LINEAR NETWORKS FOR DATA-DRIVEN REGRESSION

Prof. FRANCESCO A. N. PALMIERI

UNIVERSITÀ DELLA CAMPANIA "LUIGI VANVITELLI"

[CORSO DI SIGNAL PROCESSING & DATA FUSION oct 2023]

LRØ

INTRODUCTION

A successful regressor or classifier learned from data depends on the type of parametric function $f(\underline{x}, \underline{\theta})$ chosen. Generally speaking the function has to be sufficiently "powerful" to represent the relations hidden in the data. On the other end, as pointed out before, an excessive parametrization may lead to useless overfitted solution. The issue is not so simple because we are not able to understand simply from the data the nature of the parametric function that may work, unless we try it! Generally, more free parameters the function has, the more powerful its representation capabilities are. Clearly the danger of overfitting is always there. Given a function with a small number of parameters may be considered "smoother", i.e. providing better generalization. However this is not necessarily the case because even if the number of parameters is large, there are many ways to "regularize" the solution. We will mention some in the following. The choice of the appropriate parametric function class is still now an "art", based on extensive experiments and trials.

LINEAR NETWORKS FOR REGRESSION

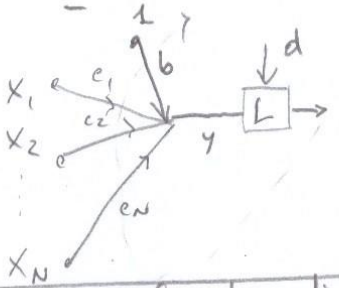
The simplest parametric function we could choose for our estimator, is a linear function.
To begin our discussion, let us consider a scalar regression problem where y and d are continuous values.

$$y = f(\underline{x}, \underline{\theta}) = \underline{c}^T \underline{x} + b$$

(The function is "affine" and not linear because of the bias b. (*) In the following we will ignore in the terminology this detail, referring generally to it as "linear")

In the model-based part of these lectures we have seen how this function emerges as the optimal solution when the likelihoods are gaussian, or more generally belong to the exponential family. Here the linear function is postulated. It has a parametric form with free parameters $\underline{\theta} = \begin{pmatrix} \underline{c} \\ b \end{pmatrix}$ that have to be learned from the training set $\mathcal{Z} = \{(\underline{x}[n], d[n]), n = 1, \dots, n_{\mathcal{Z}}\}$.
We have to solve the problem

$$\underline{\theta}^0 = \arg\min_{\underline{\theta}} \frac{1}{n_{\mathcal{Z}}} \sum_{n=1}^{n_{\mathcal{Z}}} L(\underline{c}^T \underline{x}[n] + b, d[n])$$

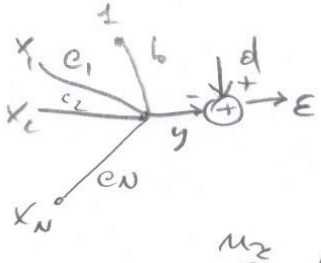


(*) A linear function has to satisfy the superposition principle
i.e. $\underline{x}_1 \to y_1$; $\underline{x}_2 \to y_2 \Rightarrow \alpha \underline{x}_1 + \beta \underline{x}_2 \to \alpha y_1 + \beta y_2 \quad \forall a, b$
This is not the case for affine functions because if $y = \underline{c}^T \underline{x} + b$
$\alpha \underline{x}_1 + \beta \underline{x}_2 \to \alpha(c^T x_1 + b) + \beta(c^T x_2 + b) = c^T \alpha x_1 + \alpha b + c^T \beta x_2 + \beta b$
$\neq c^T(\alpha x_1 + \beta x_2) + (\alpha + \beta) b = c^T(\alpha x_1 + \beta x_2) + b$

SOLUTION FOR QUADRATIC LOSS

When the loss function $L(\cdot)$ is quadratic in the difference $\varepsilon = d - y$ (most typical), we have

(LR2)



$$\theta^\circ = \begin{bmatrix} \underline{c}^\circ \\ b^\circ \end{bmatrix} = \underset{\begin{bmatrix} \underline{c} \\ b \end{bmatrix}}{\operatorname{argmin}} \sum_{n=1}^{n_2} \underbrace{\left(d[n] - \left(\underline{c}^T \underline{x}[n] + b \right) \right)}_{\varepsilon[n]}^2$$

The solution here can be found in closed-form as we have seen in designing FIR filters from data (Wiener). In that case did not include the bias as we considered zero-mean sequences. We reconsider the problem here for vector inputs, with a bias included and with no hypotheses on the means.

The cost function is

$$\mathcal{E}\left(\underline{c}, b\right) = \frac{1}{n_2} \sum_{n=1}^{n_2} \left(d[n] - \left(\underline{c}^T \underline{x}[n] + b \right) \right)^2$$

Setting the n_2 gradients to zero as usual, we have

$$\begin{cases} \nabla_{\underline{c}} \mathcal{E} = \frac{2}{n_2} \sum_{n=1}^{n_2} \left(d[n] - \left(\underline{c}^{\circ T} \underline{x}[n] + b^\circ \right) \right) \underline{x}[n] = \underline{0} \\ \frac{\partial \mathcal{E}}{\partial b} = \frac{2}{n_2} \sum_{n=1}^{n_2} \left(d[n] - \left(\underline{c}^{\circ T} \underline{x}[n] + b^\circ \right) \right) = 0 \end{cases}$$

$$\begin{cases} \sum_{n=1}^{n_2} d[n] \underline{x}[n] = \left(\sum_{n=1}^{n_2} \underline{x}[n] \underline{x}^T[n] \right) \underline{c}^\circ + b^\circ \sum_{n=1}^{n_2} \underline{x}[n] \\ \sum_{n=1}^{n_2} d[n] = \left(\sum_{n=1}^{n_2} \underline{x}^T[n] \right) \underline{c}^\circ + n_2 b^\circ \end{cases}$$

Solving for b°

$$b^\circ = \frac{1}{n_2} \left(\sum_{n=1}^{n_2} d[n] - \sum_{n=1}^{n_2} \underline{x}^T[n] \underline{c}^\circ \right)$$

μ_d $\quad$ $n_2 \underline{\mu}_x$

$$\sum_{n=1}^{n_2} d[n] \underline{x}[n] = \left(\sum_{n=1}^{n_2} \underline{x}[n] \underline{x}^T[n] \right) \underline{c}^\circ + \frac{1}{n_2} \sum_{n=1}^{n_2} d[n] \sum_{n=1}^{n_2} \underline{x}[n]$$

$$- \frac{1}{n_2} \left(\sum_{n=1}^{n_2} \underline{x}^T[n] \right) \underline{c} = \sum_{n=1}^{n_2} \underline{x}[n] - \underline{\mu}_x n_2$$

$\underline{\mu}_x^T$

Defining the empirical means (on the training set)

$$\mu_d \triangleq \frac{1}{n_2}\sum_{n=1}^{n_2} d[n] \qquad \underline{\mu}_x \triangleq \frac{1}{n_2}\sum_{n=1}^{n_2} \underline{x}[n]$$

We rewrite

$$\sum_{n=1}^{n_2} d[n]\,\underline{x}[n] = \left(\sum_{n=1}^{n_2} \underline{x}[n]\underline{x}^T[n]\right)\underline{c}^o + \mu_d \underline{\mu}_x n_2$$

$$-\underline{\mu}_x^T \underline{c}^o\, n_2 \underline{\mu}_x$$

$$\sum_{n=1}^{n_2} d[n]\,x[n] - \mu_d \underline{\mu}_x n_2 = \left(\sum_{n=1}^{n_2} \underline{x}[n]\underline{x}^T[n]\right)\underline{c}^o - \underline{\mu}_x \underline{\mu}_x^T \underline{c}^o n_2$$

$$\sum_{n=1}^{n_2} (d[n]-\mu_d)(\underline{x}[n]-\underline{\mu}_x) = \sum_{n=1}^{n_2} (\underline{x}[n]-\underline{\mu}_x)(\underline{x}[n]-\underline{\mu}_x)^T \underline{c}^o$$

$$\underline{c}^o = \left(\sum_{n=1}^{n_2} (\underline{x}[n]-\underline{\mu}_x)(\underline{x}[n]-\underline{\mu}_x)^T\right)^{-1} \sum_{n=1}^{n_2} (d[n]-\mu_d)(\underline{x}[n]-\underline{\mu}_x)$$

n_2 • sample covariance matrix $n_2 \underline{\Sigma}_x$ • sample cross covariance vector $\underline{V}_{dx}$

$$b^o = \mu_d - \underline{\mu}_x^T \underline{c}^o$$

the bias adds to the output the mean of $d[n]$ and subtracts from the output the effects of the mean of $x[n]$

The above solution generalizes the one we discussed for Wiener filters and shows the crucial role of the means and the bias to account for them. To account for the means we can use one of two strategies:

1) Remove the means at the beginning and use just a linear estimator $\underline{c}^T\underline{x}$. In this case the sample covariance is the sample autocorrelation matrix and the right-hand side is the cross correlation vector.

2) Include a bias in the estimator that automatically takes care of the means. (LR4)
(Preferred solution)

When the bias is included, a more compact formulation is obtained by considering an augmented input vector

$$\underline{x}_a[n] = \begin{bmatrix} 1 \\ \underline{x}[n] \end{bmatrix}$$

(We will use this "trick" in many neural network architectures)

and a linear mapping with an $(N+1)$ dimensional vector $\underline{w}$.

$$y = \underline{w}^T \underline{x}_a[n] = \underbrace{[b\, c_1 \cdots c_N]}_{\underline{w}^T} \begin{bmatrix} 1 \\ x_1[n] \\ x_2[n] \\ \vdots \\ x_N[n] \end{bmatrix}$$

b ← bias

The solution is immediately derived in compact form as:

$$\underline{w}^o = \left(\sum_{n=1}^{n_2} \underline{x}_a[n]\, \underline{x}_a^T[n] \right)^{-1} \left(\sum_{n=1}^{n_2} \underline{x}_a[n]\, d[n] \right) \quad (\star)$$

$\sum_{n=1}^{n_2} \underline{x}_a[n]\, \underline{x}_a^T[n] = n_2 \underline{R}_{x_a}$

$\sum_{n=1}^{n_2} \underline{x}_a[n]\, d[n] = n_2 \underline{\zeta}_{x_a d}$

$$\underline{w}^o = \underline{R}_{x_a}^{-1}\, \underline{\zeta}_{x_a d}$$

Note how the matrix $\frac{1}{n_2}\sum_{n=1}^{n_2} \underline{x}_a[n]\underline{x}_a^T[n] = \underline{\underline{R}}_{x_a}$ contains means and correlations (sample)

$$\begin{bmatrix} 1 \\ \underline{x}[n] \end{bmatrix}\begin{bmatrix} 1 & \underline{x}^T[n] \end{bmatrix} = \begin{bmatrix} 1 \\ x_1[n] \\ x_2[n] \\ \vdots \\ x_N[n] \end{bmatrix}\begin{bmatrix} 1 & x_1[n] & x_2[n] & \cdots & x_N[n] \end{bmatrix}$$

$$\begin{matrix} \underline{x}[n] & \underline{x}_a^T[n] \end{matrix}$$

$$= \begin{bmatrix} 1 & x_1[n] & x_2[n] & \cdots & x_N[n] \\ x_1[n] & x_1^2[n] & x_1[n]x_2[n] & \cdots & x_1[n]x_N[n] \\ x_2[n] & x_2[n]x_1[n] & x_2^2[n] & \cdots & x_2[n]x_N[n] \\ \vdots & & & & \\ x_N[n] & x_N[n]x_1[n] & x_N[n]x_2[n] & \cdots & x_N^2[n] \end{bmatrix}$$

$$\underline{\underline{R}}_{x_a} = \frac{1}{n_2}\sum_n \underline{x}_a[n]\underline{x}_a^T[n]$$

$$= \frac{1}{n_2}\begin{bmatrix} 1 & \sum_n x_1[n] & \sum_n x_2[n] & \cdots & \sum_n x_N[n] \\ \sum_n x_1[n] & \sum_n x_1^2[n] & \sum_n x_1[n]x_2[n] & \cdots & \sum_n x_1[n]x_N[n] \\ \sum_n x_2[n] & \sum_n x_2[n]x_1[n] & \sum_n x_2^2[n] & \cdots & \sum_n x_2[n]x_N[n] \\ \vdots & & & & \\ \sum_n x_N[n] & \sum_n x_N[n]x_1[n] & \sum_n x_N[n]x_2[n] & \cdots & \sum_n x_N^2[n] \end{bmatrix}$$

and the vector $\frac{1}{n_2}\sum_{n=1}^{n_2} \underline{x}_a[n]\,d[n] = \underline{r}_{x_a d}$ contains the mean of $d[n]$ and the cross-correlations between $d[n]$ and $\underline{x}[n]$.

$$\underline{r}_{x_a d} = \frac{1}{n_2}\sum_n \underline{x}_a[n]\,d[n]$$

$$= \frac{1}{n_2}\begin{bmatrix} \sum_n d[n] \\ \sum_n d[n]x_1[n] \\ \sum_n d[n]x_2[n] \\ \vdots \\ \sum_n d[n]x_N[n] \end{bmatrix}$$

(All the necessary information is included)

MINIMUM MEAN SQUARED ERROR (MMSE) ON THE TRAINING SET

The cost function $\mathcal{E}(\underline{w})$ in optimal conditions is expressed in one of the two forms:

$$\mathcal{E}(\underline{w}^o) = \mathcal{E}_d - \underline{z}_{xad}^T \underline{w}^o \quad \text{(I)}$$

or

$$\mathcal{E}(\underline{w}^o) = \mathcal{E}_d' - \underline{z}_{xad}^T \underline{\underline{R}}_{xa}^{-1} \underline{z}_{xad} \quad \text{(II)}$$

$$\mathcal{E}_{d1} = \frac{1}{N_z} \sum_{n=1}^{N_z} d^2[n]$$

Note that expression II produces the MMSE without computing $\underline{w}^o$ explicitely.

Proof By substitution

$$\mathcal{E}(\underline{w}^o) = \frac{1}{N_z} \sum_{n=1}^{N_z} \left(d[n] - \underline{w}^{oT} \underline{x}_a[n] \right)^2 = \frac{1}{N_z} \sum_{n=1}^{N_z} d^2[n] + \underline{w}^{oT} \left(\frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n] \underline{x}_a^T[n] \right) \underline{w}^o - 2 \frac{1}{N_z} \sum_{n=1}^{N_z} d[n] \underline{x}_a^T[n] \underline{w}^o$$

$$= \frac{1}{N_z} \sum_{n=1}^{N_z} d^2[n] + \frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a^T[n] d[n] \left(\frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n] \underline{x}_a^T[n] \right)^{-1} \left(\frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n] \underline{x}_a^T[n] \right) \underline{w}^o - 2 \frac{1}{N_z} \sum_{n=1}^{N_z} d[n] \underline{x}_a^T[n] \underline{w}^o$$

$$= \frac{1}{N_z} \sum_{n=1}^{N_z} d^2[n] \left(\frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a^T[n] d[n] \right) \underline{w}^o - 2 \frac{1}{N_z} \sum_{n=1}^{N_z} d[n] \underline{x}_a^T[n] \underline{w}^o$$

$$= \mathcal{E}_d - \underline{z}_{xad}^T \underline{w}^o \quad \boxed{\mathcal{E}(\underline{w}^o) = \mathcal{E}_d - \underline{z}_{xad}^T \underline{w}^o \quad \text{(I)}}$$

$$= \frac{1}{N_z} \sum_{n=1}^{N_z} d^2[n] - \frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n] d[n] \left(\frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n] \underline{x}_a^T[n] \right)^{-1} \frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n] d[n]$$

$$\mathcal{E}_d - \underline{z}_{xad}^T \underline{\underline{R}}_{xa}^{-1} \underline{z}_{xad} \quad \boxed{\mathcal{E}(\underline{w}^o) = \mathcal{E}_d - \underline{z}_{xad}^T \underline{\underline{R}}_{xa}^{-1} \underline{z}_{xad} \quad \text{(II)}}$$

...o

MEAN SQUARED ERROR ON THE VALIDATION SET

After computing the best solution on the training set, as in any machine learning application, we have to check how good the solution is on the validation set. Note that we cannot use expressions (I) and (II) that work only on the training set, but we have to plug in the solution in the general cost function expression

$$\mathcal{E}^V(\underline{w}^o) = \frac{1}{n_V} \sum_{n=1}^{n_V} \left(d[n] - \underline{w}^{oT} \underline{x}_a[n] \right)^2$$

that we can rewrite as

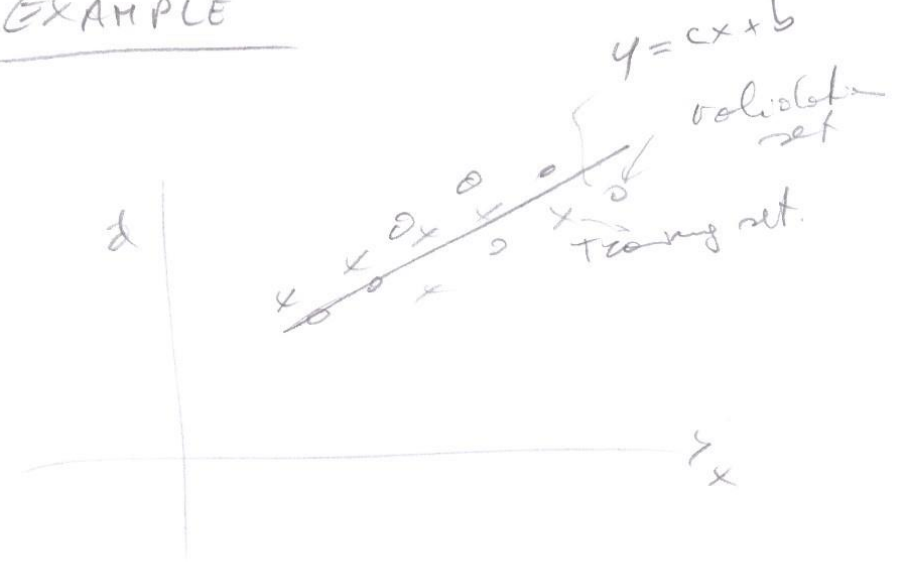
$$\mathcal{E}^V(\underline{w}^o) = \frac{1}{n_V} \sum_{n=1}^{n_V} d^2[n] + \underline{w}_o^{oT} \frac{1}{n_V} \sum_{n=1}^{n_V} \underline{x}_a[n] \underline{x}_a^T[n] \, \underline{w}_o + \frac{2}{n_V} \sum_{n=1}^{n_V} \underline{x}_a^T[n] \, d[n] \, \underline{w}^o$$

i.e. $\underline{w}^o$ is computed on the training set, but $\mathcal{E}_d^V \, \underline{\underline{R}}_{x_a}^V \, \underline{z}_{x_a d}^V$ must be computed for the validation set.

$$\begin{cases} \mathcal{E}^V(w^o) = \mathcal{E}_d + \underline{w}^{oT} \underline{\underline{R}}_{x_a}^V \underline{w}^o - 2 \underline{z}_{x_a d}^{VT} \underline{w}_o \\ \mathcal{E}_d^V = \frac{1}{n_V} \sum_{n=1}^{n_V} d^2[n] \; ; \quad \underline{\underline{R}}_{x_a}^V = \frac{1}{n_V} \sum_{n=1}^{n_V} \underline{x}_a[n] \underline{x}_a^T[n] \\ \underline{z}_{x_a d}^V = \frac{1}{n_V} \sum_{n=1}^{n_V} x_a[n] \, d[n] \end{cases}$$

In checking for generalization we compute both $\mathcal{E}$ and $\mathcal{E}^V$. It is usually better compare their square root (RMS value) $\sqrt{\mathcal{E}}$ and $\sqrt{\mathcal{E}^V}$ because RMS values are more easily interpretable as they are in the same orders of magnitude of d and y.

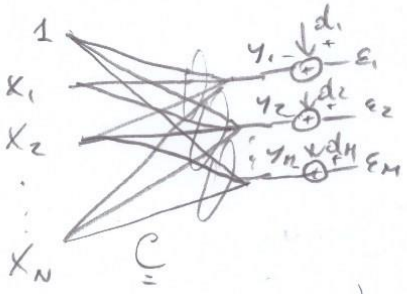
EXAMPLE



Compare $\mathcal{E}$ and $\mathcal{E}'$

THE MULTI-OUTPUT CASE

Consider now the straightforward extension to the case in which $\underline{y}$ and $\underline{d}$ are M-dimensional vectors



$$\underset{M\times 1}{\underline{y}} = \underset{M\times(N+1)}{\underline{\underline{W}}^T}\ \underset{(N+1)\times 1}{\underline{x}_a}$$

$$\underline{\underline{W}} = \underset{(N+1)\times M}{\left[\underline{w}_1\ \underline{w}_2\ \cdots\ \underline{w}_M\right]}$$

For economy of notation we consider the input to be the vector $\underline{x}_a$ augmented from $\underline{x}$, that includes the one in first position. Therefore the matrix $\underline{\underline{W}}$ already includes the biases and the situation is affine. (but linear in $\underline{x}_a$) The problem simply adding the contributions of the M outputs for the cost is

$$\underline{\underline{W}}^o = \underset{\underline{\underline{W}}}{\operatorname{argmin}} \frac{1}{N_z} \sum_{j=1}^{M} \sum_{n=1}^{N_z} \left(d_j[n] + \underline{w}_j^T \underline{x}_a[n]\right)^2$$

Which can be written in matrix form as

$$\underline{\underline{W}}^o = \underset{\underline{\underline{W}}}{\operatorname{argmin}} \frac{1}{N_z} \sum_{n=1}^{N_z} \left(\underline{d}[n] - \underline{\underline{W}}^T \underline{x}_a[n]\right)^T \left(\underline{d}[n] - \underline{\underline{W}}^T \underline{x}_a[n]\right)$$

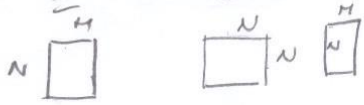
Since each "filter" $\underline{w}_j$ operates independently from the others, we can use the results of the previous section to write the solutions

$$(*) \qquad \underline{w}_j^o = \underline{\underline{R}}_{x_a x_a}^{-1}\, \underline{\mathcal{E}}_{x_a d_j} \quad \text{where} \quad \begin{cases} \underline{\underline{R}}_{x_a x_a} = \frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n]\, \underline{x}_a^T[n] \\ \underline{\mathcal{E}}_{x_a d_j} = \frac{1}{N_z} \sum_{n=1}^{N_z} \underline{x}_a[n]\, d_j[n] \end{cases}$$

In compact matrix form the solution can be written as

$$\underline{\underline{W}}^{o} = \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a d} \quad \text{with} \quad \underline{\underline{R}}_{x_a d} = \sum_{n=1}^{n_x} \underline{x}_a[n]\, \underline{d}^{T}[n]$$

cross correlation matrix



The error on the training set, as in previous section, is for output j

$$\mathcal{E}_j(\underline{w}_j^{o}) = \frac{1}{n_x} \sum_{n=1}^{n_x} d_j^2[n] - \underline{\tau}_{x_a d_j}^{T} \underline{\underline{R}}_{x_a}^{-1} \underline{\tau}_{x_a d_j} \qquad j = 1, \dots, M$$

$\mathcal{E}_j$

As pointed out above, there is a difference in the error computed on the training set and the one computed on the validation (or test) set. $\mathcal{V} = \{(\underline{d}[n], \underline{x}[n]),\ n = 1, \dots, n_v\}$ that is written as

$$\mathcal{E}_j^{\mathcal{V}}(\underline{w}_j^{o}) = \frac{1}{n_v} \sum_{n=1}^{n_v} \left(d_j[n] - \underline{w}_j^{oT} \underline{x}[n] \right)^2$$

$$= \frac{1}{n_v} \sum_{n=1}^{n_v} d_j^2[n] + \underline{w}_j^{oT} \left(\frac{1}{n_v} \sum_{n=1}^{n_v} \underline{x}_a[n]\, \underline{x}_a^{T}[n] \right) \underline{w}_j^{o} - 2 \frac{1}{n_v} \sum_{n=1}^{n_v} d_j[n]\, \underline{x}[n]$$

$\mathcal{E}_j^{\mathcal{V}}$; $\underline{\underline{R}}_{x_a}^{\mathcal{V}}$; $\underline{\tau}_{x_a d_j}^{\mathcal{V}}$

NOTATION:

When not specified $\underline{\underline{R}}_{x_a}$, $\underline{\tau}_{x_a d_j}$ are computed on the training set $\mathcal{X}$. $\underline{\underline{R}}_{x_a}^{\mathcal{V}}$, $\underline{\tau}_{x_a d_j}^{\mathcal{V}}$ are the ones computed on the validation set.

OTHER STATISTICS ON THE SOLUTION:

It is often useful to compute other statistics (other than the error) on the solution obtained from the learning set.

MEANS

For example the output mean on the learning set and on the validation set

$$\underline{\mu}_y = \underline{\underline{W}}^{oT} \underline{\mu}_{x_a} \qquad \underline{\mu}_y^v = \underline{\underline{W}}^{oT} \underline{\mu}_{x_a}^v$$

(Note that they are not the same.)

The mean error on the learning set is optimal conditionally to

$$\underline{\mu}_\varepsilon = \frac{1}{n_2} \sum_{n=1}^{n_2} \left(\underline{d}[n] - \underline{\underline{W}}^{oT} \underline{x}[n] \right) = \underline{\mu}_d - \underline{\underline{W}}^{oT} \underline{\mu}_{x_a}$$

But since $\underline{x}_a[n]$ is augmented with "1", we have

$$\underline{\mu}_{x_a} = \begin{bmatrix} 1 \\ \mu_{x_1} \\ \mu_{x_2} \\ \mu_{x_N} \end{bmatrix} = \begin{bmatrix} 1 \\ \underline{\mu}_x \end{bmatrix} \qquad \underline{\underline{W}}^o = [\underline{w}_1^o \, \underline{w}_2^o \cdots \underline{w}_M^o] = \begin{bmatrix} b_1^o & b_2^o & \cdots & b_M^o \\ \underline{c}_1^o & \underline{c}_2^o & \cdots & \underline{c}_M^o \end{bmatrix}$$

$$\underline{\mu}_\varepsilon^o = \underline{\mu}_d - \begin{bmatrix} b_1^o & \underline{c}_1^{oT} \\ b_2^o & \underline{c}_2^{oT} \\ b_M^o & \underline{c}_M^{oT} \end{bmatrix} \begin{bmatrix} 1 \\ \underline{\mu}_x \end{bmatrix} = \underline{\mu}_d - \begin{bmatrix} b_1^o + \underline{c}_1^{oT} \underline{\mu}_x \\ b_2^o + \underline{c}_2^{oT} \underline{\mu}_x \\ b_M^o + \underline{c}_M^{oT} \underline{\mu}_x \end{bmatrix}$$

From the previous section the optimal biases are

$$b_j^o = \mu_{d_j} - \underline{c}_j^{oT} \underline{\mu}_x \,,$$

Therefore the errors have zero mean on the learning set $\underline{\mu}_\varepsilon^o = \underline{0}$ (they are empirically unbiased)

These means are not necessarily null on the validation set

$$\underline{\mu}_\varepsilon^{oV} = \begin{bmatrix} \mu_{d_1}^v - b_1^o - \underline{c}_1^{oT} \underline{\mu}_x^v \\ \mu_{d_2}^v - b_2^o - \underline{c}_2^{oT} \underline{\mu}_x^v \\ \mu_{d_M}^v - b_M^o - \underline{c}_M^{oT} \underline{\mu}_x^v \end{bmatrix} \neq \underline{0}$$

LR12

The question of the means is in practical machine learning solutions is often overlooked. It is very important that the data that we present for training, for validation and then for testing, be properly "normalized". In other words the pairs $(\underline{x}^{(n)}, \underline{d}^{(n)})$ in $\mathcal{T}$, V and T have to be in the same ranges (means and excursions). Otherwise performance may be very poor. We will return on this issue later on, but for now we should observe that different means on the training set and on the validation set may produce unsatisfactory results. Therefore either:

(a) Remove the means from the data using no biases and perhaps add them again at the end (μ_d). Note that this can be done on means computed on the training set and/or on the validation set indicating confusion on the results.

(b) Make sure that the data in $\mathcal{T}$, V, and T are in the same ranges and are comparable in averages and values. In this way we can let the biases take care of the means on the training set, and expect good stable working on the other sets.

COVARIANCES

In studying the solution after optimization, it may be useful to look beyond the means and analyse variation parameters, such as variances and covariances. More specifically, the error variance, discussed before may gives us only a partial account of the solution: various output components may be better estimated than others and their values may be mutually correlated. Therefore we can consider the <u>error covariance matrix</u> on the <u>training set</u>

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \frac{1}{n_2} \sum_{n=1}^{n_2} \left(\underline{\varepsilon}^o[n] - \underline{\mu}_\varepsilon\right)\left(\underline{\varepsilon}^o[n] - \underline{\mu}_\varepsilon\right)^T$$

($\underline{\varepsilon}^o[n]$ is the error in optimal conditions)

where $\underline{\varepsilon}^o[n] = \underline{d}[n] - \underline{W}^{o^T} \underline{x}_d[n]$,

But we have already shown that on the training set $\underline{\mu}_\varepsilon = \underline{0}$, therefore

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \frac{1}{n_2} \sum_{n=1}^{n_2} \underline{\varepsilon}^o[n]\, \underline{\varepsilon}^{o^T}[n]$$

Note that on the diagonal $\underline{\underline{\Sigma}}_{\varepsilon^o}$ contains the variances of the errors on the various components, and the covariances on the other off-diagonal elements

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \begin{pmatrix} \sigma^2_{\varepsilon_1^o} & \sigma_{\varepsilon_1^o \varepsilon_2^o} & \cdots & \sigma_{\varepsilon_1 \varepsilon_M} \\ \sigma_{\varepsilon_2^o \varepsilon_1^o} & \sigma^2_{\varepsilon_2} & & \sigma_{\varepsilon_2^o \varepsilon_M^o} \\ \vdots & & & \\ \sigma_{\varepsilon_M^o \varepsilon_1^o} & & & \sigma^2_{\varepsilon_M^o} \end{pmatrix}$$

Inspection of the variances on the diagonal can tell us how accurate are the estimates on the various components (on the learning set).

$\underline{\underline{\Sigma}}_{\varepsilon^o}$ does not need to be computed from the data (training set) because we can write it in terms of the computed solution and statistics that we have already. More specifically

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \frac{1}{n_2} \sum_{n=1}^{n_2} \underline{\varepsilon}^o[n]\, \underline{\varepsilon}^{oT}[n] =$$

$$= \frac{1}{n_2} \sum_{n=1}^{n_2} \left(\underline{d}[n] - \underline{W}^{oT} \underline{x}_a[n]\right)\left(\underline{d}[n] - \underline{W}^{oT} \underline{x}_a[n]\right)^T$$

$$= \frac{1}{n_2} \sum_{n=1}^{n_2} \underline{d}[n]\underline{d}^T[n] + \underline{\underline{W}}^{oT} \frac{1}{n_2} \sum_{n=1}^{n_2} \underline{x}_a[n]\underline{x}_a^T[n]\, \underline{\underline{W}}^o$$

$$- \underline{\underline{W}}^{oT} \frac{1}{n_2} \sum_{n=1}^{n_2} \underline{x}_a[n]\underline{d}^T[n] - \frac{1}{n_2} \sum_{n=1}^{n_2} \underline{d}[n]\underline{x}_a^T[n]\, \underline{\underline{W}}^o$$

Since $\underline{\underline{W}}^o = \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a d}$

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \underline{\underline{R}}_d + \underline{\underline{R}}_{x_a d}^T \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a} \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a d} - \underline{\underline{R}}_{x_a d}^T \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a d} - \underline{\underline{R}}_{x_a d}^T \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a d}$$

$$\boxed{\underline{\underline{\Sigma}}_{\varepsilon^o} = \underline{\underline{R}}_d - \underline{\underline{R}}_{x_a d}^T \underline{\underline{R}}_{x_a}^{-1} \underline{\underline{R}}_{x_a d}}$$

or

$$\boxed{\underline{\underline{\Sigma}}_{\varepsilon^o} = \underline{\underline{R}}_d - \underline{\underline{W}}^{oT} \underline{\underline{R}}_{x_a d} = Rd - R_{x_a d}^T \underline{\underline{W}}^o}$$

(LR15)

On the validation set the situation is different because the error covariance for the computed solution is

$$\underline{\underline{\Sigma}}^{v}_{\varepsilon^o} \triangleq \frac{1}{n_v}\sum_{n=1}^{n_v}\left(\underline{\varepsilon}^o[n]-\underline{\mu}^{v}_{\varepsilon^o}\right)\left(\underline{\varepsilon}^o[n]-\underline{\mu}^{v}_{\varepsilon^o}\right)^T$$

$$= \frac{1}{n_v}\sum_{n=1}^{n_v}\underline{\varepsilon}^o[n]\,\underline{\varepsilon}^{oT}[n] - \underline{\mu}^{v}_{\varepsilon^o}\,\underline{\mu}^{vT}_{\varepsilon^o}$$

where $\underline{\mu}^{v}_{\varepsilon^o}$ is not necessarily null, as pointed out above. We have

$$\underline{\varepsilon}^o[n] = \underline{d}[n] - \underline{\underline{W}}^{oT}\underline{x}_a[n]$$

$$\underline{\mu}^{v}_{\varepsilon^o} = \underline{\mu}^{v}_{d} - \underline{\underline{W}}^{oT}\underline{\mu}^{v}_{x_a}$$

$$\underline{\mu}^{v}_{d} = \frac{1}{n_v}\sum_{n=1}^{n_v}\underline{d}[n]$$

$$\underline{\mu}^{v}_{x_a} = \frac{1}{n_v}\sum_{n=1}^{n_v}\underline{x}_a[n]$$

$\underline{\underline{\Sigma}}^{v}_{\varepsilon^o}$ must be computed on the data and its elements

$$\underline{\underline{\Sigma}}^{v}_{\varepsilon^o} = \begin{pmatrix} \sigma^{2\,v}_{\varepsilon_1^o} & \sigma^{v}_{\varepsilon_1\varepsilon_2} & \cdots & \sigma_{\varepsilon_1\varepsilon_M} \\ \sigma^{v}_{\varepsilon_2\varepsilon_1} & \sigma^{2\,v}_{\varepsilon_2^o} & & \\ & & & \\ & & & \sigma^{2\,v}_{\varepsilon_M^o} \end{pmatrix}$$

can tell how well the system works in generalization in comparison to $\underline{\underline{\Sigma}}_{\varepsilon^o}$ obtained on the training set.

SOLUTION WITH THE PSEUDOINVERSE

(LR16)

Least squares solutions to linear problems have a long history in the literature and they have been formulated in all kinds of ways. One of these is to see the linear regression problem as the approximate solution to an overdetermined linear system of equations. Consider for now the case with $M=1$ and with $\underline{x}$ already augmented

$$\underbrace{\begin{bmatrix} 1 & x_1[1] & x_2[1] & \cdots & x_N[1] \\ 1 & x_1[2] & x_2[2] & \cdots & x_N[2] \\ \vdots & & & & \\ 1 & x_1[M_2] & x_2[M_2] & \cdots & x_N[M_2] \end{bmatrix}}_{M_2 \times N} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_N \end{bmatrix} \approx \begin{bmatrix} d[1] \\ d[2] \\ \vdots \\ d[M_2] \end{bmatrix}$$

$$\underline{\underline{X}}_a \qquad \underline{w} \approx \underline{d}$$

Clearly the match between the two sides of the equation cannot be obtained exactly. We assume $M_2 \gg N$ and the system is overdetermined. Therefore we look for a solution that minimizes the sum of the squares of the differences between the two members of the equation

$$\underline{w}^\circ = \arg\min_{\underline{w}} \frac{1}{M_2} \| \underline{\underline{X}}_a \underline{w} - \underline{d} \|^2$$

$\| \underline{\underline{X}}_a \underline{w} - \underline{d} \|^2$ is exactly the cost function $E(w)$ used in the previous sections, written as a function of the large (tall) matrix $\underline{\underline{X}}_a$ and the tall vector $\underline{d}$.

$$\mathcal{E}(\underline{w}) = \frac{1}{N_2}\left(\underline{\underline{X}}_a \underline{w} - \underline{d}\right)^T \left(\underline{\underline{X}}_a \underline{w} - \underline{d}\right) = \frac{1}{N_2}\left(\underline{w}^T \underline{\underline{X}}_a^T \underline{\underline{X}}_a \underline{w} - \underline{d}^T \underline{\underline{X}}_a \underline{w} - \underline{w}^T \underline{\underline{X}}_a^T \underline{d} + \underline{d}^T \underline{d}\right) \quad (LR17)$$

$$= \frac{1}{N_2}\left(\underline{w}^T \underline{\underline{X}}_a^T \underline{\underline{X}}_a \underline{w} - 2\,\underline{d}^T \underline{\underline{X}}_a \underline{w} + \underline{d}^T \underline{d}\right)$$

Taking the gradient with respect to $\underline{w}$ (*), and setting it to zero, we have

$$2\,\underline{\underline{X}}_a^T \underline{\underline{X}}_a \underline{w} - 2\,\underline{\underline{X}}_a^T \underline{d} = \underline{0}$$

The solution is then

$$\boxed{\underline{w}^o = \left(\underline{\underline{X}}_a^T \underline{\underline{X}}_a\right)^{-1} \underline{\underline{X}}_a^T \underline{d}}$$

We can write the solution using the definition of pseudo-inverse (pinv(·) in Matlab)

$$\underline{\underline{X}}_a^{\#} = \left(\underline{\underline{X}}_a^T \underline{\underline{X}}_a\right)^{-1} \underline{\underline{X}}_a^T$$

$$\boxed{\underline{w}^o = \underline{\underline{X}}_a^{\#} \underline{d}}$$

To recognize that the solution is exactly the same derived in the previous sections, note that

$$\underline{\underline{X}}_x^T \underline{\underline{X}}_a = \begin{bmatrix} 1 & 1 & & 1 \\ x_1[1] & x_1[2] & \cdots & x_1[N_2] \\ \vdots & \vdots & & \vdots \\ x_N[1] & x_N[2] & & x_N[N_2] \end{bmatrix} \begin{bmatrix} 1 & x_1[1] & x_2[1] & \cdots & x_N[1] \\ 1 & x_1[2] & x_2[2] & \cdots & x_N[2] \\ \vdots & & & & \\ 1 & x_1[N_2] & x_2[N_2] & \cdots & x_N[N_2] \end{bmatrix} = N_2 \underline{\underline{R}}_x$$

(*) We have used the well-known results from differential matrix calculus

$$\nabla_{\underline{x}}\, \underline{b}^T \underline{x} = \underline{b} \;;\; \nabla_{\underline{x}}\, \underline{x}^T \underline{\underline{A}}\, \underline{x} = 2\,\underline{\underline{A}}\, \underline{x}$$

$$\underline{\underline{X}}_a^T \underline{d} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1[1] & x_1[2] & \dots & x_1[u_2] \\ \vdots & & & \\ x_N[1] & x_N[2] & \dots & x_N[u_2] \end{bmatrix} \begin{bmatrix} d[1] \\ d[2] \\ \vdots \\ d[u_2] \end{bmatrix} = u_2 \underline{r}_{x_a d}$$

and there is a perfect match with equation (4)
the total squared error in optimal conditions on the training set is also

$$\mathcal{E}(\underline{w}^o) = \frac{1}{u_2}\left(\underline{w}^{oT} \underline{\underline{X}}_a^T \underline{\underline{X}}_a \underline{w}^o - 2\underline{d}^T \underline{\underline{X}}_a^T \underline{w}^o + \underline{d}^T \underline{d}\right)$$

$$= \frac{1}{u_2}\left(\underline{d}^T \underline{\underline{X}}_a \left(\underline{\underline{X}}_a^T \underline{\underline{X}}_a\right)^{-1} \underline{\underline{X}}_a^T \underline{\underline{X}}_a \left(\underline{\underline{X}}_a^T \underline{\underline{X}}_a\right)^{-1} \underline{\underline{X}}_a^T \underline{d} - 2\underline{d}^T \underline{\underline{X}}_a \left(\underline{\underline{X}}_a^T \underline{\underline{X}}_a\right)^{-1} \underline{\underline{X}}_a^T \underline{d} + \underline{d}^T \underline{d}\right)$$

$$= \frac{1}{u_2} \underline{d}^T \underline{d} - \frac{1}{u_2} \underline{d}^T \underline{\underline{X}}_a \underbrace{\left(\underline{\underline{X}}_a^T \underline{\underline{X}}_a\right)^{-1} \underline{\underline{X}}_a^T \underline{d}}_{\underline{w}^o}$$

$$= \mathcal{E}_d - \frac{1}{u_2} \underline{d}^T \underline{\underline{X}}_a \underline{w}^o = \mathcal{E}_d - \underline{r}_{x_a d}^T \underline{w}^o = \mathcal{E}_d - \underline{r}_{x_a d}^T \underline{\underline{R}}_{x_a}^{-1} \underline{r}_{x_a d} \quad (*)$$

obviously the same obtained in the previous section.
Note again that the total error on the validation set is different and cannot be computed as in (*), but using the general expression

$$\mathcal{E}^v(\underline{w}^o) = \frac{1}{u_2}\left(\underline{w}^{oT} \underline{\underline{X}}_a^{vT} \underline{\underline{X}}_a^v \underline{w}^o - 2\underline{d}^{vT} \underline{\underline{X}}_a^{vT} \underline{w}^o + \underline{d}^{vT} \underline{d}^v\right)$$

where $\underline{\underline{X}}_a^v$ and $\underline{d}^v$ contain the validation set examples.

The extension to the multi-output case is straightforward. (LR 13.9)

$$\begin{bmatrix} 1 & x_1[1] & x_2[1] & \cdots & x_N[1] \\ 1 & x_1[2] & x_2[2] & \cdots & x_N[2] \\ \vdots & & & & \\ 1 & x_1[n_2] & x_2[n_2] & \cdots & x_N[n_2] \end{bmatrix} \begin{bmatrix} \underline{w}_1 & \underline{w}_2 & \cdots & \underline{w}_M \end{bmatrix} = \begin{bmatrix} d_1[1] & d_2[1] & \cdots & d_M[1] \\ d_1[2] & d_2[2] & \cdots & d_M[2] \\ \vdots & & & \\ d_1[n_2] & d_2[n_2] & \cdots & d_M[n_2] \end{bmatrix}$$

$$\underline{\underline{X}} \qquad \underline{\underline{W}} \qquad = \qquad \underline{\underline{D}}$$

Where we have arranged also the desired outputs in a unique matrix $\underline{\underline{D}}$

$$\underline{\underline{W}}^{o} = \underline{\underline{X}}^{\#}\underline{\underline{D}} = \left(\underline{\underline{X}}^{T}\underline{\underline{X}}\right)^{-1}\underline{\underline{X}}^{T}\underline{\underline{D}}$$

To recapture the correspondence to the solution found in the previous section as soon immediate if we observe that

$$\underline{\underline{R}}_{x_2} = \frac{1}{n_2}\underline{\underline{X}}^{T}\underline{\underline{X}} \quad \text{and} \quad \underline{\underline{R}}_{x_2 d} = \frac{1}{n_2}\underline{\underline{X}}^{T}\underline{\underline{D}}$$

The output error covariance, that contains the mean squared errors on the components in the diagonal, can be written for the training set in optimal condition directly as a function of $\underline{\underline{X}}$ and $\underline{\underline{D}}$.

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \frac{1}{n_2}\underline{\underline{D}}^{T}\underline{\underline{D}} - \frac{1}{n_2}\underline{\underline{D}}^{T}\underline{\underline{X}}\left(\underline{\underline{X}}^{T}\underline{\underline{X}}\right)^{-1}\underline{\underline{X}}^{T}\underline{\underline{D}}$$

or

$$\underline{\underline{\Sigma}}_{\varepsilon^o} = \frac{1}{n_2}\underline{\underline{D}}^{T}\underline{\underline{D}} - \frac{1}{n_2}\underline{\underline{D}}^{T}\underline{\underline{W}}^{o} = \frac{1}{n_2}\underline{\underline{D}}^{T}\underline{\underline{D}} - \frac{1}{n_2}\underline{\underline{W}}^{oT}\underline{\underline{D}}$$

On the validation set, the error means vector (not necessarily null) is:

LR20

$$\underline{\mu}_{\varepsilon^0}^{V} = \frac{1}{n_2}\left(\underline{\underline{X}}^{V}\underline{\underline{W}}^{0} - \underline{\underline{D}}^{V}\right)^{T}\underline{e}_{n_2} \qquad \text{where } e_{n_2} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}_{n_2}$$

$N \times n_2 \quad n_2$

and the error covariance

$$\underline{\underline{\Sigma}}_{\varepsilon^0}^{V} = \frac{1}{n_V}\left(\underline{\underline{X}}^{V}\underline{\underline{W}}^{0} - \underline{\underline{D}}^{V}\right)\left(\underline{\underline{X}}^{V}\underline{\underline{W}}^{0} - \underline{\underline{D}}^{V}\right)^{T} - \underline{\mu}_{\varepsilon^0}^{V}\underline{\mu}_{\varepsilon^0}^{V^T}$$

The compact matrix expressions in terms of $\underline{\underline{X}}$, $\underline{\underline{D}}$, $X^V D^V$, can be quite useful because they provide or closed-form immediate solution for programmers. This is ~~provided~~ provided that the training set and the validation set can be computed in large matrices.

LEARNING THE LINEAR REGRESSOR WITH A GRADIENT SEARCH

LR21

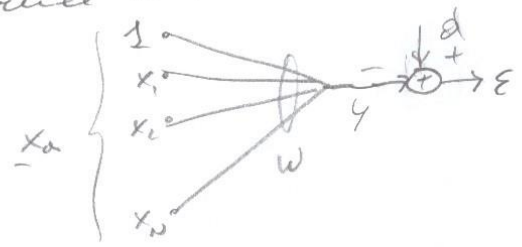
We have seen in the previous sections how the solution to the minimization of the mean of squares (MMSE) for the linear network can be found in closed-form. It is however very instructive to consider a gradient search also for this case for two main reasons:

(1) Data x's may not be available in a block (as in linear filtering)

(2) Not independently, the notion of the search already contains some elements that are important to understand gradient searches for more complex architectures.

(3) We may use loss functions different from the MMSE for which closed-form solutions do not exist.

With reference to the scalar case with $M = 1$



we seek to find

$$\underline{w}^o = \arg\min_{\underline{w}} \mathcal{E}(\underline{w}) = \arg\min_{\underline{w}} \frac{1}{n_2} \sum_{n=1}^{n_2} \psi\left(d[n] - \underline{w}^T \underline{x}_a[n]\right)$$

where $\psi(\cdot)$ is one of the loss functions defined in one of the previous sections. (The most typical being $\psi(\varepsilon) = \varepsilon^2$)

A gradient search follows the recursion

$$\underline{w}[k] = \underline{w}[k-1] - \mu(k) \nabla_{\underline{w}} \mathcal{E}(\underline{w}[k-1])$$

where

$$\mathcal{E}(\underline{w}[k-1]) = \frac{1}{n_2} \sum_{n=1}^{n_2} \psi(d[n] - \underline{w}^T[k-1] \underline{x}_a[n])$$

$$\nabla_{\underline{w}} \mathcal{E}(\underline{w}[k-1]) = \frac{1}{n_2} \sum_{n=1}^{n_2} \psi'(d[n] - \underline{w}^T[k-1] \underline{x}_a[n]) (-\underline{x}_a[n])$$

When $\psi(\xi) = \xi^2$, we have

$$\underline{w}[k] = \underline{w}[k-1] + \mu(k) \frac{2}{n_2} \sum_{n=1}^{n_2} (d[n] - \underline{w}^T[k-1] \underline{x}_a[n]) \underline{x}_a[n]$$

Since the factor $\frac{2}{n_2}$ can be absorbed into $\mu(k)$ the Batch-LMS algorithm is written as

$$\underline{w}[k] = \underline{w}[k-1] + \mu[k] \sum_{n=1}^{n_2} (d[n] - \underline{w}^T[k-1] \underline{x}_a[n]) \underline{x}_a[n]$$

(Note that $\underline{w}^T[k-1] \underline{x}_a[n]$ is the output $y[n]$ computed with the temporary value $\underline{w}[k-1]$)

More in general for other loss functions the Batch algorithm is written as

$$\underline{w}[k] = \underline{w}[k-1] + \mu[k] \sum_{n=1}^{n_2} \psi'(d[n] - \underline{w}^T[k-1] \underline{x}_a[n]) \underline{x}_a[n]$$

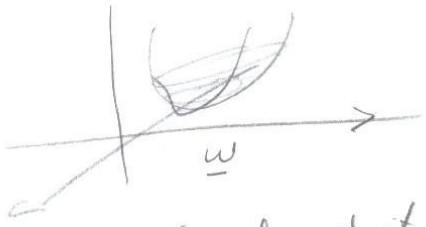
For example for $\psi(\xi) = |\xi|$ $\psi'(\xi) = \text{sgn}(\xi)$ and

$$\underline{w}[k] = \underline{w}[k-1] + \mu[k] \sum_{n=1}^{n_2} \text{sgn}[d[n] - \underline{w}^T[k-1] \underline{x}_a[n]] \underline{x}_a[n]$$

the algorithm is named Batch-sign algorithm. Any other cost function in the table X can be used. The question is:

Does the algorithm converge?

In the LMS case, we know that the cost function $\mathcal{E}(\underline{w})$ is quadratic and therefore convex in $\underline{w}$



The minimum is unique (we found it in closed-form) and the gradient search points towards the bottom of the bowl. There may be an issue at the bottom of the bowl because unless the updated value $\underline{w}[k]$ is exactly such that

$$\sum_{n=1}^{n_2} \left(d[n] - \underline{w}^T[k]\underline{x}_a[n]\right)\underline{x}_a[n] = \underline{0},$$

the search keeps wandering around the minimum. It is therefore advisable that there exists $\mu[k]$, or ADAM, to help conclude the search. This is usually a minor issue in linear problems. All the considerations of using mini-batch searches or on-line searches of section X clearly applies here too.

Noteworthy is the on-line (instantaneous) version of the algorithm

$$\underline{w}[k] = \underline{w}[k-1] + \mu[k]\underline{x}_a[k]\left(d[k] - \underline{w}^T[k-1]\underline{x}_a[k]\right)$$

which is the famous LMS algorithm. Note the use of the time index here because every time

an example is presented to the network, there is an update. This algorithm is usually very noisy and to get convergence we have to use a small stepsize parameter and present the training set many times.

A CONVERGENCE ANALYSIS

In linear systems there is a vast literature on studying the convergence of these algorithms, mostly for adaptive filters. Even if such analyses may not apply to more complex architectures, it is instructive for the student to be aware of the main ideas, also because some features may be transported into larger architectures. Consider the batch update with a constant stepsize parameter $\mu[k]=\mu$.

Convergence for the weight vector $\underline{w}[k]$

$$\underline{w}[k] = \underline{w}[k-1] + \mu \sum_{n=1}^{n_2} \left(d[n] - \underline{w}^T[k-1]\, \underline{x}_a[n]\right) \underline{x}_a[n]$$

that we can write as

$$\underline{w}[k] = \underline{w}[k-1] + \mu \sum_{n=1}^{n_2} \underline{x}_a[n] d[n] - \mu \left(\sum_{n=1}^{n_2} \underline{x}_a[n] \underline{x}_a^+[n]\right) \underline{w}[k-1]$$

$$\underline{w}[k] = \underline{w}[k-1] + \mu\, \underline{r}_{xad} - \mu\, \underline{\underline{R}}_{xa}\, \underline{w}[k-1]$$

We know that the optimal solution is $\underline{w}^o = \underline{\underline{R}}_{xa}^{-1} \underline{r}_{xad}$, therefore we consider the difference

$$\underline{w}[k] - \underline{w}^o = \underline{w}[k-1] - \underline{w}^o + \mu\, \underline{r}_{xad} - \mu\, \underline{\underline{R}}_{xa}\, \underline{w}[k-1]$$

Define $\underline{w}[k] - \underline{w}^o = \underline{v}[k]$ to be the weight error vector

$$\underline{v}[k] = \underline{v}[k-1] + \mu \underline{r}_{xd} - \mu \underline{\underline{R}}_x \left(\underline{v}[k-1] - \underline{w}^o\right)$$

$$\underline{v}[k] = \underline{v}[k-1] + \mu \underline{r}_{xd} - \mu \underline{\underline{R}}_x \underline{v}[k-1] - \mu \underline{\underline{R}}_x \underline{\underline{R}}_x^{-1} \underline{r}_{xd}$$

$$\underline{v}[k] = \left(\underline{\underline{I}} - \mu \underline{\underline{R}}_x\right) \underline{v}[k-1]$$

The sequence $\{\underline{v}[k]\}$ will converge to zero.
This is not easily seen in this recursion because all the components are coupled (R_x is not diagonal).
Therefore we use the spectral decomposition of R_x

$$\underline{\underline{R}}_x = \underline{\underline{Q}}\, \underline{\underline{\Lambda}}\, \underline{\underline{Q}}^T$$

$\underline{\underline{Q}} = [\underline{q}_1, \underline{q}_2 \cdots \underline{q}_N]$ eigenvectors

$\underline{\underline{\Lambda}} = \text{diag}(\lambda_1 \cdots \lambda_N)$ eigenvalues.

Substituting

$$\underline{v}[k] = \left(\underline{\underline{I}} - \mu \underline{\underline{Q}} \underline{\underline{\Lambda}} \underline{\underline{Q}}^T\right) \underline{v}[k-1]$$

and projecting the error vector in the eigenspace of $\underline{\underline{R}}_x$, $\underline{V}[k] = \underline{\underline{Q}}^T \underline{v}[k]$,

$$\underline{\underline{Q}}^T \underline{v}[k] = \underline{\underline{Q}}^T \left(\underline{\underline{I}} - \mu \underline{\underline{Q}} \underline{\underline{\Lambda}} \underline{\underline{Q}}^T\right) \underline{v}[k-1]$$

$$\underline{V}[k] = \left(\underline{\underline{Q}}^T - \mu \underline{\underline{Q}}^T \underline{\underline{Q}} \underline{\underline{\Lambda}} \underline{\underline{Q}}^T\right) \underline{v}[k-1]$$

$$\underline{V}[k] = \left(\underline{\underline{I}} - \mu \underline{\underline{\Lambda}}\right) \underline{V}[k-1]$$

Now all the components of $\underline{V}[k]$ are "decoupled" (N modes)

$$\begin{cases} V_1[k] = (1 - \mu\lambda_1) V_1[k-1] \\ V_2[k] = (1 - \mu\lambda_2) V_2[k-1] \\ \vdots \\ V_N[k] = (1 - \mu\lambda_N) V_N[k-1] \end{cases}$$

The condition for convergence of all the components is that

$$|1-\mu\lambda_i| < 1 \quad \forall i = 1,\dots,N$$

i.e.

$$-1 < 1-\mu\lambda_i < 1$$

$$-2 < -\mu\lambda_i < 0$$

$$0 < \mu\lambda_i < 2$$

$$0 < \mu < \frac{2}{\lambda_i}$$

A condition that works for all is

$$\boxed{0 < \mu < \frac{2}{\lambda_{max}}}$$

this means that μ must be positive and not too large.

Each component of $v[k]$ converges to the optimal value with a factor $(1-\mu\lambda_i)$. If we want to associate a convergence time to each mode, we write

$$(1-\mu\lambda_i) = e^{-\frac{1}{\tau_i}}$$

$$\tau_i = -\frac{1}{\ln(1-\mu\lambda_i)}$$

τ_i is the time that $v_i[k]$ takes to reach $\frac{1}{e}$ of $v_i[0]$.

If μ is small, $1-\mu\lambda_i$ is around 1 and negative,

using $\ln x \simeq x-1$ $\quad \ln(1-\mu\lambda_i) \simeq 1-\mu\lambda_i-1$

$$\tau_i \simeq \frac{1}{\mu\lambda_i}$$

A more detailed analysis of the evolution of $\underline{w}[k]$ can be obtained observing that (LR27)

$$\underline{v}[k] = Q\,\underline{v}[k]$$

or

$$\underline{w}[k] = \underline{w}^o + Q\,\underline{v}[k]$$

$$\underline{w}[k] = \underline{w}^o + \sum_{i=1}^{N} q_i\, v_i[k]$$

Each element of $\underline{w}[k]$ can be written as

$$w_i[k] = w_i^o + \sum_{j=1}^{N} q_{ij}\, v_j[0]\,(1-\mu\lambda_j)^k \qquad i = 1, \dots, N$$

the overall convergence of $\underline{v}[k]$ is the composition of N modes. The slowest mode is the one corresponding to the largest eigenvalue and the fastest by the smallest. The time to reach the solution $\underline{w}^o$ will be between the bounds

$$-\frac{1}{\ln(1-\mu\lambda_{max})} < \tau_a < -\frac{1}{\ln(1-\mu\lambda_{min})}$$

CONVERGENCE FOR THE SQUARED ERROR

In one of our previous chapters, we have shown that the cost function for the mean squared error, can be expressed in the canonical form

$$\mathcal{E}(\underline{w}[k]) = \mathcal{E}(w^o) + (\underline{w}[k]-\underline{w}^o)^T R_x (\underline{w}[k]-\underline{w}^o)$$

(easy to verify expanding the second term)

Using the spectral decomposition for R_{xx}, and $v[k] = w[k] - w^o$

LR28

$$\mathcal{E}(w[k]) = \mathcal{E}(w^o) + v[k]^T Q \Lambda Q^T v[k]$$

$$= \mathcal{E}(w^o) + V^T[k] \Lambda V[k]$$

$$= \mathcal{E}(w^o) + \sum_{i=1}^{N} V_i^2[k] \lambda_i$$

$$= \mathcal{E}(w^o) + \sum_{i=1}^{N} \lambda_i (1-\mu\lambda_i)^{2k} V_i^2[0]$$

Under the same conditions $|1-\mu\lambda_i| < 1$ we have that

$$\mathcal{E}[w[k]) \to \mathcal{E}(w^o)$$

Now since the error is the superposition of N modes, just as before, we can write that the time to reach the solution for each mode of $\mathcal{E}(w[k])$ is about

$$\tau_{i,\mathcal{E}} \approx -\frac{1}{2\ln(1-\mu\lambda_i)} \approx \frac{1}{2\mu\lambda_i}$$

Note that the time it takes for the error to reach convergence is about a half of the time it takes the coefficients to converge.

This is a behavior that we find also in more complex neural networks.

OTHER CONVERGENCE ANALYSES

As mentioned above, there are many contributions in the literature that analyze the convergence of LMS algorithms per the linear filters.
A more expanded version of these ideas goes beyond the scope of these notes that are

directed towards more complex neural network architectures. (LRZ9)

We just would like to mention a popular analysis of the instantaneous or LMS or stochastic gradient algorithm

$$w[k] = w[k-1] + \mu x_a[k] \left(d[k] - \underline{w}^T[k-1] \underline{x}_a[k] \right)$$

Where we consider $w[k]$ to be a stochastic process and study the evolution of its expected value.

$$E[w[k]] = E[w[k-1]] + \mu E[\underline{x}_a[k] d[k]] - \mu E[x_a[k] \underline{x}_a^T[k]] E[w[k-1]]$$

with $E[x_a[k] d[k]] = r_{x_a d}$ and $E[x_a[k] x_a^T[k]] = R_{x_a}$

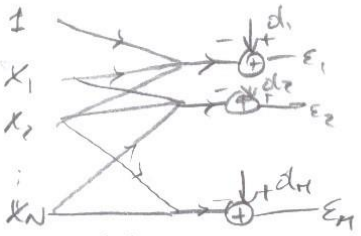
The same analysis reported above can be immediately applied to the recursion

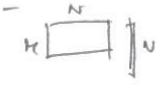
$$E[\underline{w}[k]] = E[\underline{w}[k-1]] - \mu R_{x_a d} - \mu R_{x_a} E[w[k-1]]$$

We refer the interested student to the vast literature on the topic [Haykin book] where simplifying assumptions are carefully detailed.

GRADIENT SEARCH FOR THE MULTI-OUPUT CASE (LR3D)

The extension of the gradient search for the linear case when there are multiple outputs is quite straightforward. Now $\underline{y} = \underline{W}^T \underline{x}_a$ with $\underline{W}$ $M \times N$ matrix





$\underline{W} = [\underline{w}_1, \dots \underline{w}_M]$

$\underline{W}$ $\underline{y}$ $\underline{\varepsilon}$

the cost function to minimize is

$$\mathcal{E}(\underline{W}) = \frac{1}{n_2} \sum_{n=1}^{n_2} \sum_{j=1}^{M} \psi_j \left(d_j[n] - \underline{w}_j^T \underline{x}_a[n] \right)$$

Where the losses for the M outputs are simply added. Now we can compute the gradients with respect to each one of the vectors $\underline{w}_j$ as

$$\frac{\partial}{\partial \underline{w}_j} = \frac{-1}{n_2} \sum_{n=1}^{n_2} \psi_j' \left(d_j[n] - \underline{w}_j^T \underline{x}_a[n] \right) \underline{x}_a[n] \qquad j = 1, \dots, M$$

and use the gradient update

$$\underline{w}_j[K] = \underline{w}_j[K-1] + \mu[K] \sum_{n=1}^{n_2} \psi_j' \left(d_j[n] - \underline{w}_j^T[K-1] \underline{x}_a[n] \right) \underline{x}_a[n]$$

$$j = 1, \dots M$$

However it is quite appealing to use directly a matrix formulation and consider a gradient flow, i.e. the derivative with respect to a matrix. Let us see how.

LR3.5

The cost function is written as

$$\mathcal{E}(\underline{\underline{W}}) = \frac{1}{n_2} \sum_{n=1}^{n_2} L\left(\underline{d}[n], \underline{\underline{W}}^T \underline{x}_a[n]\right)$$

And the updates are

$$\underline{\underline{W}}[k] = \underline{\underline{W}}[k-1] - \mu[k] \sum_{n=1}^{n_2} \nabla_{\underline{\underline{W}}} L\left(\underline{d}[n], \underline{\underline{W}}^T \underline{x}_a[n]\right)$$

where $\nabla_{\underline{\underline{W}}} L$ is the matrix with all its elements being the derivatives of L with respect to each element of $\underline{\underline{W}}$. For example if we consider $L\left(\underline{d}, \underline{\underline{W}}^T \underline{x}_a\right)$ to be the sum of the squares of the differences, i.e

$$L\left[\underline{d}, \underline{\underline{W}}^T \underline{x}_a\right) = \sum_{j=1}^{M} \left(d_j - \underline{w}_j^T \underline{x}_a\right)^2$$

we can write it in compact matrix form as

$$L\left(\underline{d}, \underline{\underline{W}}^T \underline{x}_a\right) = \left(\underline{d} - \underline{\underline{W}}^T \underline{x}_a\right)^T \left(\underline{d} - \underline{\underline{W}}^T \underline{x}_a\right)$$

which can be expanded to give

$$L\left(\underline{d}, \underline{\underline{W}}^T \underline{x}_a\right) = \underline{d}^T \underline{d} + \underline{x}_a^T \underline{\underline{W}} \underline{\underline{W}}^T \underline{x}_a - \underline{x}_a^T \underline{\underline{W}} \underline{d} - \underline{d}^T \underline{\underline{W}}^T \underline{x}_a$$

$$= \underline{d}^T \underline{d} + \underline{x}_a^T \underline{\underline{W}} \underline{\underline{W}}^T \underline{x}_a - 2\, \underline{d}^T \underline{\underline{W}}^T \underline{x}_a$$

Now we can use <u>matrix differential calculus</u> to compute the gradient with respect to $\underline{\underline{W}}$ (gradient flow) [Hagins and Nerdaker or Matrix cookbook]

We need the following results

$\nabla_{\underline{\underline{X}}} \, tr(AX) = A^T$ (1)

$\nabla_{\underline{\underline{X}}} \, tr(XAX^TB) = B^TXA^T + BXA$ (2)

$tr(\cdot)$ is the trace

Now, since the trace of a scalar is a scalar and $tr(AB) = tr(BA)$, we can write the loss as

$$L(\underline{d}, \underline{\underline{W}}^T\underline{x}_a) = \underline{d}^T\underline{d} + tr[\underline{x}_a^T\underline{\underline{W}}\underline{\underline{W}}^T\underline{x}_a] - 2\,tr[\underline{d}^T\underline{\underline{W}}^T\underline{x}_a]$$

$$= \underline{d}^T\underline{d} + tr[\underline{\underline{W}}\,\underline{\underline{W}}^T\underline{x}_a\underline{x}_a^T] - 2\,tr[\underline{x}_a\underline{d}^T\underline{\underline{W}}^T]$$

We can use (1) setting $X = W^T$ and $A = \underline{x}_a\underline{d}^T$, to get

$$\nabla_{\underline{\underline{W}}^T} tr[\underline{x}_a\underline{d}^T\underline{\underline{W}}^T] = d\,x_a^T \Rightarrow \nabla_{\underline{\underline{W}}} = \underline{x}_a\underline{d}^T$$

We can use (2) setting $\overset{X=W}{A=\underline{\underline{I}}}$, $B = \underline{x}_a\underline{x}_a^T$, to get

$$\nabla_{\underline{\underline{W}}} tr[\underline{\underline{W}}\underline{\underline{W}}^T\underline{x}_a\underline{x}_a^T] = x_a x_a^T W + x_a x_a^T W = 2\underline{x}_a\underline{x}_a^T\underline{\underline{W}}$$

Therefore

$$\nabla_{\underline{\underline{W}}} (\underline{d} - \underline{\underline{W}}^T\underline{x}_a)^T(\underline{d} - \underline{\underline{W}}^T\underline{x}_a) = 2\,\underline{x}_a\underline{x}_a^T\underline{\underline{W}} - 2\,\underline{x}_a\underline{d}^T$$

the gradient of $\mathcal{E}(\underline{\underline{W}})$ is clearly null when

$$\left(\sum_{u=1}^{u_x} \underline{x}_a[u]\,\underline{x}_a^T[u]\right)\underline{\underline{W}}^0 = \sum_{u=1}^{u_z} \underline{x}_a[u]\,\underline{d}^T[u]$$

$$\underline{\underline{R}}_{x_a}\underline{\underline{W}}^0 = \underline{\underline{R}}_{x_a d}$$

the we had already obtained. The update (batch) is also immediately the obvious matrix version of the one obtained for the individual $\underline{w}_j$

$$\underline{\underline{W}}[k] = \underline{\underline{W}}[k-1] + \mu[k] \sum_{n=1}^{N_2} \left(\underline{x}_a[n]\, \underline{d}^T[n] - \underline{x}_a[n]\underline{x}_a^T[n]\, \underline{\underline{W}}[k-1] \right)$$

which can be written as

$$\underline{\underline{W}}[k] = \underline{\underline{W}}[k-1] + \mu[k] \sum_{n=1}^{N_2} \underline{x}_a[n] \left(d[n] - \underline{W}^T[k-1]\, \underline{x}_a[n] \right)^T$$

$$= \underline{\underline{W}}[k-1] + \mu[k] \sum_{n=1}^{N_2} \underline{x}_a[n] \left(\underline{d}[n] - \underline{y}[n, k-1] \right)^T$$

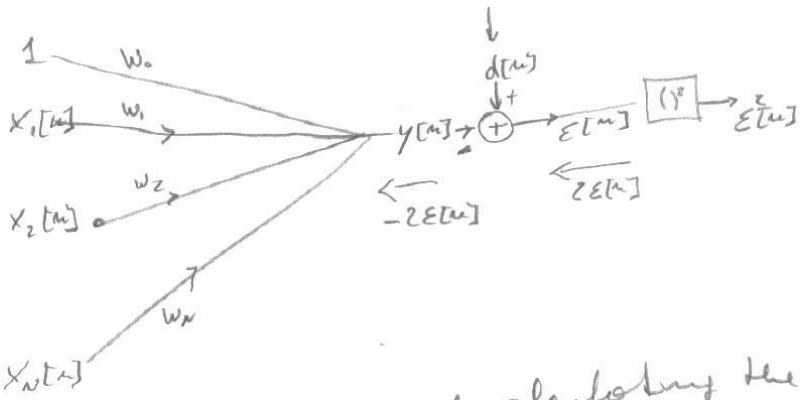
$\underline{y}[n, k-1]$: output computed with the old value $\underline{W}[k-1]$.

$\underline{\varepsilon}[n]$: error vector with the old value $\underline{W}[k-1]$ and the current example $(d[n], \underline{x}_a[n])$

To write the recursions in matrix form may simplify greatly the programming of the algorithm.

A FIRST LOOK AT THE GRADIENT ALGORITHM AS BACKPROPAGATION

(LR3.4)



Note that in the objective of calculating the derivative of $\varepsilon^2[n]$ with respect to w_i we can use the chain rule

$$\frac{\partial \varepsilon^2}{\partial w_i} = \frac{\partial \varepsilon^2}{\partial \varepsilon}\frac{\partial \varepsilon}{\partial w_i} = \frac{\partial \varepsilon^2}{\partial \varepsilon}\frac{\partial \varepsilon}{\partial y}\frac{\partial y}{\partial w_i}$$

$$2\varepsilon \quad -1 \quad x_i$$

(1) At each example backpropagate $\varepsilon^2[n]$ on $y[n]$

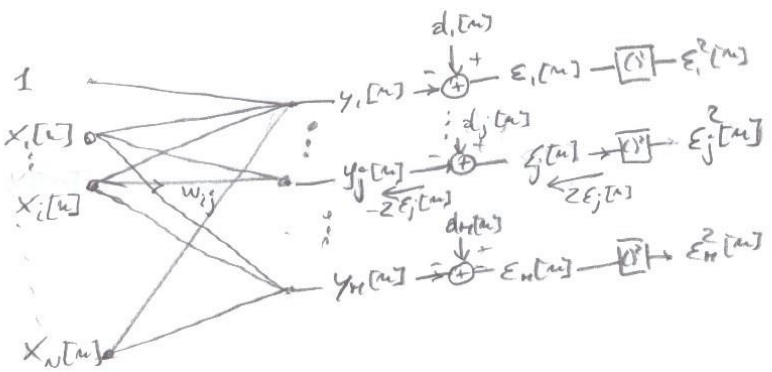
(2) Multiply by $x_i[n]$

(3) Store this product and use it to update w_i at the end of the epoque (or a selection of it for minibatch and instantaneous versions)

Backpropagation through the block $(\cdot)^2$ is simply the multiplication by 2. Back propagation through the minus node is simply multiplication by -1.

The same for multiple outputs

(LR35)



$$\frac{\partial}{\partial w_{ij}} \sum_{j=1}^{M} \varepsilon_j^2 = \frac{\partial \varepsilon_j^2}{\partial w_{ij}} = \frac{\partial \varepsilon_j^2}{\partial \varepsilon_j} \frac{\partial \varepsilon_j}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}}$$

$$2\varepsilon_j \quad -1 \quad x_i$$

Note that in this architecture the gradients do not interact. For each j and w_j the updates are independent. This will not be the case in multi-layer structures where the gradients in the previous layers will interact.

COMPRESSING THE INPUT SPACE

It is quite common in the applications of signal processing that the data vector $x[n]$ presented to the network is largely redundant. This may be the consequence that many of its components carry the same information on different scales and noises. In a linear system this information redundancy is captured by the covariances among the various components. Assuming that the means have been removed we can look at the structure of the autocorrelation matrix $\underline{\underline{R}}_x$, that is we have seen previously, plays an essential role in providing least squares solutions to regression problems.

Therefore we consider the spectral decomposition

$$\underline{\underline{R}}_x = Q \Lambda Q^T$$

where Λ is a diagonal matrix containing the eigenvalues

$$\Lambda = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_N)$$

and Q the unitary matrix containing on its columns the corresponding eigenvectors

$$\underline{\underline{Q}} = [\underline{q}_1, \underline{q}_2, \ldots, \underline{q}_N]$$

It often happens that some of the eigenvalues are very small, or even null. This means that the input space is degenerate and only a smaller number of components may be sufficient to represent the whole input dataset.

Assume that the eigenvalues are ranked in a descending order:

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \dots \geq \lambda_N$$

we could consider a division of the set into two parts

$$\underbrace{\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_\ell}_{} \geq \underbrace{\lambda_{\ell+1} \geq \dots \geq \lambda_N}_{}$$

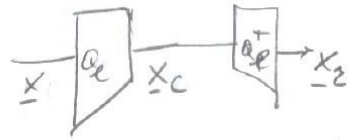
where the first ℓ eigenvalues are the most relevant and the others may be considered too small or null to be relevant.

The spectral factorization can be written in a partitioned form as

$$\underline{\underline{R}}_{\underline{X}\underline{X}} = \underline{\underline{Q}}\,\underline{\underline{\Lambda}}\,\underline{\underline{Q}}^T = \begin{bmatrix} \underline{\underline{Q}}_\ell & \underline{\underline{Q}}_{N-\ell} \end{bmatrix} \begin{pmatrix} \underline{\underline{\Lambda}}_\ell & 0 \\ 0 & \underline{\underline{\Lambda}}_{N-\ell} \end{pmatrix} \begin{bmatrix} \underline{\underline{Q}}_\ell^T \\ \underline{\underline{Q}}_{N-\ell}^T \end{bmatrix}$$

and consider the projection of the input vector onto the first ℓ components on the <u>compressed</u> vector (PRINCIPAL COMPONENTS)

$$\underline{x}_C = \underline{\underline{Q}}_\ell^T \underline{x}$$



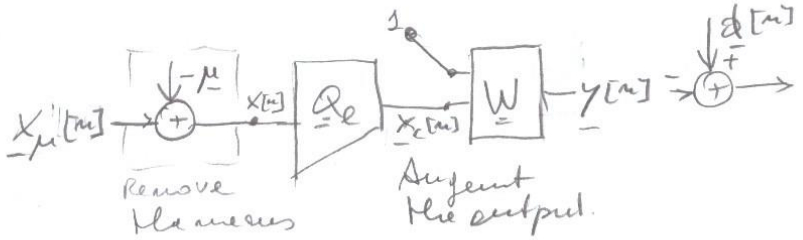
It is easy to demonstrate that the reconstruction of $\underline{x}$ by the inverse $\underline{\underline{Q}}_\ell^T$ (remember that $\underline{\underline{Q}}$ is unitary)

$$\underline{x}_c = \underline{\underline{Q}}_c \underline{x}_c$$

$$\frac{1}{n_2} \sum_{n=1}^{n_2} \left\| \underline{x}_c[n] - \underline{x}[n] \right\|^2 = \sum_{i=c+1}^{N} \lambda_i$$

So that if the ignored eigenvalues are small or null, the new representation of $\underline{x}[n]$ may be sufficiently accurate. This would reduce the computational complexity of the following stages.

For a linear regression problem the scheme may be the following



Note that even if the means have been removed from the original vector, and therefore $\underline{x}[n]$ and $\underline{x}_c[n]$ have zero means, the bias before the linear regressor may be necessary to take care of the means of $\underline{d}$.

Approaches to digital (FIR) filters

Wiener Filters

Adaptive filters

CMS

etc...