Perceptrons

Lecture 5, COMP / ELEC / STAT 502
© E. Merényi

Inception: January 26, 2012 Last updated: January 31, 2017[1]

[1] Sources:
Colin Fyfe, Artificial Neural Networks and Information Theory. Course notes, Dept. of Computing and Information Systems, The University of Paisley, 2000
Frederick Ham and Ivica Kostanic: Principles of Neurocomputing for Science & Engineering. McGraw-Hill, 2001
Simon Haykin: Neural Networks. A Comprehensive Foundation. McMillan, New Jersey, 1999

Outline

Simple Supervised Learning

The outer product rule of the Associative Memory was not *really* learning. The MAM was given precalculated weights *at once*. The weights were designed directly from associated pattern pairs, in one step.

The iterative version (the error correcting scheme), however, *does learn* the weights over time based on error feedback, and can achieve optimal association (mapping). without computing inverse (pseudoinverse) of a matrix.

We will study this type of learning, when the mapping between input - output training pairs is *derived* by iterative adaptation of the weights.

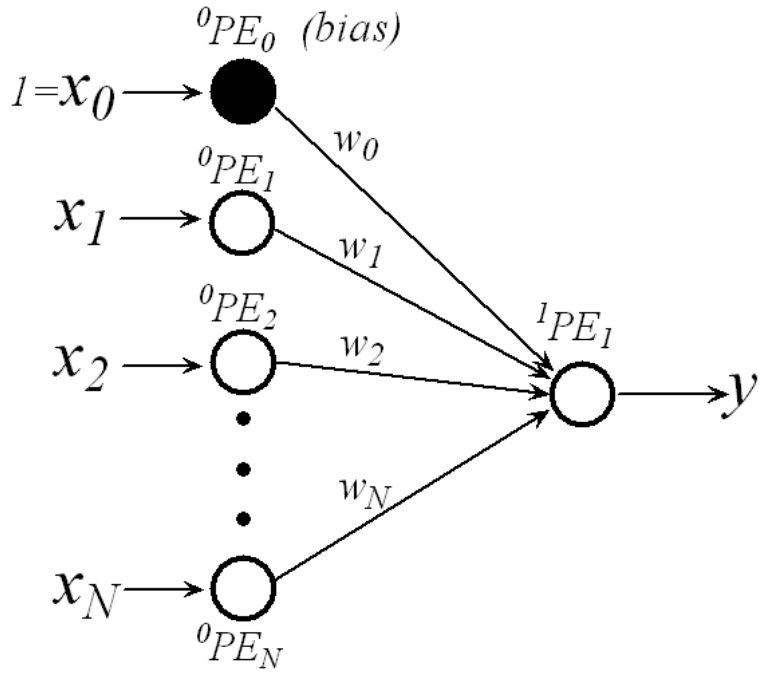
Multilayer Perceptrons (MLPs) are a principal family of ANNs with such (supervised) learning.

To study this family, we start with single and simple PEs, and then proceed to progressively more complex / sophisticated learning. This will build up to the derivation of the **Back Propagation** learning rule for multilayer, fully connected, feed forward ANNs.

Notations, and Simplification For Single Output PE

Simplify notation: omit layer index, and omit PE index in output layer

$$
\begin{aligned}
{}^1w_{ij} &\longrightarrow w_j \\
{}^1y_i &\longrightarrow y \\
D_i{}^k &\longrightarrow D^k \\
y_i{}^k &\longrightarrow y^k \\
{}^1NET_i &\longrightarrow NET \\
{}^1NET_i^k &\longrightarrow NET^k \\
{}^1NPE &\longrightarrow M(=1) \\
{}^0NPE &\longrightarrow N \qquad (5.1)
\end{aligned}
$$



$$
{}^1NET_i^k = \sum_{j=0}^{N} {}^1w_{ij}x_j^k \longrightarrow NET^k = \sum_{j=0}^{N} w_j x_j^k
$$

Simple Thresholded Binary Perceptron — Algorithm

1. Randomize weights, $w_j \in [-1, 1]$, set $t = 0$, learning rate $0 < \eta < 1$
2. $t = t + 1$. Select an input pattern $\mathbf{x}^k$ randomly from $(\mathbf{x}^1, \ldots \mathbf{x}^P)$
3. Calculate PE response

$$y^k = f(NET^k) = f(\sum_{j=0}^{N} w_j x_j^k)$$

$$= f(\sum_{j=1}^{N} w_j x_j^k + \underbrace{w_0 1}_{\text{bias}}) = \begin{cases} 1 & NET^k > 0 \\ -1 & NET^k < 0 \end{cases} \quad (5.2)$$

4. If $y^k = D^k$ go to 2)
5. Otherwise adjust the weights as

$$\Delta w_j = \eta x_j^k (D^k - y^k), \quad \eta > 0 \quad \text{learning rate} \quad (5.3)$$

6. Repeat 2) - 5) until $E^k = |D^k - y^k|$ is acceptable $\forall k$.

Simple Thresholded Binary Perceptron

Extension to multiple outputs is straightforward:

$$y_i^k = f(NET_i^k) = f(\sum_{j=0}^{N} w_{ij} x_j^k) \ \forall i = 1, \cdots, M$$

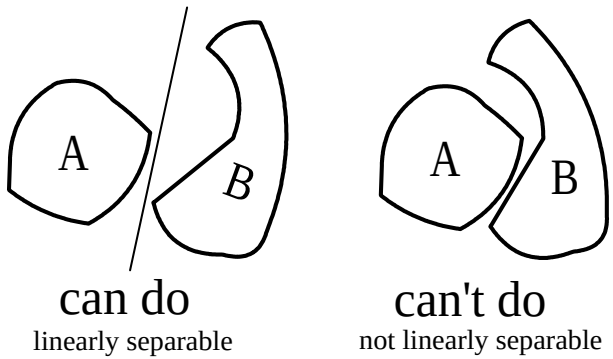
$$\Delta w_{ij} = \eta x_j^k (D_i^k - y_i^k), \quad \eta > 0 \tag{5.4}$$

Train until $E_i^k = |D_i^k - y_i^k| \ \forall k, i$ is acceptable.

Or, train until $\sum_{k=1}^{P} E^k$ is acceptable, where $E^k = \sum_{i=1}^{M} |D_i^k - y_i^k|$

Limitations Of the Simple Perceptron

- Binary
- Can only solve *linearly separable* problems, where decision boundaries can be single lines (examples: AND, OR, but not XOR) ... as you saw. It is a linear classifier.



can do
linearly separable

can't do
not linearly separable

Remember:
$y = w_1 x_1 + \cdots + w_n x_N + \underbrace{w_0 1}_{\text{bias}}$

N-dimensional line (hyperplane)
$\rightarrow$ divides the N-dim. space into 2 half spaces.

Importance Of the Simple Perceptron

- Simplicity
- Guaranteed convergence! (Frank Rosenblatt, 1959)

  If a given mapping exists, the weights of the perceptron will converge from **any** starting state. But it does not necessarily converge to the right solution!!!

  Proof of convergence can be found in Haykin, p 141. Not required, only to know that this proof exists.
- Historical (see examples in CF).
- Became a roadblock in ANN research in the late 60's ...

Some History

A book "Perceptrons" by Marvin Minsky & Seymour Papert (1969) strongly criticized ANNs based on the capabilities of the simple single-layer perceptron. They pointed out that many important functions could not be learned by **this** perceptron. XOR is one — important in communication, computer science, etc.

The popularity of ANNs dropped, for the next 10 – 15 years. Rosenblatt believed that *multi-layer* perceptrons could accomplish more sophisticated tasks, but at that time no learning rule was known for MLPs ...

Ironically, the Back Propagation learning rule was conceived in 1974, but because of the setback caused by Minsky and Papert, it did not catch on until the mid-80's.

Some History

Improvements soon after Minsky & Papert's book:

- binary $\rightarrow$ continuous
- transfer function $\rightarrow$ non-linear, continuous, asymptotic
- learning $\rightarrow$ by error descent, "Delta rule" or "Least Mean Square rule" or "Widrow-Hoff" rule

The Delta rule proved powerful, and it is at the heart of most supervised ANN paradigms ever since Widrow[2] and Hoff (1960). We will study learning with the Delta rule now.

However, convergence is harder to prove. Conditions that guaranteed convergence of the simple preceptron do not hold anymore.

[2] Bernard Widrow is active to date!

Training a Single Linear PE With Delta Rule

Consider again the simple perceptron architecture shown at (5.1). Let y^k be the response to input pattern $\mathbf{x}^k$.

$$y^k = \sum_{j=0}^{N} w_j x_j^k \quad \text{Linear transfer function!} \tag{5.5}$$

Formulate the objective (cost, error) function as

$$E = \sum_{k=1}^{P} E^k := \frac{1}{2}\sum_{k=1}^{P}(D^k - y^k)^2 = \frac{1}{2}\sum_{k=1}^{P}(D^k - \sum_{j=0}^{N} w_j x_j^k)^2 \tag{5.6}$$

Examine the error gradient, $\forall k$:

$$\frac{\partial E^k}{\partial w_j} = \underbrace{\frac{\partial E^k}{\partial y^k}}_{-(D^k - y^k)} \underbrace{\frac{\partial y^k}{\partial w_j}}_{x_j^k} = -(D^k - y^k)x_j^k \tag{5.7}$$

Training a Single Linear PE With Delta Rule

This suggests the following learning rule:

$$\Delta w_j = -\gamma[-(D^k - y^k)x_j^k] = \gamma \underbrace{(D^k - y^k)}_{\delta^k} x_j^k = \gamma \delta^k x_j^k \qquad \text{or} \qquad (5.8)$$

$$w_j(t+1) = w_j(t) + \gamma \delta^k x_j^k = w_j(t) + \gamma(D^k - y^k)x_j^k \qquad (5.9)$$

Process a different, randomly selected $\mathbf{x}^k$ at every time step t.

Show that this training converges / stabilizes in some interval, by showing that $E = E(\mathbf{w}(t))$ is a Lyapunov function in that interval. (Then an equilibrium point $\mathbf{w}^* : E(\mathbf{w}^*) = 0$ exists in the interval.) Equivalently, show that each term E^k is a Lyapunov function.

Convergence of Training With Delta Rule

E^k is a Lyapunov function:

1. E^k is scalar, positive definite since $E(\mathbf{w}(t)) \geq 0$ with $E = 0$ iff $\mathbf{w}(t) = \mathbf{w}^* : D^k - y^k = D^k - \mathbf{w}^{*T}\mathbf{x}^k = 0 \quad \forall k$.
2. E^k is continuously differentiable wrt $\mathbf{w}$ ($\frac{\partial E^k}{\partial w_j}$ is differentiable $\forall j$ and all orders)
3. $\frac{dE^k}{dt}$ is negative semidefinite:

$$\frac{dE^k}{dt} \approx \sum_{j=0}^{N}\left(\frac{\partial E^k}{\partial w_j}\Delta w_j\right) = -\sum_{j=0}^{N}[\underbrace{(D^k - y^k)}_{\delta^k}x_j^k]\gamma[\underbrace{(D^k - y^k)}_{\delta^k}x_j^k] \quad \leq 0 \tag{5.10}$$

Note 1: Remember though that γ has to be "appropriate" step size!

Note 2: In this simple case, E is purely quadratic (sum of squares) $\rightarrow E = 0$ only at $\mathbf{w} = \mathbf{w}^*$. In non-linear PEs the error function can be much more complicated, and the gradient descent training process can get stuck in local minima.

Training a Single Non-Linear PE With Delta Rule

Consider once more the same simple perceptron as described by the figure and eq. (5.1) and eq. (5.5) except with a transfer function $f(.)$ that is

1. non-linear
2. continuous, and continuously differentiable wrt $w_j \ \forall j$

$$y^k = f(\sum_{j=0}^{N} w_j x_j^k) = f(NET^k) \tag{5.11}$$

Formulate the objective (cost, error) function as before:

$$E = \sum_{k=1}^{P} E^k \stackrel{\Delta}{=} \frac{1}{2} \sum_{k=1}^{P} (D^k - y^k)^2 \quad = \frac{1}{2} \sum_{k=1}^{P} (D^k - f(\sum_{j=0}^{N} w_j x_j^k))^2 \tag{5.12}$$

$$= \frac{1}{2} \sum_{k=1}^{P} (D^k - f(NET^k))^2 \tag{5.13}$$

Training a Single Non-Linear PE With Delta Rule

We derive a learning rule with which E^k is a Lyapunov function for the equilibrium point $\mathbf{w}^*$: $E^k(\mathbf{w}^*) = 0$. Therefore the training will stabilize.

1. E^k is scalar, positive definite since $E^k(\mathbf{w}(t)) \geq 0$ with $E^k = 0$ iff $\mathbf{w}(t) = \mathbf{w}^*$.
2. E^k has continuous derivatives wrt $\mathbf{w}$, i.e., $\frac{\partial E^k}{\partial w_j}$ is differentiable $\forall j$ (We chose f(.) so.)
3. $\frac{dE^k}{dt}$ is negative semidefinite, as we show next. Therefore, any change of $\mathbf{w}$ in time according to the above learning rule, will cause E^k to decrease or stay unchanged.

Training a Single Non-Linear PE With Delta Rule

Let's look at the time derivative for one E^k term:

$$\frac{dE^k}{dt} \approx \sum_{j=0}^{N} \frac{\partial E^k}{\partial w_j} \Delta w_j = \sum_{j=0}^{N} \frac{\partial}{\partial w_j} \frac{1}{2} \left[D^k - y^k\right]^2 \Delta w_j$$

$$= \sum_{j=0}^{N} \frac{\partial E^k}{\partial y^k} \frac{\partial y^k}{\partial NET^k} \frac{\partial NET^k}{\partial w_j} \Delta w_j$$

$$= \sum_{j=0}^{N} (-1)(D^k - f(NET^k)) f'(NET^k) x_j^k \Delta w_j$$

$$= \sum_{j=0}^{N} (-1) \underbrace{(D^k - f(\sum_{l=0}^{N} w_l x_l^k)) f'(\sum_{l=0}^{N} w_l x_l^k)}_{\delta^k} x_j^k \Delta w_j = -\sum_{j=0}^{N} \delta^k x_j^k \Delta w_j \tag{5.14}$$

Training a Single Non-Linear PE With Delta Rule

Define the learning rule as

$$\Delta w_j = -\gamma \frac{\partial E^k}{\partial w_j} = \gamma \delta^k x_j^k \quad \gamma > 0 \tag{5.15}$$

Then

$$\frac{dE^k}{dt} \approx -\sum_{j=0}^{N} \delta^k x_j^k \Delta w_j = -\sum_{j=0}^{N} \delta^k x_j^k \gamma \delta^k x_j^k = -\sum_{j=0}^{N} \gamma (\delta^k x_j^k)^2 \leq 0 \tag{5.16}$$

Notice that

$$\delta^k = \underbrace{(D^k - f(\sum_{l=0}^{N} w_l x_l^k))}_{error} \underbrace{f'(\sum_{l=0}^{N} w_l x_l^k)}_{f'(NET^k)} \tag{5.17}$$

δ^k contains $f' \Longrightarrow \Delta w_j$ contains $f' \implies$ "Nice" transfer functions with easy to compute derivatives are desirable, as derivatives must be computed each learning step.

A Note On "Nice" Transfer Functions

The sigmoid and the hyperbolic tangent are functions whose derivatives are inexpensive to compute. An example using the sigmoid function:

$$f(x) = \frac{1}{1 + e^{-ax}} \qquad f'(x) = a * f(x)(1 - f(x)) \tag{5.18}$$

Applying this to eq. (5.17) gives

$$\delta^k = \underbrace{(D^k - f(\sum_{l=0}^{N} w_l x_l^k))}_{error} \underbrace{f'(\sum_{l=0}^{N} w_l x_l^k)}_{f'(NET^k)} = (D^k - f(NET^k)) * a * f(NET^k)(1 - f(NET^k)) \tag{5.19}$$

- $f(NET^k)$ computed once, used 3 x in computing δ^k
- No numerical derivation is needed

Similarly for hyperbolic tangent.

Desirable Properties Of Transfer Functions

In general

Linear — no good, allows unlimited grows of weights
Non-linear — must be the choice

Properties of good transfer functions:

- Non-linear, bounded
- Continuous, and continuously differentiable
- Monotonously increasing (with some exceptions s.a. Gaussians in RBF networks)
- Asymptotically tends to bounds
- Has "nice" derivatives

The hyperbolic tangent, in particular, is usually preferable to the sigmoid function because of its symmetric range (discussions to follow).

Training a Single Linear PE Again — Batch Training

Consider the simple linear perceptron with a single output PE as in (5.1). Similarly as we did at the OLAM, group the P N-dimensional training input patterns into a matrix $\mathbf{X}$ (adding also the constant bias input $x_0 = 1$):

$$\mathbf{X}^T = \begin{bmatrix} \mathbf{x}^{1\,T} \\ \mathbf{x}^{2\,T} \\ \vdots \\ \mathbf{x}^{P\,T} \end{bmatrix} = \begin{bmatrix} x_0^1 = 1 & x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_0^2 = 1 & x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^P = 1 & x_1^P & x_2^P & \cdots & x_N^P \end{bmatrix}_{P \times (N+1)} \tag{5.20}$$

Training a Single Linear PE Again — Batch Training

If y^k is the output in response to input pattern $\mathbf{x}^k$, that is, $y^k = {\mathbf{x}^k}^T \mathbf{w}$, where $\mathbf{w} = (w_0, w_1, w_2, \cdots, w_N)^T$ is the current weight vector, then we can collect the outputs in response to each of the input patterns for $k = 1, 2, \cdots, P$, into a (one-column) matrix $\mathbf{Y}$ (by running through all input patterns without a weight update):

$$\mathbf{Y} = \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^P \end{bmatrix} = \begin{bmatrix} x_0^1 = 1 & x_1^1 & x_2^1 & \cdots & x_N^1 \\ x_0^2 = 1 & x_1^2 & x_2^2 & \cdots & x_N^2 \\ \vdots & \vdots & \vdots & \vdots & \\ x_0^P = 1 & x_1^P & x_2^P & \cdots & x_N^P \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} = \begin{bmatrix} {\mathbf{x}^1}^T \mathbf{w} \\ {\mathbf{x}^2}^T \mathbf{w} \\ \vdots \\ {\mathbf{x}^P}^T \mathbf{w} \end{bmatrix} = \mathbf{X}^T \mathbf{w} \tag{5.21}$$

Training a Single Linear PE Again — Batch Training

If $\mathbf{D} = (D^1, D^2, \cdots, D^P)^T$ is the (one-column) matrix of the desired responses for $\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^P$, the (one-column) matrix of errors is

$$\mathbf{E} = \mathbf{D} - \mathbf{Y} = \begin{bmatrix} D^1 - y^1 \\ D^2 - y^2 \\ \vdots \\ D^P - y^P \end{bmatrix} = \begin{bmatrix} D^1 - {\mathbf{x}^1}^T\mathbf{w} \\ D^2 - {\mathbf{x}^2}^T\mathbf{w} \\ \vdots \\ D^P - {\mathbf{x}^P}^T\mathbf{w} \end{bmatrix} = \mathbf{D} - \mathbf{X}^T\mathbf{w} \quad (5.22)$$

Define cost function E^k for input pattern $\mathbf{x}^k$, and total cost E, resp., as

$$E^k(\mathbf{w}) = \frac{1}{2}(\underbrace{D^k - y^k}_{e^k})^2 = \frac{1}{2}(D^k - {\mathbf{x}^k}^T\mathbf{w})^2 \quad (5.23)$$

$$E = E(\mathbf{w}) = \sum_{k=1}^{P} E^k = \sum_{k=1}^{P} \frac{1}{2}(D^k - {\mathbf{x}^k}^T\mathbf{w})^2 = \frac{1}{2}||\mathbf{D} - \mathbf{X}^T\mathbf{w}||_2^2 \quad (5.24)$$

Training a Single Linear PE Again — Batch Training

In the above matrix notation this is equivalent to

$$E = E(\mathbf{w}) = \| \mathbf{E} \|_2^2 = \frac{1}{2} \| \mathbf{D} - \mathbf{X}^T\mathbf{w} \|_2^2$$
$$= \frac{1}{2}(\mathbf{D} - \mathbf{X}^T\mathbf{w})^T(\mathbf{D} - \mathbf{X}^T\mathbf{w}) \quad (5.25)$$

Once again, let us define the weight update rule to be negatively proportional to the gradient of $E(\mathbf{w})$ wrt $\mathbf{w}$ (minimize $E(\mathbf{w})$ by gradient descent):

$$\mathbf{w}(t+1) = \mathbf{w}(t) - \alpha \nabla_{\mathbf{w}} E(\mathbf{w})$$
$$= \mathbf{w}(t) + \alpha \mathbf{X} \underbrace{(\mathbf{D} - \mathbf{X}^T\mathbf{w})}_{\text{error for all patterns}} \quad (5.26)$$

(See computation of $\nabla_{\mathbf{w}} E(\mathbf{w})$ on the next page.)

Note To Eq. (5.26)

$\nabla_{\mathbf{w}} E(\mathbf{w})$ in eq. (5.26) can be derived as in (4.20) or using the following expansion:

$$E = E(\mathbf{w}) = \| \mathbf{E} \|_2^2 = \frac{1}{2} \| \mathbf{D} - \mathbf{X}^T\mathbf{w} \|_2^2 = \frac{1}{2}(\mathbf{D} - \mathbf{X}^T\mathbf{w})^T(\mathbf{D} - \mathbf{X}^T\mathbf{w})$$
$$= \frac{1}{2}(\mathbf{D}^T - \mathbf{w}^T\mathbf{X})(\mathbf{D} - \mathbf{X}^T\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} - \mathbf{w}^T\mathbf{X}\mathbf{D} + \frac{1}{2}\mathbf{D}^T\mathbf{D} \qquad (5.27)$$

The last equality holds because $(\mathbf{w}^T\mathbf{X}\mathbf{D})$ is a scalar, its transposed is itself: $(\mathbf{w}^T\mathbf{X}\mathbf{D})^T = \mathbf{D}^T(\mathbf{w}^T\mathbf{X})^T = \mathbf{D}^T\mathbf{X}^T\mathbf{w}$.

Thus

$$\nabla_{\mathbf{w}} E(\mathbf{w}) = \frac{1}{2}\left[(\mathbf{w}^T\mathbf{X})(\mathbf{X}^T\mathbf{w})\right]'_{\mathbf{w}} - \mathbf{X}\mathbf{D} + \mathbf{0} = \frac{1}{2}\left[(\mathbf{X})(\mathbf{X}^T\mathbf{w}) + (\mathbf{w}^T\mathbf{X})\mathbf{X}^T\right] - \mathbf{X}\mathbf{D}$$
$$= \textit{complete} \cdots \qquad (5.28)$$

Training a Single Linear PE Again — Batch Training

The weight update rule (eq. (5.26)) can be written as

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha \underbrace{\mathbf{X}(\mathbf{D} - \mathbf{X}^T\mathbf{w})}_{\mathbf{Xe}} = \mathbf{w}(t) + \alpha \sum_{k=1}^{P} \mathbf{x}^k \underbrace{(D^k - \mathbf{x}^{k\,T}\mathbf{w}(t))}_{\text{error } e^k\text{, scalar}} \tag{5.29}$$

$$\mathbf{Xe} = \begin{bmatrix} x_0^1 = 1 & \cdots & x_0^P = 1 \\ x_1^1 & \cdots & x_1^P \\ \vdots & & \\ x_N^1 & \cdots & x_N^P \end{bmatrix} \begin{bmatrix} D^1 - \mathbf{x}^{1\,T}\mathbf{w} \\ \vdots \\ D^P - \mathbf{x}^{P\,T}\mathbf{w} \end{bmatrix} = \begin{bmatrix} x_0^1 e^1 + \cdots + x_0^P e^P \\ x_1^1 e^1 + \cdots + x_1^P e^P \\ \vdots \\ x_N^1 e^1 + \cdots + x_N^P e^P \end{bmatrix}$$

$$= \mathbf{x}^1 e^1 + \mathbf{x}^1 e^1 + \cdots + \mathbf{x}^P e^P = \sum_{k=1}^{P} \mathbf{x}^k e^k$$

Training a Single Linear PE Again — Batch Training

Repeating (eq. (5.29)) in more general form

$$\mathbf{w}(t+1) = \mathbf{w}(t) + \alpha \sum_{k=1}^{K} \mathbf{x}^k \underbrace{(D^k - \mathbf{x}^{k\,T}\mathbf{w}(t))}_{\text{error for pattern } k\text{, scalar}} \tag{5.30}$$

$$= \mathbf{w}(t) + \alpha \sum_{k=1}^{K} \mathbf{x}^k e^k(t) \tag{5.31}$$

where the error $e^k(t)$ is a correction for all components of $\mathbf{w}(t)$, for training pattern $\mathbf{x}^k$.

Eq. (5.30) is the formula of *batch training* mode. An "epoch" is the number of (randomly selected) training patterns K for which errors (the $\mathbf{x}^k e^k(t)$ terms) are summed before one update of all weights. The summation above can be done for any *epoch size* K, $1 \leq K \leq P$. Very often an ideal epoch is much smaller than P, depending on data characteristics. $K = 1$ is *on-line training*.

Recommended Exercise

Derive the training rule for a single *non-linear PE* with the Delta rule, in matrix notation.

Then, do the same in batch formulation.

Crutch pages:
Crutch-Perceptron-eq.5.30.pdf
Crutch-AssociativeMemory-eq4.20.pdf
typical-variables-in-ANN.pdf

Excellent matrix algebra book:
Gene Golub and Charles Van Loan, Matrix Computations, John Hopkins University Press, (2nd ed.) 1989.