq i^*$ where $i^*$ is the winning neuron.

**Self-Organizing with the Kohonen Rule:**

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha\, (\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$$

$= (\mathbf{1} - \alpha)\ {}_i\mathbf{w}(q-1) + \alpha\, \mathbf{p}(q)$, $i \in N_{i^*}(d)$

$$N_i(d) = \{j, d_{i,j} \leq d\}$$

**LVQ Network:** $(w_{k,i}^2 = 1) \Rightarrow$ subclass $i$ is a part of class $k$

$n_i^1 = -\|{}_i\mathbf{w}^1 - \mathbf{p}\|$, $\mathbf{a}^1 = compet(\mathbf{n}^1)$, $\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1$

**LVQ Network Learning with the Kohonen Rule:**

$${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) + \alpha\, (\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)),$$

$$if\ a_{k^*}^2 = t_{k^*} = 1$$

$${}_{i^*}\mathbf{w}^1(q) = {}_{i^*}\mathbf{w}^1(q-1) - \alpha\, (\mathbf{p}(q) - {}_{i^*}\mathbf{w}^1(q-1)),$$

$$if\ a_{k^*}^2 = 1 \neq t_{k^*} = 0$$

---

***hardlim***: $a = \begin{cases} 0 & n < 0 \\ 1 & n \geq 0 \end{cases}$, ***hardlims***: $a = \begin{cases} -1 & n < 0 \\ +1 & n \geq 0 \end{cases}$, ***purelin***: $a = n$, ***Logsig***: $a = \frac{1}{1+e^{-n}}$, ***tansig***: $a = \frac{e^n - e^{-n}}{e^n + e^{-n}}$, ***poslin***: $a = \begin{cases} 0 & n < 0 \\ n & n \geq 0 \end{cases}$,

***compet***: $a = \begin{cases} 1 & \text{neuron with max } n \\ 0 & \text{all other neurons} \end{cases}$, ***satlin***: $a = \begin{cases} 0 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$, ***satlins***: $a = \begin{cases} -1 & n < 0 \\ n & -1 \leq n \leq 1 \\ 1 & n > 1 \end{cases}$,

***Delay***: $a(t) = u(t-1)$, ***Integrator***: $a(t) = \int_0^t u(\tau)d\tau + a(0)$

****HINT:**

$$diag([1\ 2\ 3]) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$