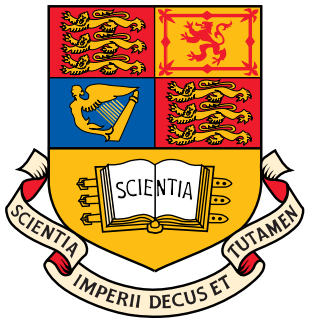
Advanced Signal Processing

Adaptive Estimation and Filtering

Danilo Mandic

room 813, ext: 46271



Department of Electrical and Electronic Engineering

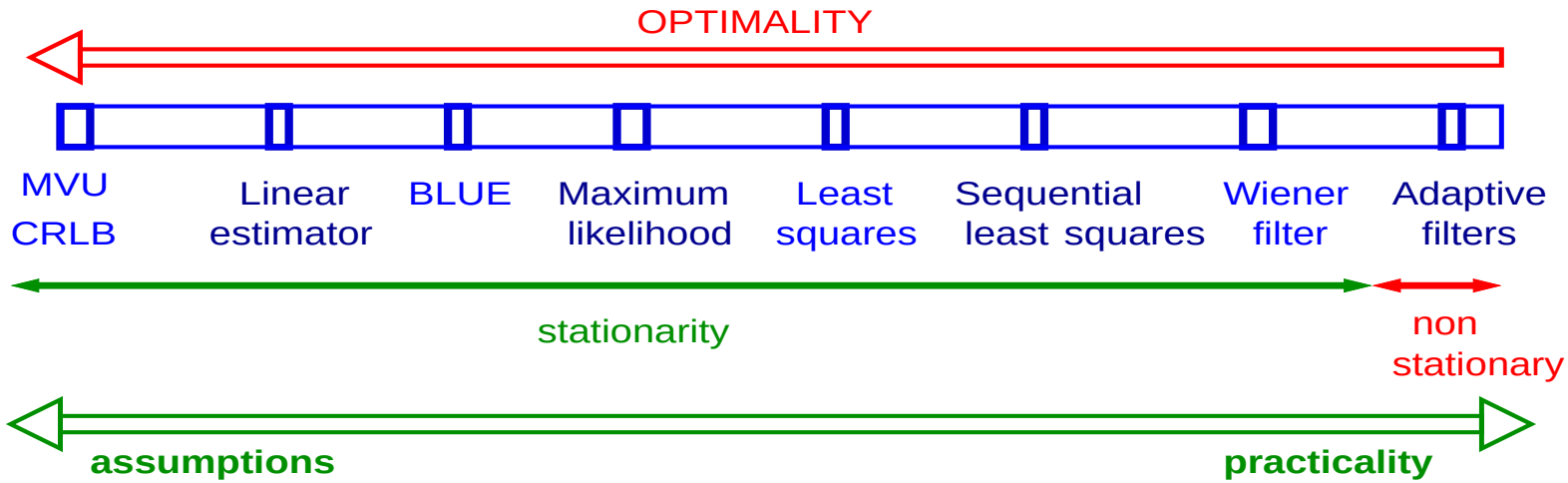
Imperial College London, UK

d.mandic@imperial.ac.uk, URL: www.commsp.ee.ic.ac.uk/∼mandic

Aims

- To introduce the concept of adaptive estimation
- Adaptive filters $\looparrowright$ "AR models with adaptive coefficients"
- The method of steepest descent
- Stochastic gradient and the Least Mean Square (LMS) algorithm
- Role of learning rate, bias and variance in estimation
- Adaptive filtering configurations (prediction, SYS ID, ...)
- Simple nonlinear structures (model of an artificial neuron)
- Stability and convergence of adaptive estimators
- Applications (also a link with your Coursework)

A big picture of estimators so far



- Minimum variance unbiased estimator (MVU), Cramer Rao Lower Bound (CRLB) $\looparrowright$ known pdf, linearity assumption, stationarity
- Linear model $\looparrowright$ known pdf, stationarity, and linearity
- Best linear unbiased estimator (BLUE) $\looparrowright$ linear estimator, data linear in the unknown parameter, stationarity, pdf not needed
- Maximum likelihood estimation (MLE) $\looparrowright$ stationarity, known pdf
- Least squares estimation (LS) $\looparrowright$ stationarity, deterministic data model
- Wiener filter $\looparrowright$ stationarity, no other assumptions
- **Adaptive filters $\looparrowright$ no assumptions**

Number guessing game
principle of adaptive estimation

Let us play a guessing game: One person will pick an integer between -100 and 100 and remember it, and the rest of us will try to discover that number in the following ways:

- Random guess with no feedback;
- Random guess followed by feedback $\rightsquigarrow$ the only information given is whether the guess was high or low;
- But we can make it a bit more complicated $\rightsquigarrow$ the guessed number may change along the iterations (nonstationarity).

Let us formalise this: If the current guess is denoted by $g_i(n)$, we can build a **recursive update** in the form

$$\begin{aligned} g_i(n+1) &= g_i(n) + \text{sign}\big(e(n)\big)\,\text{rand}\big[g_i(n), g_i(n-1)\big] \\ \text{new guess} &= \text{old guess} + \text{correction} \end{aligned}$$

Welcome to the wonderful world of adaptive filters!

Adaptive filters

basis for computational intelligence

The last equation was actually an adaptive filter in the form:

$$\begin{pmatrix} \text{New} \\ \text{Estimate} \end{pmatrix} = \begin{pmatrix} \text{Old} \\ \text{Estimate} \end{pmatrix} + \begin{pmatrix} \text{Correction} \\ \text{Term} \end{pmatrix}$$

Usually

$$\begin{pmatrix} \text{Correction} \\ \text{Term} \end{pmatrix} = \begin{pmatrix} \text{Learning} \\ \text{Rate} \end{pmatrix} \times \begin{pmatrix} \text{Function of} \\ \text{Input Data} \end{pmatrix} \times \begin{pmatrix} \text{Function of} \\ \text{Output Error} \end{pmatrix}$$

This is the very basis of learning in any adaptive machine

The most famous example is the **Least Mean Square (LMS)** algorithm, for which the parameter (weights) update equation is given by (more later)

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$$

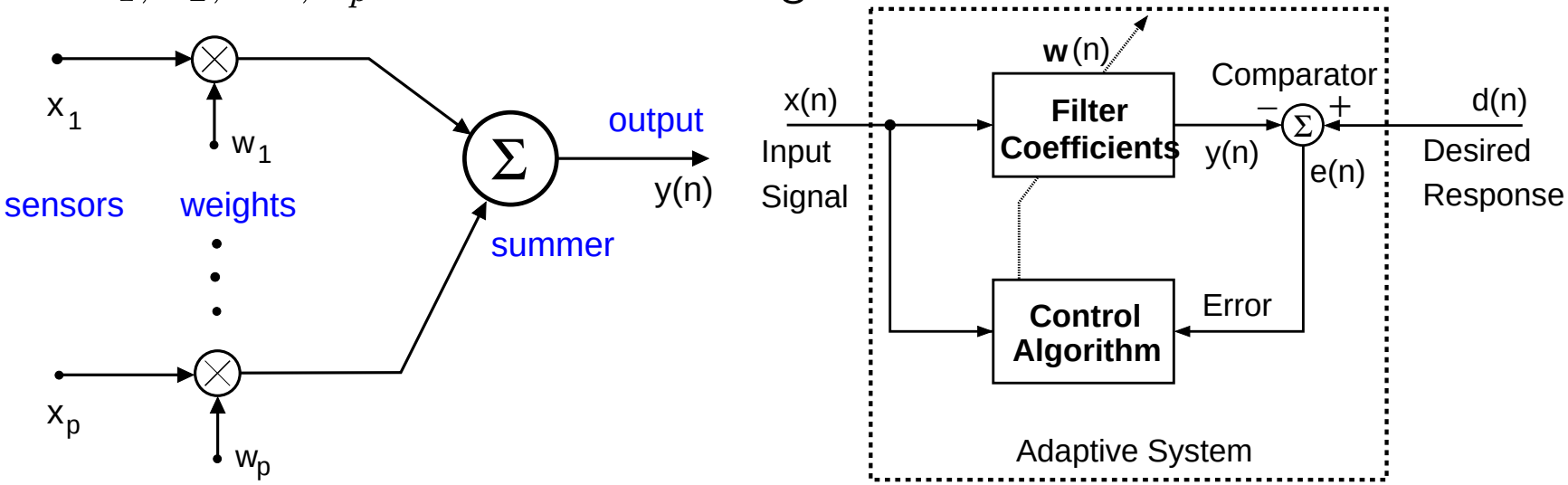
where $\mathbf{w}(n)$ are filter coefficients (time-varying), $\mathbf{x}(n) \in \mathbb{R}^{p\times 1}$ are input data in filter memory, $e(n)$ is the output error at time instant n, and μ is the learning rate (step size).

Problem formulation

from a fixed $\mathbf{h}$ in digital filters to a time–varying $\mathbf{w}(n)$ in adaptive filters

Consider a set of p sensors at different points in space (filter order p)

Let $x_1, x_2, \ldots, x_p$ be the individual signals from the sensors



- The sensor signals are weighted by the corresponding set of **time–varying** filter parameters $w_1(n), \ldots, w_p(n)$ (filter weights)
- The weighted signals are then summed to produce the output

$$y(n) = \sum_{i=1}^{p} w_i(n)x_i(n) = \mathbf{x}^T(n)\mathbf{w}(n) = \mathbf{w}^T(n)\mathbf{x}(n), \qquad n = 0, \ldots (\infty)$$

where $\mathbf{x}^T(n) = [x_1(n), \ldots, x_p(n)], \quad \mathbf{w}^T(n) = [w_1(n), \ldots, w_p(n)]$

Wiener–Hopf solution: The setting

Objective: To determine the **optimum** set of **fixed** weights $\mathbf{w}_o = [w_{o1}, \ldots, w_{op}]^T$ so as to minimize the difference between the system output and some desired response d in the mean square sense.

- The input-output relation of the filter is given by

$$y(n) = \sum_{k=1}^{p} w_k x_k(n)$$

- Let $\{d(n)\}$ denote the **desired response** or *target output* for the filter. Then, the **error signal** is

$$e(n) = d(n) - y(n)$$

- A natural **objective function** or **cost function** we wish to optimise is the **mean square error**, defined as (note the expectation $E\{\cdot\}$)

$$J = \frac{1}{2}E\{e^2(n)\} = \frac{1}{2}E\Big\{\big(d(n) - \sum_{k=1}^{p} w_k(n)x_k(n)\big)^2\Big\}$$

In other words, we wish to **minimise** the expected value of error power.

Wiener filter and the optimal filtering problem

The optimum filtering problem for a given signal $\mathbf{x}(n) = \{x_i(n)\}$:

Determine the optimum set of weights $\mathbf{w}_o = [w_{01}, \ldots, w_{0p}]^T$ (fixed) for which the mean squared error $J = E\{e^2(n)\}$ is minimum.

The solution to this problem is known as the Wiener filter.

The cost function is quadratic in the error (and thus in the weights), it is convex and has exactly one minimum J_{min} corresponding to $\mathbf{w}_o$, $J(\mathbf{w}_o)$.

$$J = \frac{1}{2}E\{e^2\} = \frac{1}{2}E\{d^2\} - E\left\{\sum_{k=1}^{p} w_k x_k d\right\} + \frac{1}{2}E\left\{\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k x_j x_k\right\}$$

where the "double summation" calculates the square of a sum, that is $\left(\sum_k\right)^2 = \sum_k \sum_j$

$$\text{Then} \quad J = \frac{1}{2}E\{d^2\} - \sum_{k=1}^{p} w_k E\{x_k d\} + \frac{1}{2}\sum_{j=1}^{p}\sum_{k=1}^{p} w_j w_k E\{x_j x_k\}$$

Wiener filter: Error surface

Introduce the notation:

$$r_d = E\{d^2\} \quad \rightarrow \quad \text{power of teaching (desired) signal}$$

$$r_{dx}(k) = E\{dx_k\}, \quad k = 1, 2, \ldots, p \quad \rightarrow \quad \text{crosscorrelation between } d \text{ \& } x_k$$

$$r_x(j,k) = E\{x_j x_k\}, \quad j, k = 1, 2, \ldots, p \quad \rightarrow \quad \text{autocorrelation at lag } (j-k)$$

Plug back into J to yield

$$J = \frac{1}{2} r_d - \sum_{k=1}^{p} w_k r_{dx}(k) + \frac{1}{2} \sum_{j=1}^{p} \sum_{k=1}^{p} w_j w_k r_x(j,k)$$

Definition: A multidimensional plot of the cost function J versus the weights (free parameters) $w_1, \ldots, w_p$ constitutes the **error performance surface** or simply the **error surface** of the filter.

The error surface is bowl–shaped with a well–defined bottom (global minimum point). It is precisely at this point where the spatial filter from Slide 6 is optimal in the sense that the mean squared error attains its minimum value $J_{min} = J(\mathbf{w}_o)$.

Recall that $J = J(e) = J(\mathbf{w})$, as **the unknown parameter is the weight vector.**

Finally, the Wiener solution
(fixed set of optimum weight $\looparrowright$ a static solution)

To determine the optimum weights, follow the least squares approach:

$$\nabla_{w_k} J = \frac{\partial J}{\partial w_k}, \qquad k = 1, \ldots, p$$

Differentiate wrt to w_k and set to zero to give

$$\nabla_{w_k} J = -r_{dx}(k) + \sum_{j=1}^{p} w_j r_x(j,k) = 0$$

Let w_{ok} denote the optimum value of weight w_k. Then, the optimum weights are determined by the following set of simultaneous equations

$$\sum_{j=1}^{p} w_{oj} r_x(j,k) = r_{xd}(k), \qquad k = 1, 2, \ldots, p$$

or in a compact form

$$\mathbf{w}_o = \mathbf{R}_{xx}^{-1} \mathbf{r}_{dx}$$

This system of equations is termed the **Wiener-Hopf** equations. The filter whose weights satisfy the Wiener-Hopf equations is called a **Wiener filter**. ($\mathbf{R}_{xx}$ is the input autocorrelation matrix and r_{dx} the vector of $\{r_{dx}\}$)

Notice, this is a block filter, operating on the whole set of data (non-sequential)

Vector-matrix formulation of the Wiener filter

The cost (error, objective) function can be expanded as

$$\begin{aligned} J &= \frac{1}{2}\, E\{e\ e^T\} = \frac{1}{2} E\Big\{ \big(d - \mathbf{w}^T\mathbf{x}\big)\big(d - \mathbf{w}^T\mathbf{x}\big)^T \Big\} \\ &= \frac{1}{2}\, E\{d^2 - d\mathbf{x}^T\mathbf{w} - d\mathbf{w}^T\mathbf{x} + \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w}\} \\ &= \frac{1}{2}\, E\{d^2 - 2d\mathbf{x}^T\mathbf{w} + \mathbf{w}^T\mathbf{x}\mathbf{x}^T\mathbf{w}\} \\ &= \frac{1}{2}\, E\{d^2\} - \frac{1}{2}\, 2\mathbf{w}^T E\{\mathbf{x}d\} + \frac{1}{2}\, \mathbf{w}^T E\{\mathbf{x}\mathbf{x}^T\}\mathbf{w} \end{aligned}$$

where the cross-correlation vector $\mathbf{p} \equiv E[\mathbf{x}d]^T$ and autocorr. matrix $\mathbf{R} \equiv E[\mathbf{x}\mathbf{x}^T]$

Thus, ($\mathbf{w}$ is still a fixed vector for the time being) the cost function

$$J = \sigma_d^2 - 2\mathbf{w}^T\mathbf{p} + \mathbf{w}^T\mathbf{R}\mathbf{w}$$

is quadratic in $\mathbf{w}$ and for a full rank $\mathbf{R}$, it has **a unique minimum**, $J(\mathbf{w}_o)$.

Now: For $J_{min} = J(\mathbf{w}_o) \quad \looparrowright \quad \partial J \,/\, \partial \mathbf{w} = -\mathbf{p} + \mathbf{R}\cdot\mathbf{w} = \mathbf{0} \quad \Rightarrow \quad -\mathbf{p} + \mathbf{R}\cdot\mathbf{w}_o = \mathbf{0}$

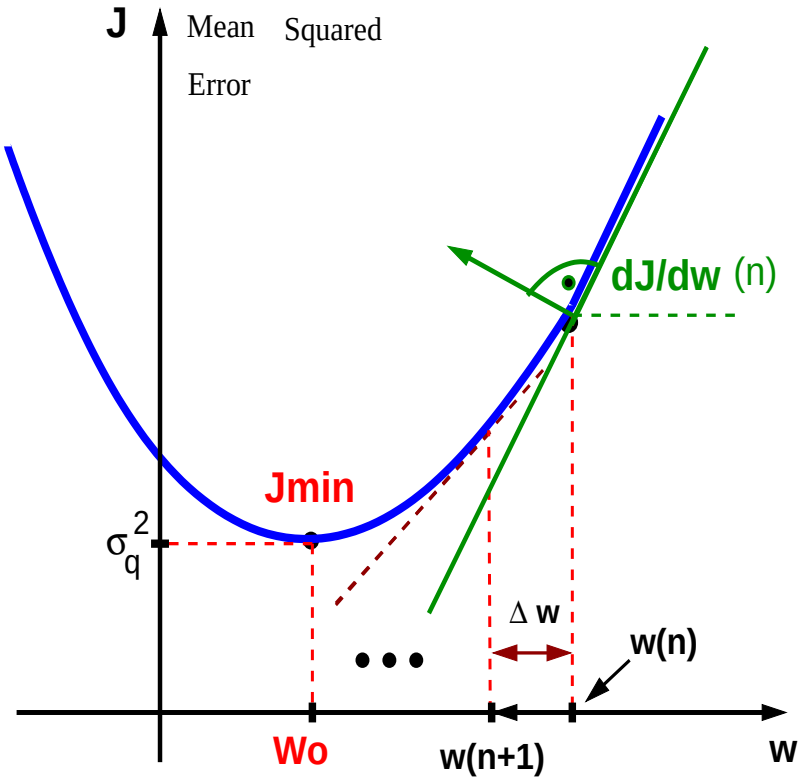
Finally $\mathbf{w}_o = \mathbf{R}^{-1}\mathbf{p}$ the co-called **Wiener–Hopf equation**

Method of steepest descent: Iterative Wiener solution

we reach $\mathbf{w}_o$ by an iteration $\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta\mathbf{w}(n) = \mathbf{w}(n) - \mu\nabla_{\mathbf{w}}J(n)$

Problem with the Wiener filter: it is computationally demanding to calculate the inverse of the possibly large correlation matrix $\mathbf{R}_{xx}$.

Solution: Allow the weights to have a **time–varying** form, so that they can be adjusted in an **iterative** fashion along the error surface.



This is achieved in the direction of steepest descent of error surface, that is, in a **direction opposite to the gradient vector** whose elements are defined by $\nabla_{w_k}J, \quad k = 1, 2, \ldots, p.$

For a teaching signal, assume

$$d(n) = \mathbf{x}^T(n)\mathbf{w}_o + q(n),$$

where $q \in \mathcal{N}(0, \sigma_q^2)$, so that we have $J_{min} = \sigma_q^2$

Method of steepest descent $\looparrowright$ continued

The gradient of the error surface of the filter wrt the weights now takes on a *time varying* form

$$\nabla_{w_k} J(n) = -r_{dx}(k) + \sum_{j=1}^{p} w_j(n) r_x(j,k) \qquad (*)$$

where the indices j, k refer to locations of different sensors in space, while the index n refers to iteration number.

According to the method of steepest descent, the adjustment applied to the weight $w_k(n)$ at iteration n is defined **along the direction of the negative of the gradient**, as

$$\Delta w_k(n) = -\mu \nabla_{w_k} J(n), \qquad k = 1, 2, \ldots, p$$

where μ is a small positive constant, $\mu \in \mathbb{R}^+$, called the **learning rate** parameter (also called step size, usually denoted by μ).

Method of steepest descent: Final form

Recall that $\nabla_{w_k} J(n) = -r_{dx}(k) + \sum_{j=1}^{p} w_j(n) r_x(j,k)$

Given the **current** value of the kth weight $w_k(n)$ at iteration n, the **updated** value of this weight at the next iteration $(n+1)$ is computed as

$$\begin{aligned} w_k(n+1) &= w_k(n) + \Delta w_k(n) = w_k(n) - \mu \nabla_{w_k} J(n) \\ \text{vector form} \quad \mathbf{w}(n+1) &= \mathbf{w}(n) + \Delta \mathbf{w}(n) = \mathbf{w}(n) - \mu \nabla_{\mathbf{w}} J(n) \end{aligned}$$

(updated filter weights) = (current weights) + (weight correction)

Upon combining with (*), we have (derive the vector form yourselves)

$$w_k(n+1) = w_k(n) + \mu \left[r_{dx}(k) - \sum_{j=1}^{p} w_j(n) r_x(j,k) \right], \quad k = 1, \ldots, p \quad (**)$$

The SD method is **exact** in the sense that no approximations are made in the derivation - the key difference is that the solution is obtained iteratively.

Observe that there is no matrix inverse in the update of filter weights!

Method of steepest descent: Have you noticed?

We now have an adaptive parameter estimator in the sense
'new parameter estimate'='old parameter estimate'+'update'

The derivation is based on minimising the mean squared error

$$J(n) = \frac{1}{2}E\{e^2(n)\}$$

For a spatial filter (sensor array), this cost function is an *ensemble average* taken at time n over an ensemble of *spatial filters* (e.g. nodes in sensor network).

For a temporal filter, the SD method can also be derived by minimising the *sum of error squares*

$$\mathcal{E}_{total} = \sum_{i=1}^{n} \mathcal{E}(i) = \frac{1}{2}\sum_{i=1}^{n} e^2(i)$$

In this case the ACF etc. are defined as *time averages* rather than ensemble averages. If the physical processes considered are *jointly ergodic* then we are justified in substituting time averages for ensemble averages.

The role of learning rate (also called "step size")

the step size governs the behaviour of gradient descent algorithms

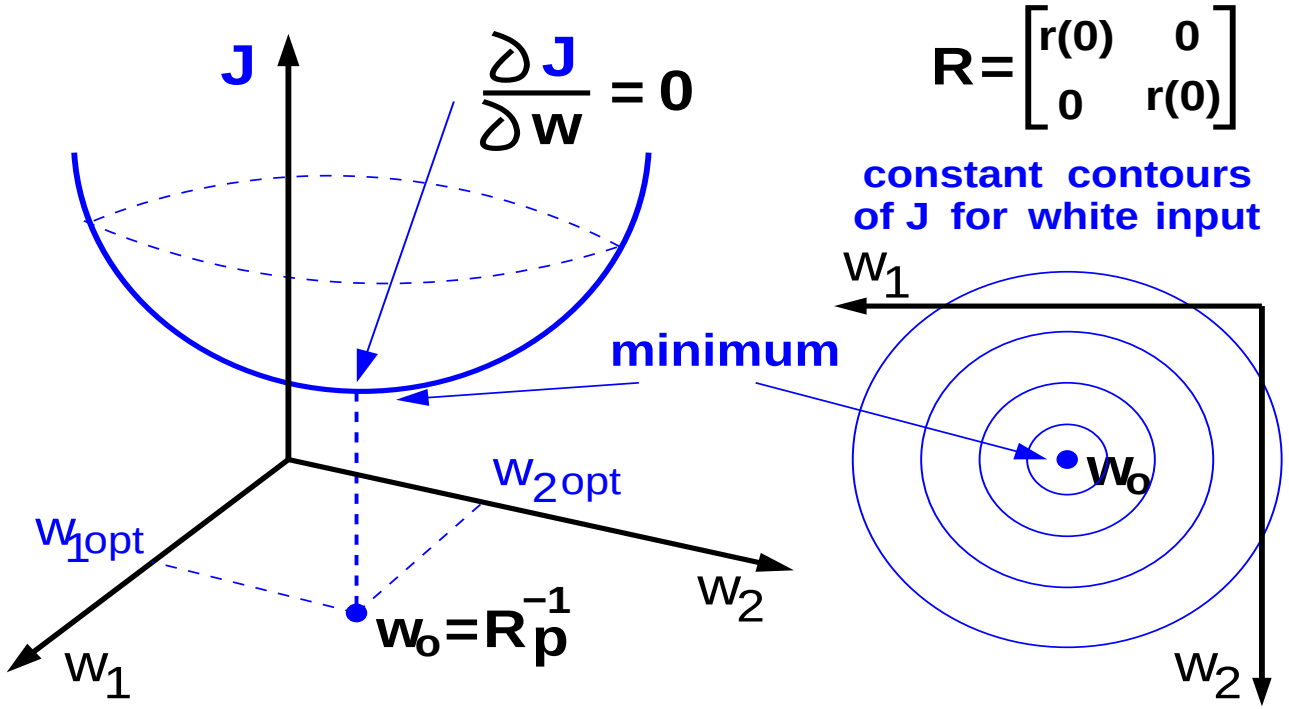
Care must be taken when selecting the learning rate μ, because:

- For μ small enough, the method of SD converges to a stationary point of the cost function $J(e) \equiv J(\mathbf{w})$, for which $\nabla_{\mathbf{w}} J(\mathbf{w}_0) = \mathbf{0}$. This stationary point can be a local or global minimum
- The method of steepest descent is an iterative procedure, and its behaviour depends on the value assigned to the step–size parameter μ
- When μ is small compared to a certain critical value μ_{crit}, the trajectory traced by the weight vector $\mathbf{w}(n)$ for increasing number of iterations, n, is overdamped
- When μ is allowed to approach (but remain less than) the critical value μ_{crit}, the trajectory is oscillatory or overdamped
- When μ exceeds μ_{crit}, the trajectory becomes unstable.

Condition $\mu < \mu_{crit}$ corresponds to a convergent or stable system, whereas condition $\mu > \mu_{crit}$ corresponds to a divergent or unstable system. Therefore, finding μ_{crit} defines a stability bound. (see Slide 23)

Learning rate and error performance surface

- These conditions are demonstrated for the two–dimensional case by plotting $w_1(n)$ versus $w_2(n)$ (elements of the two-dimensional $\mathbf{w}(n)$) for increasing n, and different values of μ.
- The distinguishing feature here is that it is possible for a linear system to find a global minimum of the cost function, whereas in a nonlinear adaptive system, we can have cases of global or local minima.



The Least Mean Square (LMS) algorithm

The LMS is based on the use of **instantaneous** estimates of the autocorrelation function $r_x(j,k)$ and the crosscorrelation function $r_{xd}(k)$

$$\hat{r}_x(j,k;n) = x_j(n)x_k(n) \qquad \hat{r}_{dx}(k;n) = x_k(n)d(n) \qquad J(n) = \frac{1}{2}e^2(n)$$

Substituting these into the method of steepest descent in (**) we have

$$\begin{aligned} w_k(n+1) &= w_k(n) + \mu\Big[x_k(n)d(n) - \sum_{j=1}^{p} w_j(n)x_j(n)x_k(n)\Big] \\ &= w_k(n) + \mu\Big[d(n) - \sum_{j=1}^{p} w_j(n)x_j(n)\Big]x_k(n) \\ &= w_k(n) + \mu[d(n) - y(n)]x_k(n) = w_k(n) + \mu e(n)x_k(n), \ \mathrm{k} = 1, ..., \mathrm{p} \end{aligned}$$

or in the vector form $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$

Because of the 'instantaneous statistics' used, the weights follow a "zig-zag" trajectory along the error surface, converging at the optimum solution $\mathbf{w}_0$, if μ is chosen properly.

The LMS algorithm $\looparrowright$ operates in an "unknown" environment

- The LMS operates in "unknown" environments, and the weight vector follows a **random** trajectory along the error performance surface
- Along the iterations, as $n \to \infty$ (steady state) the weights perform a random walk about the optimal solution $\mathbf{w}_0$ (measure of MSE)
- The cost function of LMS is based on an instantaneous estimate of the squared error. Consequently, the gradient vector in LMS is "random" and its direction accuracy improves "on the average" with increasing n

The LMS summary:

Initialisation. $w_k(0) = 0, \quad k = 1, \ldots, p \quad \equiv \quad \mathbf{w}(0) = \mathbf{0}$

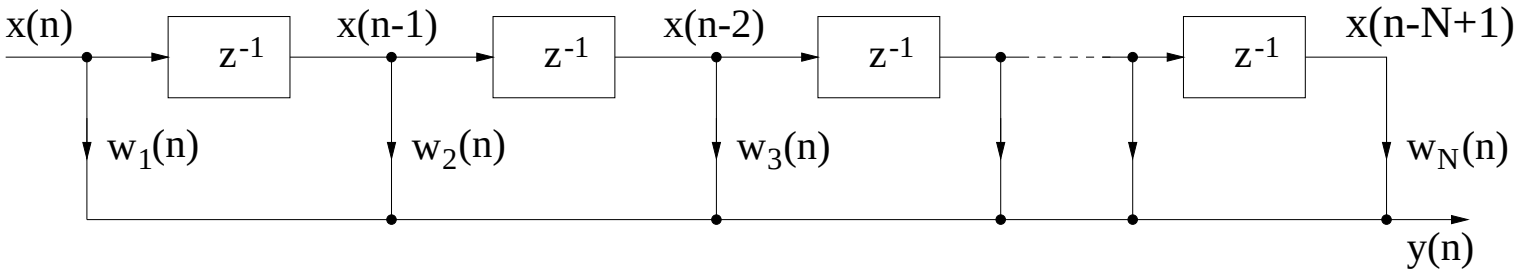
Filtering. For $n = 1, \ldots, (\infty)$ compute

$$y(n) = \sum_{j=1}^{p} w_j(n) x_j(n) = \mathbf{x}^T(n)\mathbf{w}(n) = \mathbf{w}^T(n)\mathbf{x}(n)$$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$$

Further perspective on LMS: Temporal problems

Our spatial problem with sensors signals: $\mathbf{x}(n) = [x_1(n), \ldots, x_N(n)]^T$ becomes a temporal one where $\mathbf{x}(n) = [x(n), \ldots, x(n-N+1)]^T$.



The output of this **temporal** filter of memory N is $y(n) = \mathbf{x}^T(n)\mathbf{w}(n)$

Alternative derivation of LMS: Since e(n)=d(n)-y(n)=$d(n) - \mathbf{x}^T(n)\mathbf{w}(n)$, then

$$J(n) = \frac{1}{2}\, e^2(n) \quad \rightarrow \quad \nabla_{\mathbf{w}} J(n) = \frac{\partial J(n)}{\partial \mathbf{w}(n)} = \frac{1}{2}\frac{\partial e^2(n)}{\partial e(n)}\frac{\partial e(n)}{\partial y(n)}\frac{\partial y(n)}{\partial \mathbf{w}(n)}$$

These partial gradients can be evaluated as

$$\frac{\partial e^2(n)}{\partial e(n)} = 2e(n), \quad \frac{\partial e(n)}{\partial y(n)} = -1, \quad \frac{\partial y(n)}{\partial \mathbf{w}(n)} = \mathbf{x}(n) \quad \Rightarrow \quad \frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\mathbf{x}(n)$$

Finally, we obtain the same LMS equations

Finally

$$\frac{\partial J(n)}{\partial \mathbf{w}(n)} = -\, e(n)\mathbf{x}(n)$$

The set of equations that describes the LMS is therefore given by

$$\begin{aligned} y(n) &= \sum_{i=1}^{N} x_i(n) w_i(n) \;=\; \mathbf{x}^T(n)\mathbf{w}(n) \\ e(n) &= d(n) - y(n) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) + \mu\, e(n)\mathbf{x}(n) \end{aligned}$$

◦ The LMS algorithm is a very simple yet extremely popular algorithm for adaptive filtering.
◦ LMS is robust (optimal in H^∞ sense) which justifies its practical utility.
◦ The forms of the spatial and temporal LMS are identical $\looparrowright$ it us up to us to apply adaptive filters according to the problem in hand

Convergence of LMS - parallels with MVU estimation - the parameter is the optimal filter weight vector $\mathbf{w}_o$

- **Convergence in the mean $\rightsquigarrow$ bias in parameter estimation** (think of the requirement for an unbiased optimal weight estimate)

$$E\{\mathbf{w}(n)\} \rightarrow \mathbf{w}_0 \qquad as \quad n \rightarrow \infty \quad \text{(steady state)}$$

- **Convergence in the mean square (MSE) $\rightsquigarrow$ estimator variance**, (fluctuation of the instantaneous weight vector estimates around $\mathbf{w}_o$)

$$E\{e^2(n)\} \rightarrow \text{constant} \qquad \text{as} \quad n \rightarrow \infty \quad \text{(steady state)}$$

We can write this since the error is a function of the filter weights.

☞ We expect the MSE convergence condition to be tighter: if LMS is convergent in the mean square, then it is convergent in the mean. The converse is not necessarily true (if an estimator is unbiased $\not\leftarrow$ it is not necessarily minimum variance $\not\leftrightsquigarrow$ if it is min. var. $\not\leftarrow$ likely unbiased).

☞ The logarithmic plot of the mean squared error (MSE) along time, $10 \log e^2(n)$ is called the **learning curve**.

For more on learning curves see your Coursework booklet and Slide 23

Example 1: Learning curves and performance measures

Task: Adaptively identify an AR(2) system given by

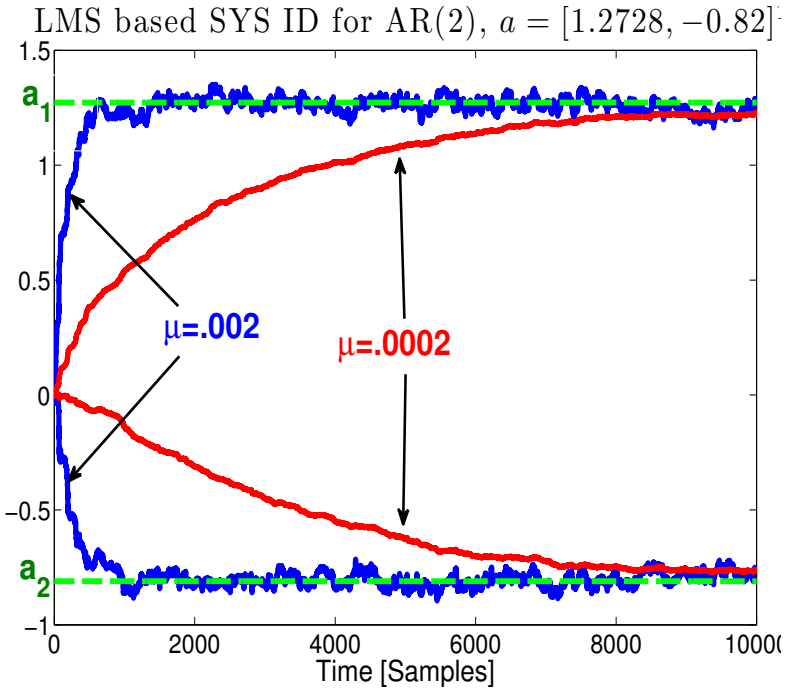
$$x(n) = 1.2728x(n-1) - 0.81x(n-2) + q(n), \quad q \sim \mathcal{N}(0, \sigma_q^2)$$

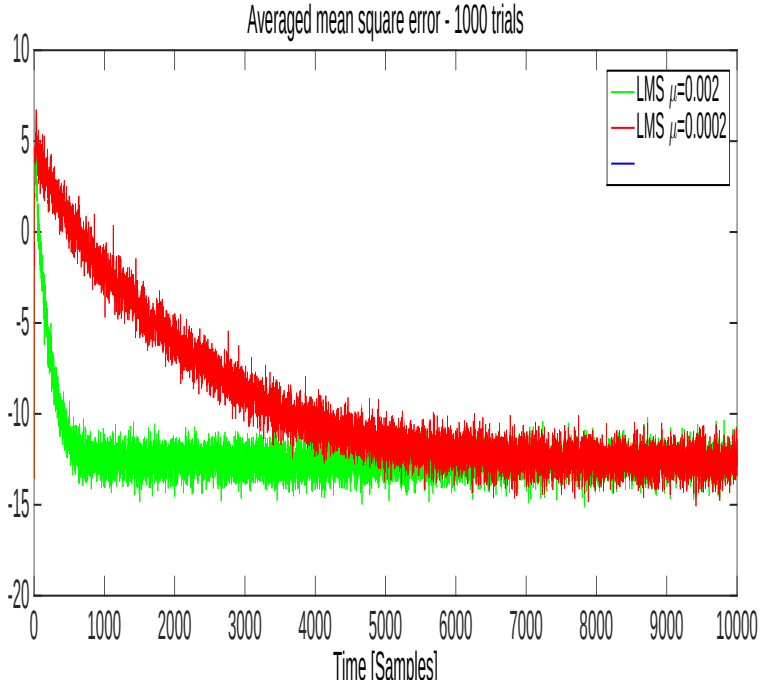
System model:

LMS and NLMS: $\hat{x}(n) = w_1(n)x(n-1) + w_2(n)x(n-2)$

Normalised LMS weights:

NLMS weights (i=1,2): $w_i(n+1) = w_i(n) + \dfrac{\mu}{\varepsilon + x^2(n-1) + x^2(n-2)} e(n)x(n-i)$





Convergence analysis of LMS: Mean– and Mean–Square convergence of the weight vector (Contour_Plot.m)

1) **Convergence in the mean**. Assume that the weight vector is uncorrelated with the input vector, $E\{\mathbf{w}(n)\mathbf{x}(n)\} = \mathbf{0}$, and $d \perp x$ (the usual "independence" assumptions but not true in practice)

   Therefore $$E\{\mathbf{w}(n+1)\} = [\mathbf{I} - \mu\mathbf{R}_{xx}]E\{\mathbf{w}(n)\} + \mu\mathbf{r}_{dx}$$

   The condition of convergence in the mean becomes (for i.i.d input)

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

   where λ_{max} is the largest eigenvalue of the autocorrelation matrix $\mathbf{R}_{xx}$.

2) **Mean square convergence.** The analysis is complicated and gives

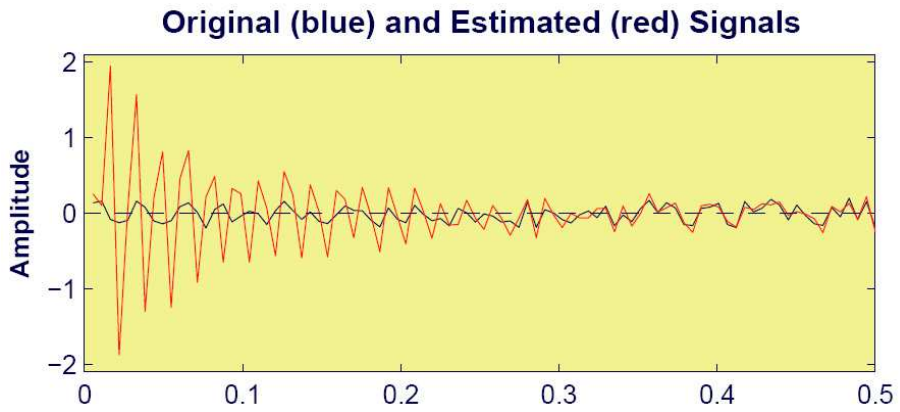
$$0 < \mu < \frac{2}{tr[\mathbf{R}_{xx}]} \qquad \text{using} \quad tr[\mathbf{R}_{xx}] = \sum_{k=1}^{p} \lambda_k \geq \lambda_{max}$$

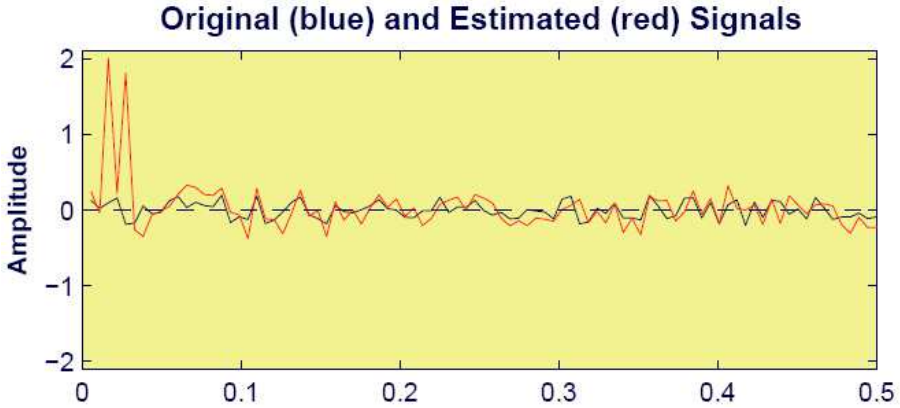
Since **"The trace" = "Total Input Power"**, we have

$$0 < \mu < \frac{2}{\text{total input power}}$$

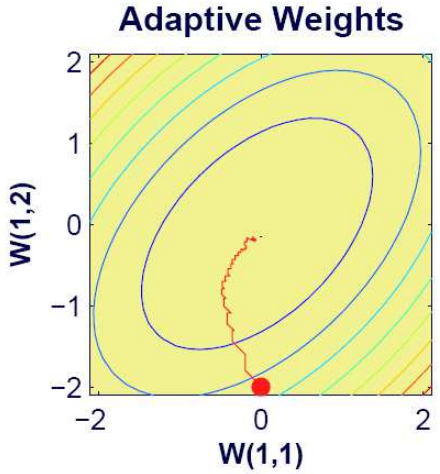
Example 2: Error surface for adaptive echo cancellation function 'nnd10nc' in Matlab

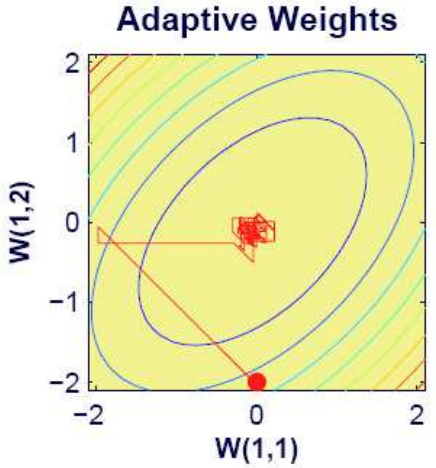
Original signal and its prediction





Error contour surface





Top panel - learning rate $\mu = 0.1$ Bottom panel - learning rate $\mu = 0.9$

Adaptive filtering configurations

ways to connect the filter, input, and teaching signal

- LMS can operate in a stationary or nonstationary environment
- LMS not only seeks for the minimum point of the error surface, but it also *tracks* it if $\mathbf{w}_o$ is time–varying
- The smaller the stepsizse μ the better the tracking behaviour (at steady state, in the MSE sense), however, this means slow adaptation.

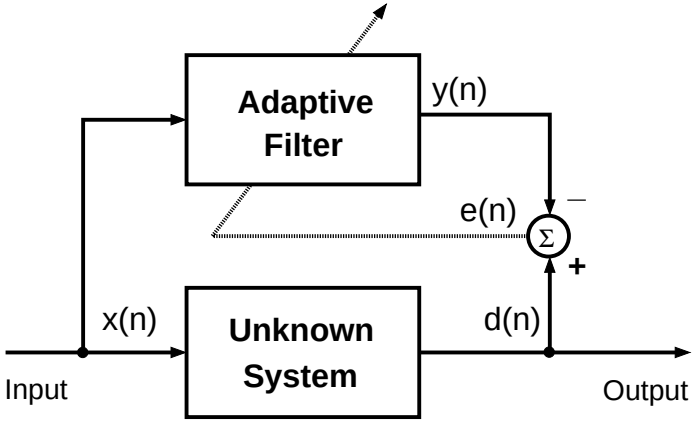
Adaptive filtering configurations:

- **Linear prediction.** The set of past values serves as the input vector, while the current input sample serves as the desired signal.
- **Inverse system modelling.** The adaptive filter is connected in series with the unknown system, whose parameters we wish to estimate.
- **Noise cancellation.** Reference noise serves as the input, while the measured noisy signal serves as the desired response, $d(n)$.
- **System identification.** The adaptive filter is connected in parallel to the unknown system, and their outputs are compared to produce the estimation error which drives the adaptation.

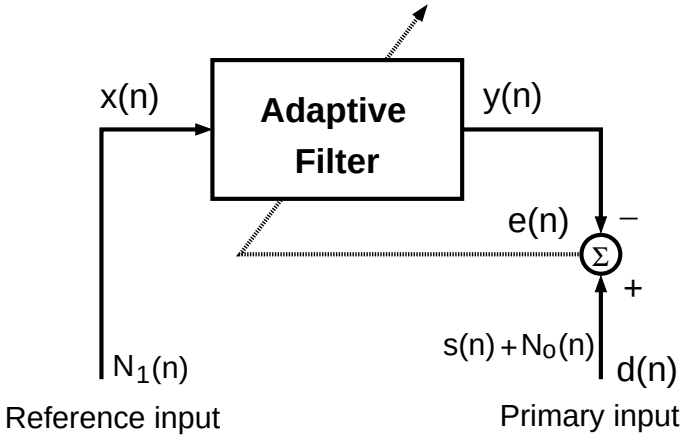
Adaptive filtering configurations: Block diagrams

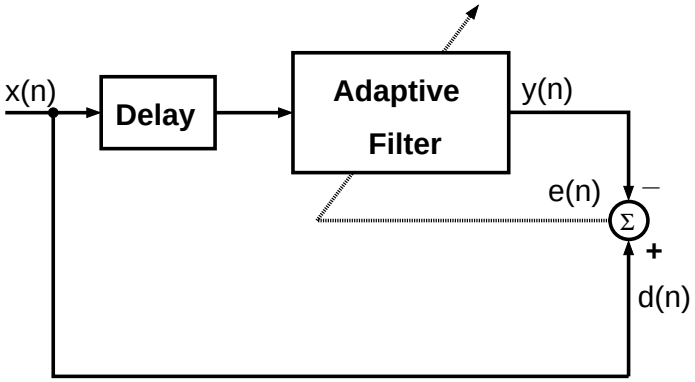
the same learning algorithm, e.g. the LMS, operates for any configuration

System identification

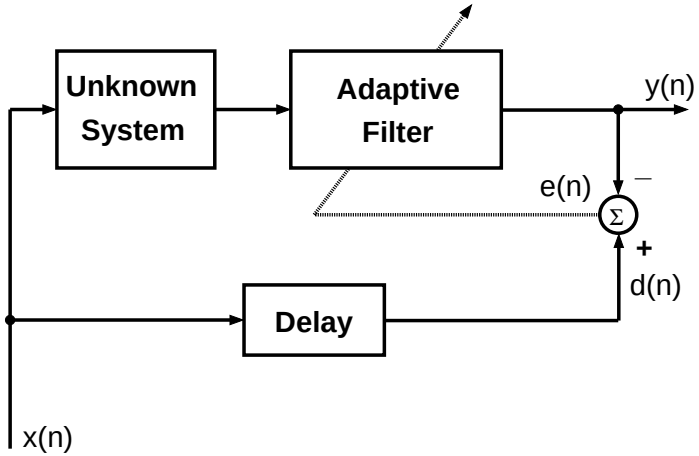


Noise cancellation





Adaptive prediction



Inverse system modelling

Applications of adaptive filters

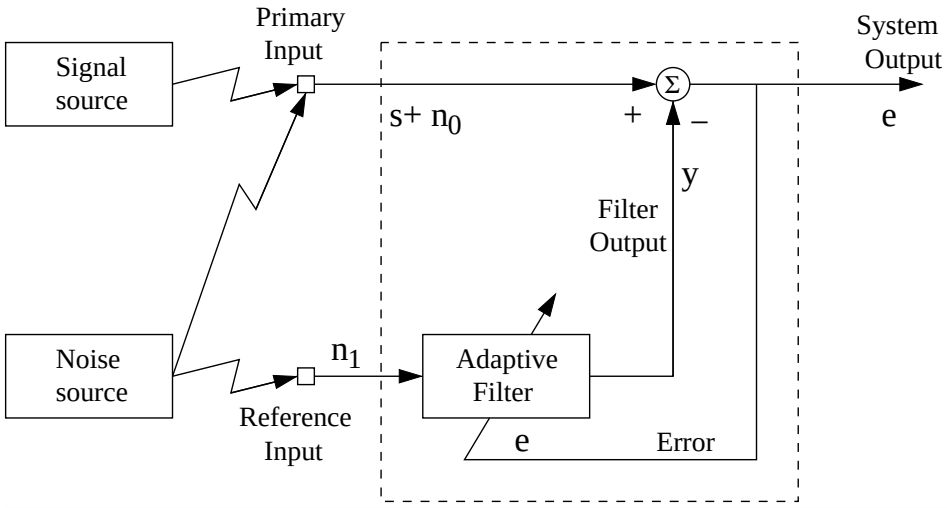
Adaptive filters have found an enormous number of applications.

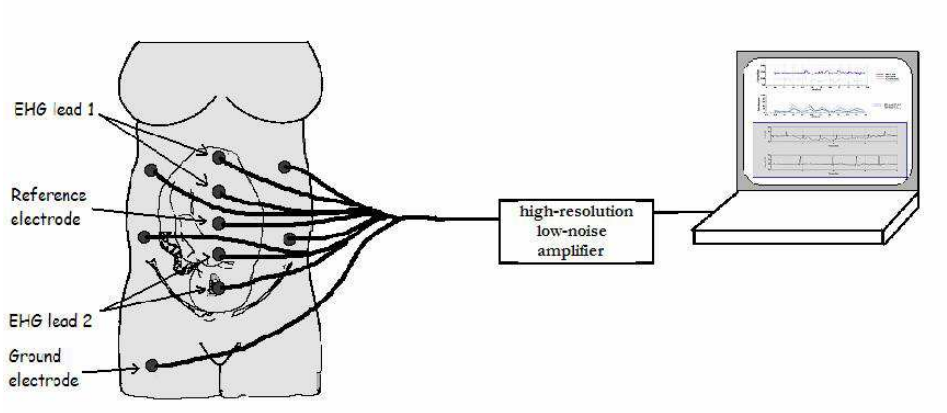
1. **Forward prediction** (the desired signal is the input signal advanced relative to the input of the adaptive filter). Applications in financial forecasting, wind prediction in renewable energy, power systems
2. **System identification** (the adaptive filter and the unknown system are connected in parallel and are fed with the same input signal $x(n)$). Applications in acoustic echo cancellation, feedback whistling removal in teleconference scenarios, hearing aids, power systems
3. **Inverse system modelling** (adaptive filter cascaded with the unknown system), as in channel equalisation in mobile telephony, wireless sensor networks, underwater communications, mobile sonar, mobile radar
4. **Noise cancellation** (the only requirement is that the noise in the primary input and the reference noise are correlated), as in noise removal from speech in mobile phones, denoising in biomedical scenarios, concert halls, hand-held multimedia recording

Example 3: Noise cancellation for foetal ECG recovery

Data acquisition

ANC with Reference



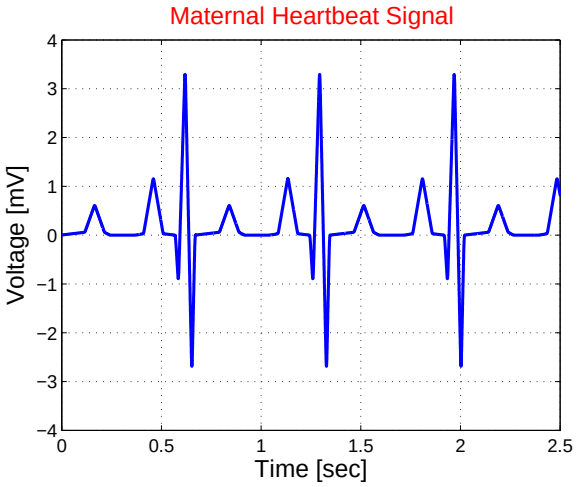


ECG recording (Reference electrode $\neq$ Reference input)

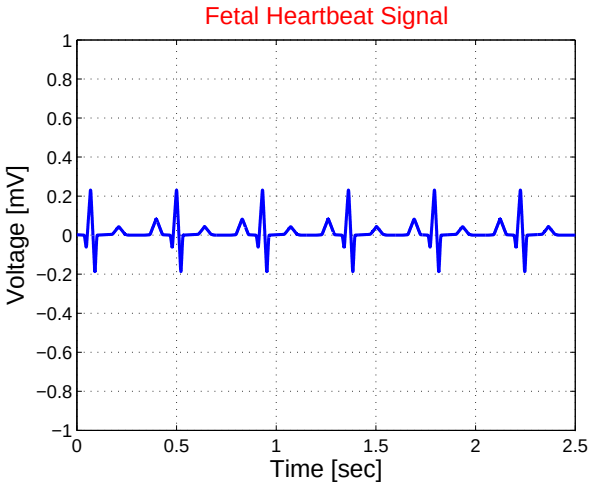
Example 3: Noise cancellation for foetal ECG estimation

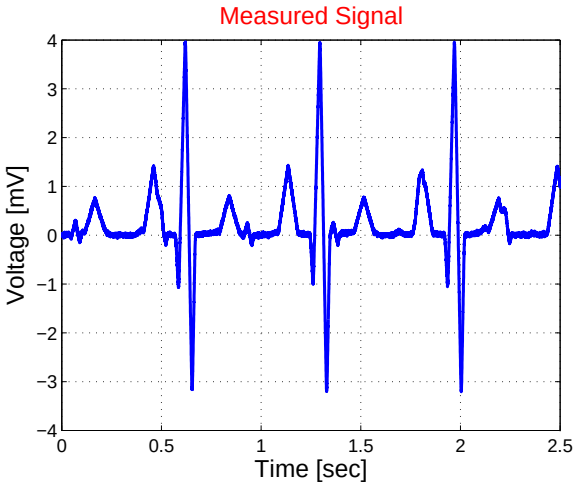
(similar to your CW Assignment IV)

Maternal ECG signal

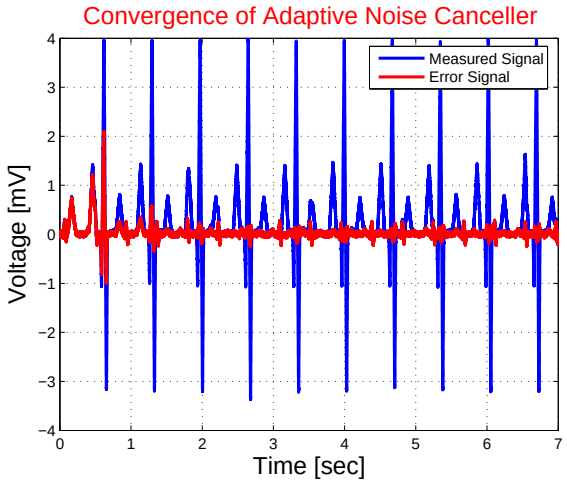


Foetal heartbeat



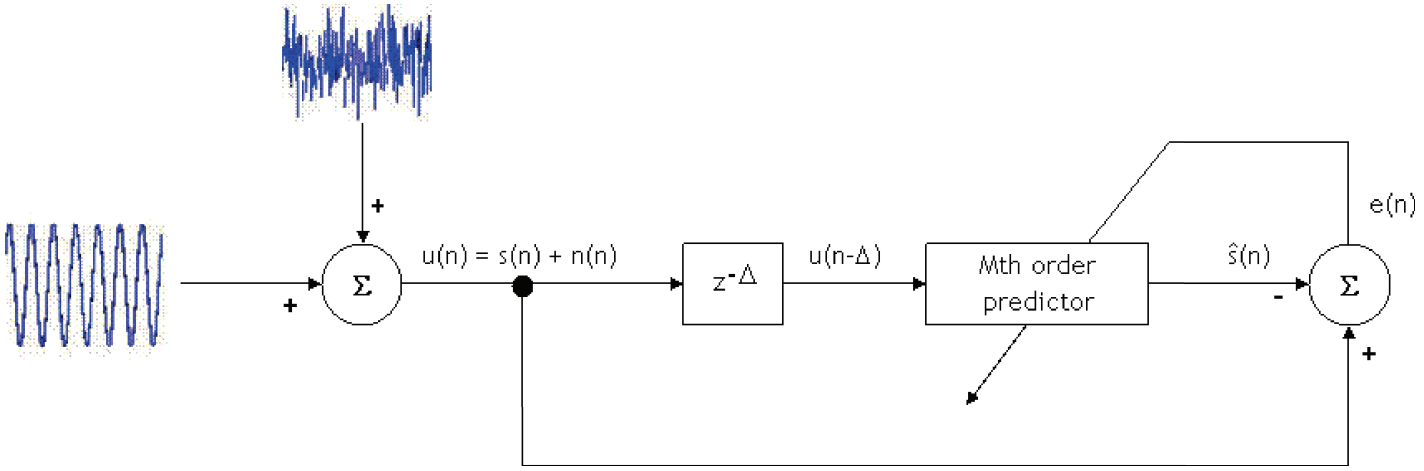


Measured foetal ECG



Maternal and foetal ECG

Example 4: Adaptive Line Enhancement (ALE), no need for referent noise due to self-decorrelation 'lms_fixed_demo'



Enhancement of a 100Hz signal in band–limited WN, with a $N = 30$ FIR LMS filter

- Adaptive line enhancement (ALE) refers to the case where a noisy signal, $u(n) =$ 'sin(n)' + 'w(n)', is filtered to obtain $\hat{u}(n) =$'sin(n)'
- ALE consists of a de-correlation stage, denoted by $z^{-\Delta}$ and an adaptive predictor. The value of Δ should be greater than the ACF lag for w
- The de-correlation stage attempts to remove any correlation that may exist between the samples of noise, by shifting them Δ samples apart
- This results in a phase shift at the output (input lags Δ steps behind)

Lecture summary

- Adaptive filters are simple, yet very powerful, estimators which do not require any assumptions on the data, and adjust their coefficients in an online adaptive manner
- In this way, they reach the optimal Wiener solution in a recursive fashion
- The steepest descent, LMS, NLMS, sign-algorithms etc. are **learning algorithms** which operate in certain **adaptive filtering configurations**
- Within each configuration, the function of the filter is determined by the way the input and teaching signal are connected to the filter (prediction, system identification, inverse system modelling, noise cancellation)
- The online adaptation makes adaptive filters suitable to operate in nonstationary environments, a typical case in practical applications
- Applications of adaptive filters are found everywhere (mobile phones, audio devices, biomedical, finance, seismics, radar, sonar, ...)
- Many modern, more complex, models are based on adaptive filters (neural networks, deep learning, reservoir computing, etc.)

Appendix 1: Wiener filter vs. Yule–Walker equations

we start from $x(n) = a_1(n)x(n-1) + \cdots + a_p(n)x(n-p) + q(n)$, q=white

The estimate: $y(n) = E\{x(n)\} = a_1(n)x(n-1) + \cdots + a_p(n)x(n-p)$

Teaching signal: $d(n)$, **Output error:** $e(n) = d(n) - y(n)$

1) Yule–Walker solution

Fixed coefficients $\mathbf{a}$ & $x(n) = y(n)$

Autoregressive modelling

$$
\begin{aligned}
r_{xx}(1) &= a_1 r_{xx}(0) + \cdots + a_p r_{xx}(p-1) \\
r_{xx}(2) &= a_1 r_{xx}(1) + \cdots + a_p r_{xx}(p-2) \\
\vdots &= \vdots \\
r_{xx}(p) &= a_1 r_{xx}(p-1) + \cdots + a_p r_{xx}(0) \\
\cdots & \quad \cdots \\
\mathbf{r}_{xx} &= \mathbf{R}_{xx}\mathbf{a}
\end{aligned}
$$

Solution: $\mathbf{a} = \mathbf{R}_{xx}^{-1}\mathbf{r}_{xx}$

2) Wiener–Hopf solution

Fixed optimal coeff. $\mathbf{w}_o = \mathbf{a}_{opt}$

$$J = E\{\frac{1}{2}e^2(n)\} = \sigma_d^2 - 2\mathbf{w}^T\mathbf{p} + \mathbf{w}^T\mathbf{R}\mathbf{w}$$

is quadratic in $\mathbf{w}$ and for a full rank $\mathbf{R}$, it has **one unique minimum.**

Now:

$$\frac{\partial J}{\partial \mathbf{w}} = -\mathbf{p} + \mathbf{R}\cdot\mathbf{w} = \mathbf{0}$$

Solution: $\mathbf{w}_0 = \mathbf{R}^{-1}\mathbf{p}$

Appendix 2: Improving the convergence and stability of LMS. The Normalised Least Mean Square (NLMS) alg.

Uses an adaptive step size by normalising μ by the signal power in the filter memory, that is

from fixed μ $\rightsquigarrow$ data adaptive $\mu(n) = \dfrac{\mu}{\mathbf{x}^T(n)\mathbf{x}(n)} = \dfrac{\mu}{\| \mathbf{x}(n) \|_2^2}$

Can be derived from the Taylor Series Expansion of the output error

$$e(n+1) = e(n) + \sum_{k=1}^{p} \frac{\partial e(n)}{\partial w_k(n)} \Delta w_k(n) + \underbrace{\text{higher order terms}}_{=0,\ since\ the\ filter\ is\ linear}$$

Since $\partial e(n)/\partial w_k(n) = -x_k(n)$ and $\Delta w_k(n) = \mu e(n) x_k(n)$, we have

$$e(n+1) = e(n)\Big[1 - \mu \sum_{k=1}^{p} x_k^2(n)\Big] = \big[1 - \mu \| \mathbf{x}(n) \|_2^2\big] \quad \text{as} \left(\sum_{k=1}^{p} x_k^2 = \| \mathbf{x} \|_2^2 \right)$$

To minimise the error, set $e(n+1) = 0$, to arrive at the NLMS step size:

$$\mu = \frac{1}{\| \mathbf{x}(n) \|_2^2} \qquad \text{however, in practice we use} \qquad \mu(n) = \frac{\mu}{\| \mathbf{x}(n) \|_2^2 + \varepsilon}$$

where $0 < \mu < 2$, $\mu(n)$ is time-varying, and ε is a small "regularisation" constant, added to avoid division by 0 for small values of input, $\| \mathbf{x} \| \rightarrow 0$

Appendix 3: Derivation of the formula for the convergence in the mean on Slide 24

◦ For the weights to converge in the mean (unbiased condition), we need

$$E\{\mathbf{w}(n)\} = \mathbf{w}_o \quad \text{as } n \to \infty \qquad \Leftrightarrow \qquad E\{\mathbf{w}(n+1)\} = E\{\mathbf{w}(n)\} \quad \text{as } n \to \infty$$

◦ The LMS update is given by: $\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n)$, then

$$\begin{aligned} E\{\mathbf{w}(n+1)\} &= E\Big\{\mathbf{w}(n) + \mu\big[d(n) - \mathbf{x}^T(n)\mathbf{w}(n)\big]\mathbf{x}(n)\Big\} \\ &\quad \text{since } \; d(n) - \mathbf{x}^T(n)\mathbf{w}(n) \; \text{ is a scalar, we can write} \\ &= E\Big\{\mathbf{w}(n) + \mu\mathbf{x}(n)\big[d(n) - \mathbf{x}^T(n)\mathbf{w}(n)\big]\Big\} \\ &= \big[\mathbf{I} - \mu \underbrace{E\{\mathbf{x}^T(n)\mathbf{x}(n)\}}_{\mathbf{R}_{xx}}\big]\mathbf{w}(n) + \mu \underbrace{E\{\mathbf{x}(n)d(n)\}}_{\mathbf{r}_{dx}} \\ &= \big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]\mathbf{w}(n) + \mu\mathbf{r}_{dx} = \big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]^n\mathbf{w}(0) + \mu\mathbf{r}_{dx} \end{aligned}$$

↬ the convergence is governed by the **homogeneous** part $\big[\mathbf{I} - \mu\mathbf{R}_{xx}\big]$

⇝ the filter will converge **in the mean** if $|\mathbf{I} - \mu\mathbf{R}_{xx}| < 1$

Appendix 4: Sign algorithms

Simplified LMS, derived based on $sign(e) = |e|/e$ and $\nabla|e| = sign(e)$.

Good for hardware and high speed applications.

- **The Sign Algorithm** (The cost function is $J(n) = |e(n)|$)

  Replace $e(n)$ by its sign to obtain

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu sign(e(n))\mathbf{x}(n)$$

- **The Signed Regressor Algorithm**

  Replace $\mathbf{x}(n)$ by $sign(\mathbf{x}(n)$

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n) sign(\mathbf{x}(n))$$

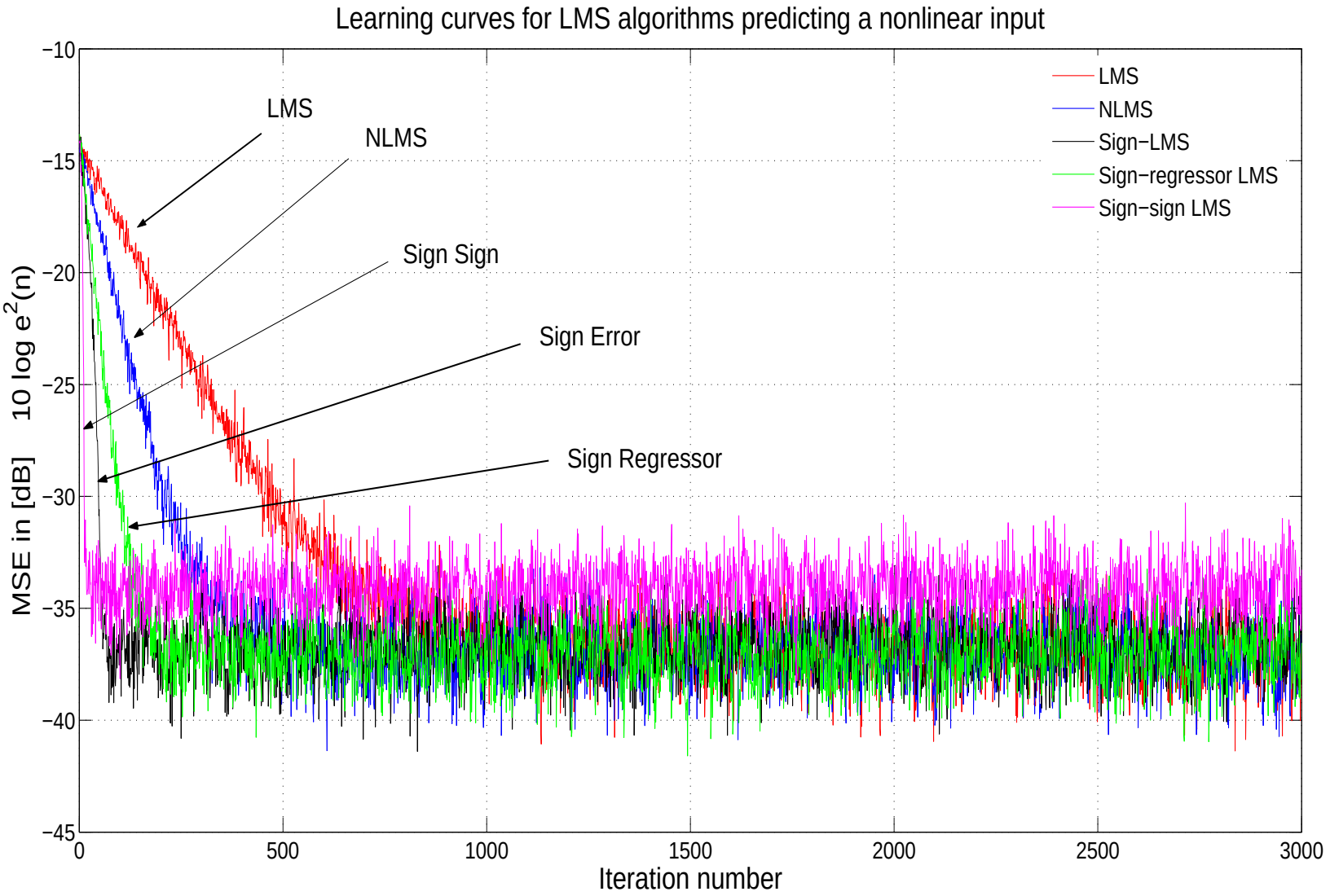
  Performs much better than the sign algorithm.

- **The Sign-Sign Algorithm**

  Combines the above two algorithms

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu sign(e(n)) sign(\mathbf{x}(n))$$

Appendix 4: Performance of sign algorithms



Notes

○

The next several slides introduce the concept of an artificial neuron – the building block of neural networks

Lecture supplement:

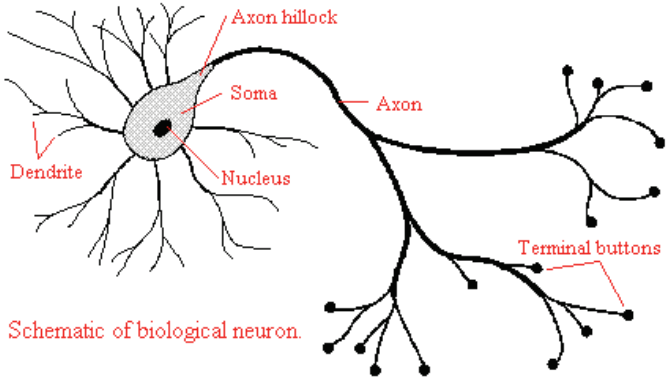
Elements of neural networks

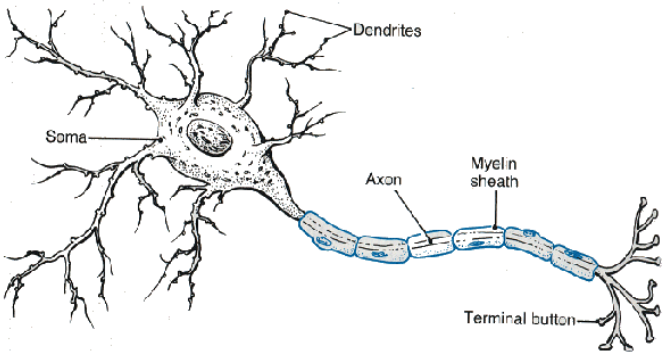
The need for nonlinear structures

There are situations in which the use of linear filters and models is suboptimal:

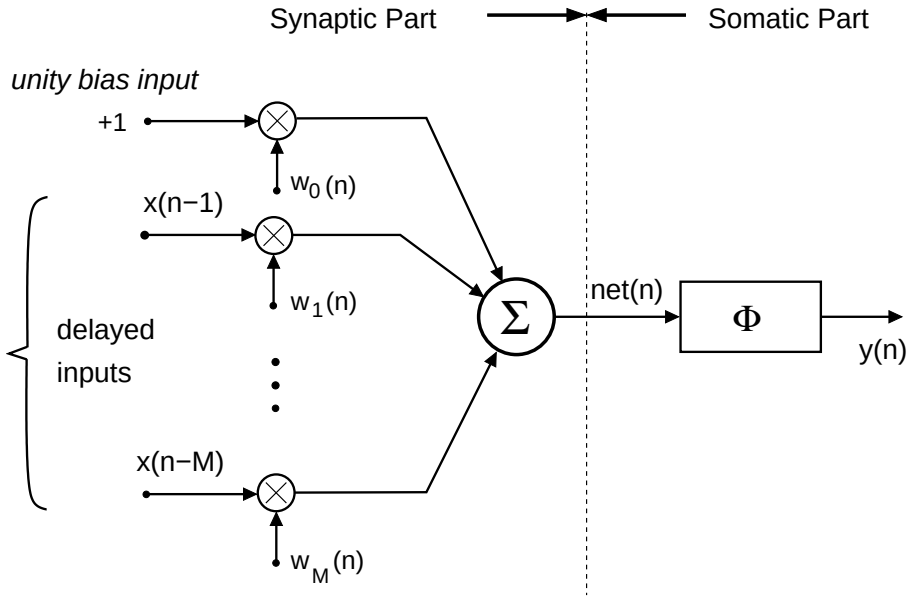
- when trying to identify dynamical signals/systems observed through a saturation type sensor nonlinearity, the use of linear models will be limited
- when separating signals with overlapping spectral components
- systems which are naturally nonlinear or signals that are non-Gaussian, such as limit cycles, bifurcations and fixed point dynamics, cannot be captured by linear models
- communications channels, for instance, often need nonlinear equalisers to achieve acceptable performance
- signals from humans (ECG, EEG, ...) are typically nonlinear and physiological noise is not white $\rightsquigarrow$ it is the so-called 'pink noise' or 'fractal noise' for which the spectrum $\sim 1/f$

An artificial neuron: The adaptive filtering model





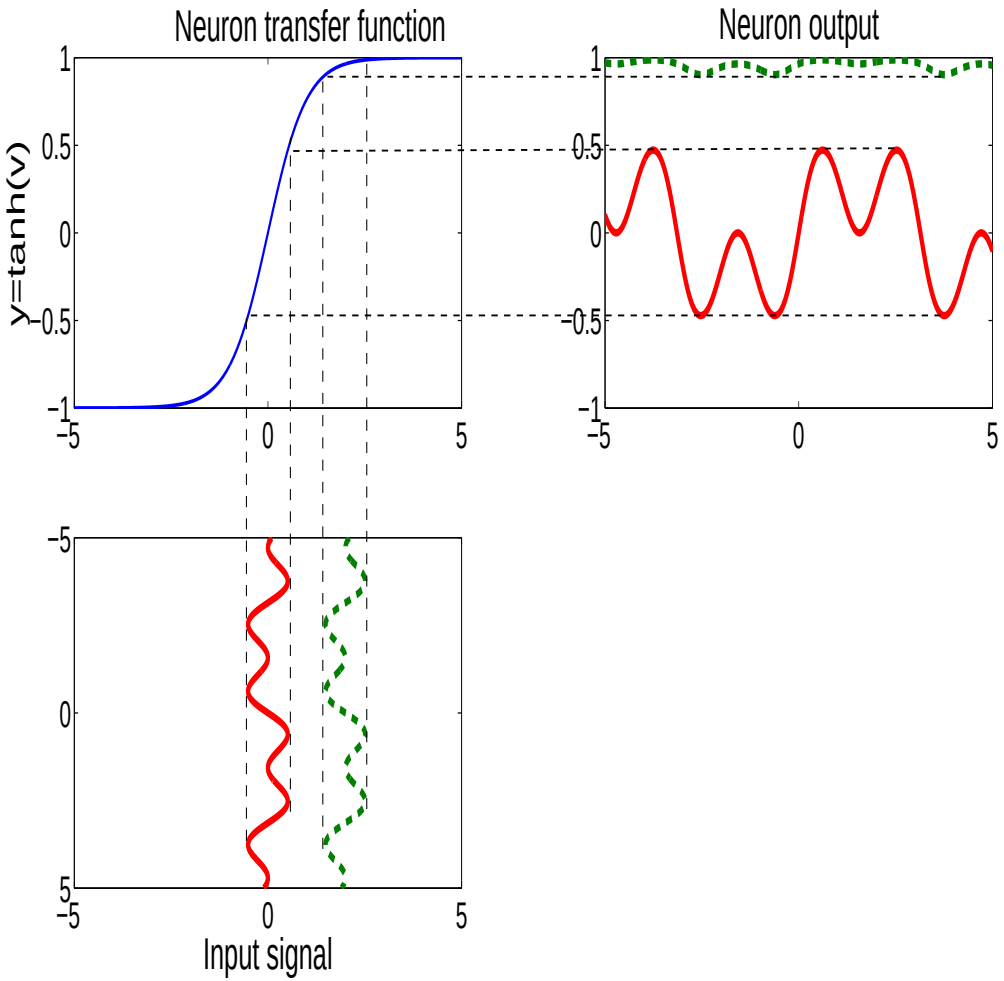
Biological neuron



Model of an artificial neuron

- delayed inputs $\mathbf{x}$
- bias input with unity value
- sumer and multipliers
- output nonlinearity

Effect of nonlinearity: An artificial neuron



Input: two identical signals with different DC offsets

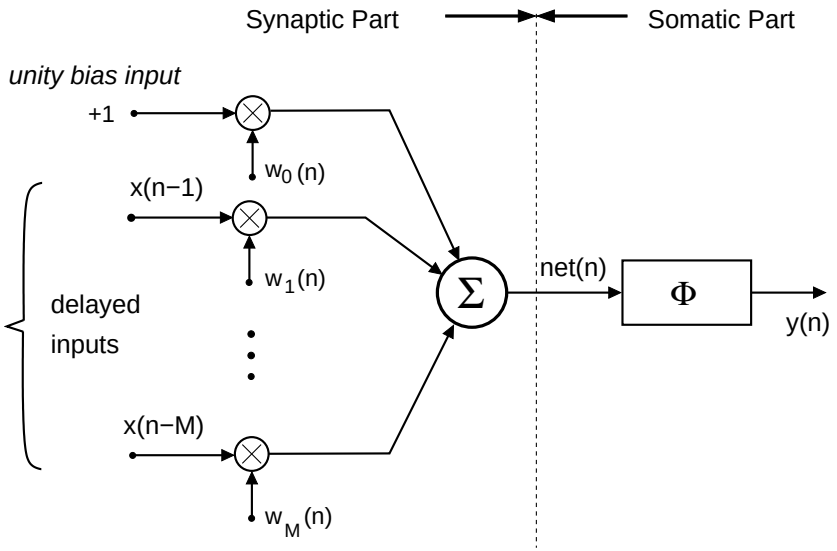
- observe the different behaviour depending on the operating point
- the output behaviour varies from amplifying and slightly distorting the input signal to attenuating and considerably distorting
- From the viewpoint of system theory, neural networks represent nonlinear maps, mapping one metric space to another.

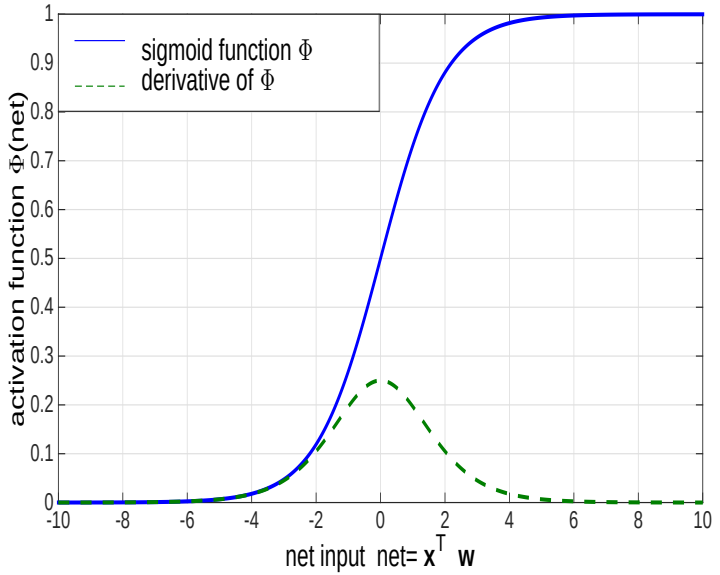
A simple nonlinear structure, referred to as the perceptron, or dynamical perceptron

- Consider a simple nonlinear FIR filter

  $$\text{Nonlinear FIR filter} = \text{standard FIR filter} + \text{memoryless nonlinearity}$$

- This nonlinearity is of a saturation type, like $\tanh$
- This structure can be seen as a single neuron with a dynamical FIR synapse. This FIR synapse provides memory to the neuron.

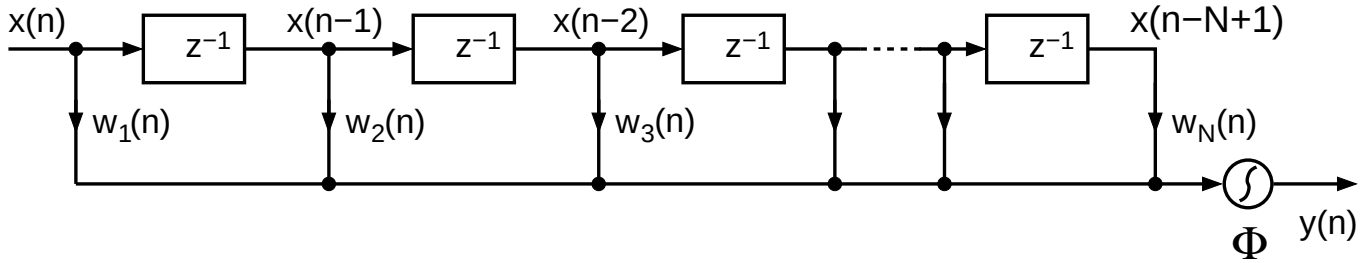




Model of artificial neuron (dynamical perceptron, nonlinear FIR filter)

Model of artificial neuron

This is the adaptive filtering model of every single neuron in our brains



The output of this filter is given by

$$y(n) = \Phi\left(\mathbf{x}^T(n)\mathbf{w}(n)\right)$$

The nonlinearity $\Phi(\cdot)$ after the tap–delay line is typically the so-called sigmoid, a saturation-type nonlinearity like that on the previous slide.

$$\begin{aligned} e(n) &= d(n) - \Phi\left(\mathbf{x}^T(n)\mathbf{w}(n)\right) \\ \mathbf{w}(n+1) &= \mathbf{w}(n) - \mu\nabla_{\mathbf{w}(n)}J(n) \end{aligned}$$

where $e(n)$ is the instantaneous error at the output neuron, $d(n)$ is some teaching (desired) signal, $\mathbf{w}(n) = [w_1(n), \ldots, w_N(n)]^T$ is the weight vector, and $\mathbf{x}(n) = [x_1(n), \ldots, x_N(n)]^T$ is the input vector.

Dynamical perceptron: Learning algorithm

Using the ideas from LMS, the cost function is given by

$$J(n) = \frac{1}{2}e^2(n)$$

Gradient $\nabla_{\mathbf{w}(n)}J(n)$ calculated from

$$\frac{\partial J(n)}{\partial \mathbf{w}(n)} = e(n)\frac{\partial e(n)}{\partial \mathbf{w}(n)} = -\,e(n)\,\Phi'\left(\mathbf{x}^T(n)\mathbf{w}(n)\right)\mathbf{x}(n)$$

where $\Phi'(\mathbf{x}^T(n)\mathbf{w}(n)) = \Phi'(n)$ denotes the first derivative of $\Phi(\cdot)$.

The weight update equation becomes

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\Phi'\left(\mathbf{x}^T(n)\mathbf{w}(n)\right)e(n)\mathbf{x}(n) = \mathbf{w}(n) + \mu\Phi'(n)e(n)\mathbf{x}(n)$$

◦ This filter is BIBO (bounded input bounded output) stable, as the output range is limited due to the saturation type of nonlinearity Φ.

◦ Also, for large inputs, due to the saturation in the nonlinearity, for large arguments $\Phi' \approx 0$ and the above equation is not updated.

Notes

○