# Adaptive SP & Machine Intelligence

## Advanced Learning Systems and Neural Networks

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

## Outline:

### Really, to de-mystify and make rigorous several concepts

- Recursive 'deterministic' solution to the Wiener filter $\looparrowright$ Recursive Least Squares (RLS) (as opposed to the 'stochastic' LMS solution)
- A general framework for training learning systems
- Connection between LMS, RLS, and Kalman filter
- Incorporation of constraints (sparsity, smoothness, non-negativity)
- The concept of artificial neuron, dynamical perceptron, and perceptron learning rule (effectively a nonlinear adaptive filter)
- Neural networks (NNs), multilayer perceptron, the backpropagation algorithm, and nonlinear separation of patterns
- From feedforward (shallow) NNs to deep neural networks (DNNs)
- Recurrent neural networks and RTRL algorithm
- Applications

## The big picture of learning machines two main families of algorithms

- **Gradient descent methods** (e.g. the Least Mean Square (LMS)) provide an **approximate minimisation** of the mean square error (MSE)

  $$J \sim E\{e^2(n)\} \quad \text{for LMS} \quad J(n) \sim e^2(n) \qquad \textbf{stochastic}$$

  ◦ Knowledge of the autocorrelation of the input process and cross-correlation between the input an teaching signal required.

  ◦ Rapid convergence or sufficiently small excess MSE not guaranteed.

- **Least squares (LS) techniques** are based on the **exact minimisation** of a sum of instantaneous squared errors

  $$J(n) = \sum_{i=0}^{n} e^2(i) = \sum_{i=0}^{n} \left| d(i) - \mathbf{x}^T(i)\mathbf{w}_n \right|^2 \qquad \textbf{deterministic}$$

  Deterministic cost function $\looparrowright$ no statistical information about $\mathbf{x}$ and $d$!

☞ The 'deterministic' LS error depends on a particular combination of $x(n)$ and $d(n)$ $\looparrowright$ for different signals we obtain different filters.

☞ The 'stochastic' MSE does not depend on a particular realisation of $x(n)$ and $d(n)$ $\looparrowright$ produces equal weights for signals with the same statistics.

## The Recursive Least Squares (RLS) algorithm

### (recursive solution to the "deterministic" Wiener filtering problem)

**Aim:** For a filter of order $N$, find the weight vector $\mathbf{w}_n = \mathbf{w}_{opt}(n)$ which minimizes the sum of squares of output errors up until the time instant $n$

$$\mathbf{w}_n = \mathbf{w}_{opt}^{LS} \text{ over n time instants} \quad \Rightarrow \quad \mathbf{w}_n = arg \ \min_{\mathbf{w}} \sum_{i=0}^{n} \left| d(i) - \mathbf{x}^T(i)\mathbf{w}_n \right|^2$$

(the summation over $i$ is performed for the **latest** set of coefficients $\mathbf{w}_n$)

☞ Notice, the weights $\mathbf{w}_n$ are **held constant** over the whole observation $[0, n]$ – similar to the Wiener setting (but a deterministic cost function)

**To find $\mathbf{w}_n$:** set $\nabla_{\mathbf{w}} J(n) = \mathbf{0}$ to solve for $\mathbf{w}_{opt}(n) = \mathbf{w}_n = \mathbf{R}^{-1}(n)\mathbf{p}(n)$

☞ **Careful here, as:** $\mathbf{R}(n) = \sum_{i=0}^{n} \mathbf{x}(i)\mathbf{x}^T(i), \quad \mathbf{p}(n) = \mathbf{r}_{d\mathbf{x}} = \sum_{i=0}^{n} d(i)\mathbf{x}(i)$

that is, $\mathbf{R}(n)$, $\mathbf{p}(n)$ are purely ***deterministic*** (*cf.* $\mathbf{R} = E\{\mathbf{x}\mathbf{x}^T\}, \mathbf{p} = E\{d\mathbf{x}\}$ in Wiener filt.)

**Recursive least squares (RLS) summary.** The aim is to recursively update:

(a) $J(n+1) = J(n) + |e(n+1)|^2$ (b) $\mathbf{p}(n+1) = \mathbf{p}(n) + d(n+1)\mathbf{x}(n+1)$

(c) $\mathbf{R}(n+1) = \mathbf{R}(n) + \mathbf{x}(n+1)\mathbf{x}^T(n+1)$

The weight update: $\mathbf{w}_{n+1} = \mathbf{w}_{opt}(n+1) = \mathbf{w}(n+1) = \mathbf{R}^{-1}(n+1)\mathbf{p}(n+1)$

## Ways to compute a matrix inverse

**while direct, this technique is computationally wasteful, $N^3 + 2N^2 + N$ multiplic.**

There are many ways (Neumann series, geometric series). For $|a| < 1$

$$\sum_{i=0}^{\infty} a^i = \frac{1}{1-a}, \qquad \text{now swap} \quad \mathrm{b} = 1 - \mathrm{a} \quad \text{to get} \quad \sum_{\mathrm{i}=1}^{\infty} (1-\mathrm{b})^{\mathrm{i}} = \frac{1}{\mathrm{b}}$$

☞ **we have found the inverse of $b$ via a series in $1 - b$.**

**Generalization to matrices:** Consider a nonsingular matrix $\mathbf{A}$, to yield

$$\lim_{i = \rightarrow \infty} (\mathbf{I} - \mathbf{A})^i = \mathbf{0} \quad \Rightarrow \quad \mathbf{A}^{-1} = \sum_{i=0}^{\infty} (\mathbf{I} - \mathbf{A})^i$$

Then, for an invertible matrix $\mathbf{X}$, and its inverse $\mathbf{A}$, we can write

$$\lim_{i \rightarrow \infty} (\mathbf{I} - \mathbf{X}^{-1}\mathbf{A})^i = \lim_{i \rightarrow \infty} (\mathbf{I} - \mathbf{A}\mathbf{X}^{-1})^i = \mathbf{0} \quad \Rightarrow \quad \mathbf{A}^{-1} = \sum_{i=0}^{\infty} \Big(\mathbf{X}^{-1}(\mathbf{X} - \mathbf{A})\Big)^i \mathbf{X}^{-1}$$

If $(\mathbf{A} - \mathbf{X})$ has rank 1, we finally have: $\mathbf{A}^{-1} = \mathbf{X}^{-1} + \dfrac{\mathbf{X}^{-1}\big(\mathbf{X} - \mathbf{A}\big)\mathbf{X}^{-1}}{1 - \mathrm{tr}\Big(\mathbf{X}^{-1}\big(\mathbf{X} - \mathbf{A}\big)\Big)}$

## The RLS formulation $\looparrowright$ $\mathbf{R}^{-1}$ calculation is $\mathcal{O}(N^3)$

**(we use the standard matrix inversion lemma, also known as Woodbury's identity)**

**Key to RLS:** all we have to do is to employ a recursive update of $\mathbf{R}^{-1}$

Start from the matrix inversion lemma (also known as ABCD lemma)

$$(A+BCD)^{-1} = A^{-1} - A^{-1}B\big(DA^{-1}B + C^{-1}\big)^{-1}DA^{-1}$$

and set $A = \mathbf{R}(n), B = \mathbf{x}(n+1), C = 1, D = \mathbf{x}^T(n+1)$, to give

$$\mathbf{R}(n+1) \quad = \quad \mathbf{R}(n) + \mathbf{x}(n+1)\mathbf{x}^T(n+1) = A + BCD$$

Then $\mathbf{R}^{-1}(n+1)$ is given by

$$\mathbf{R}^{-1}(n+1) \quad = \quad \mathbf{R}^{-1}(n) - \frac{\mathbf{R}^{-1}(n)\mathbf{x}(n+1)\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)}{\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)\mathbf{x}(n+1)+1} \quad \looparrowright \quad \mathcal{O}(N^2)$$

$\looparrowright$ **we never compute $\mathbf{R}(n+1)$ or $\mathbf{R}^{-1}(n)$ directly, but recursively**

The optimal RLS weight vector: $\mathbf{w}_{opt}(n+1) = \mathbf{w}(n+1) + \mathbf{R}^{-1}(n+1)\mathbf{p}(n+1)$

Minimum LSE: $J^{LS}_{min} = \| \mathbf{d}(n) \|_2^2 - \mathbf{r}^T_{dx}(n)\mathbf{w}_n$

where $\mathbf{r}_{dx}(n) = \mathbf{p}(n), \mathbf{d}(n) = [d(0), \ldots, d(n)]^T$ (also see the Appendix)

## RLS: Adaptive noise cancellation

### The system has two microphones (primary & reference) *`rlsdemo` in Matlab

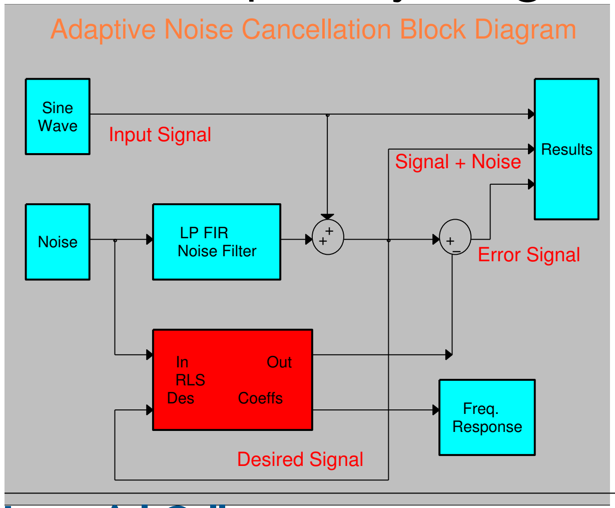
Similar scenario to noise-cancelling headphones.

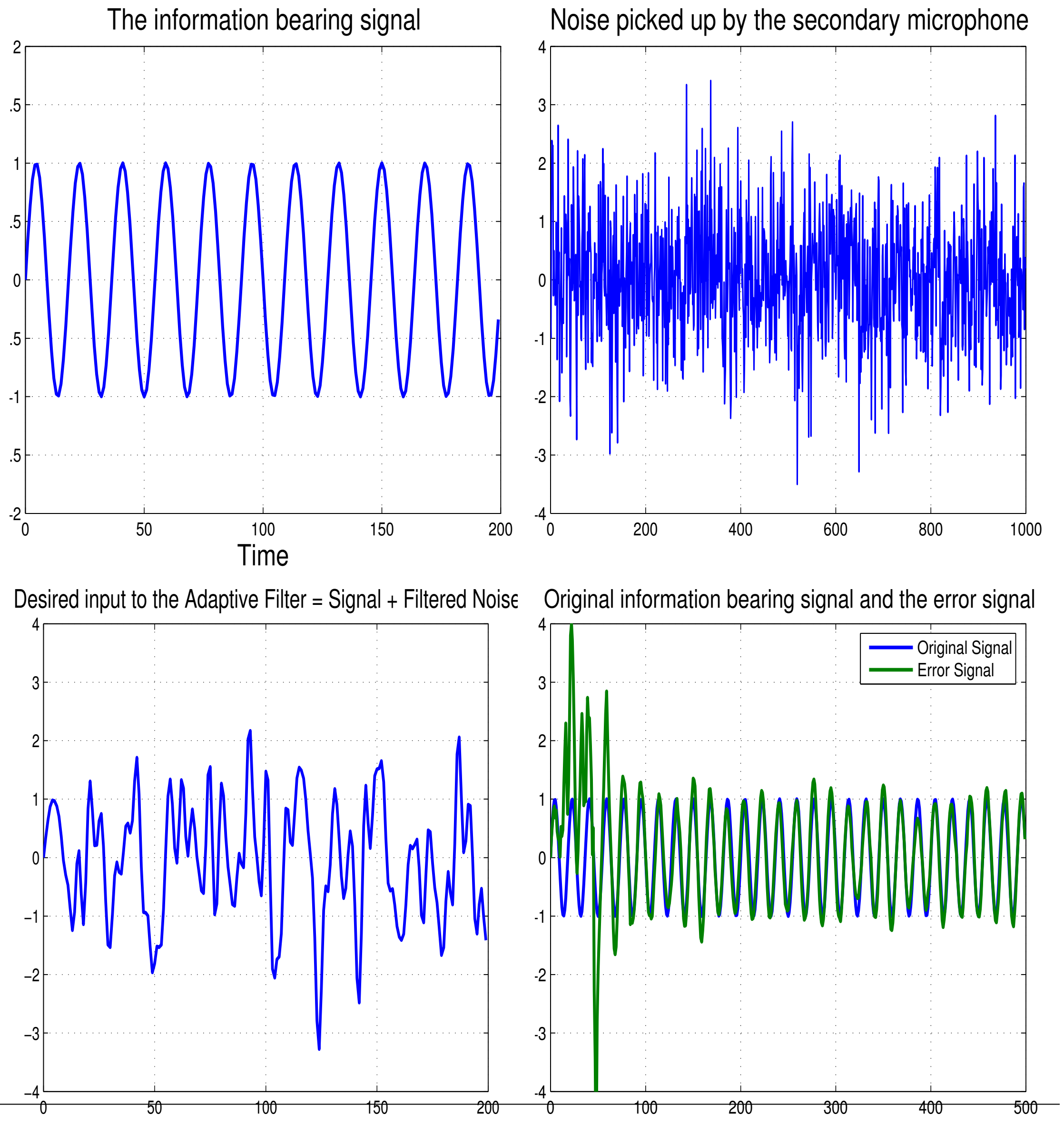
**Primary signal:**

$$\sin(n) + q(n)$$

**Reference signal:**

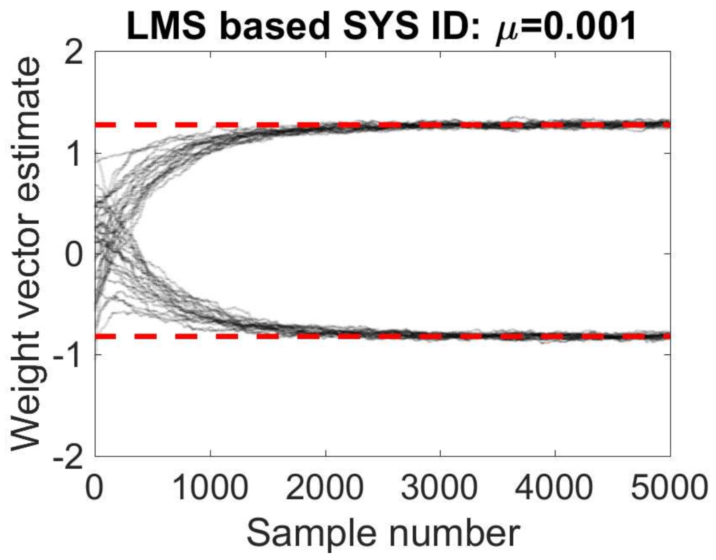
any noise correlated with the noise $q(n)$ in the primary signal

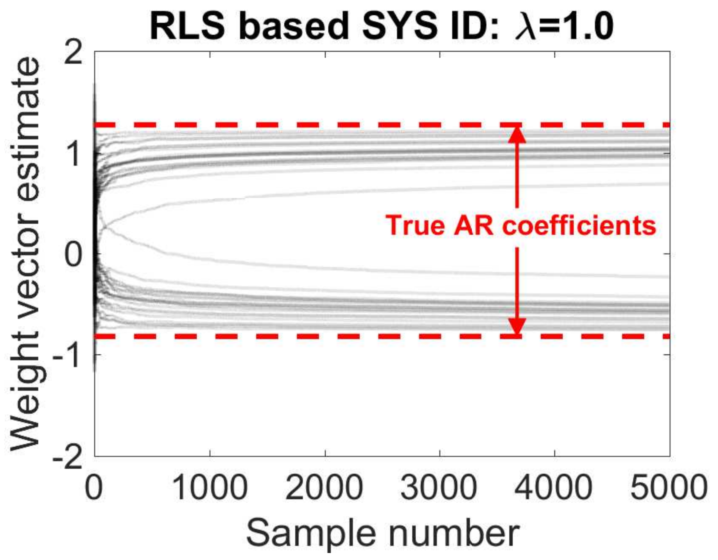


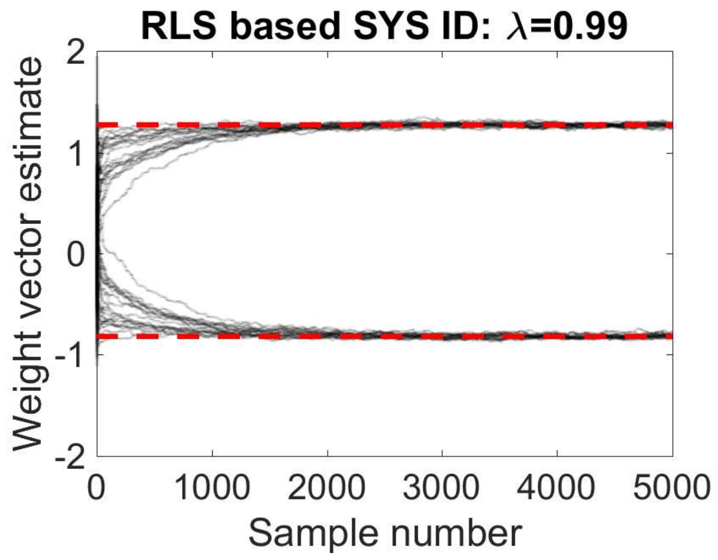


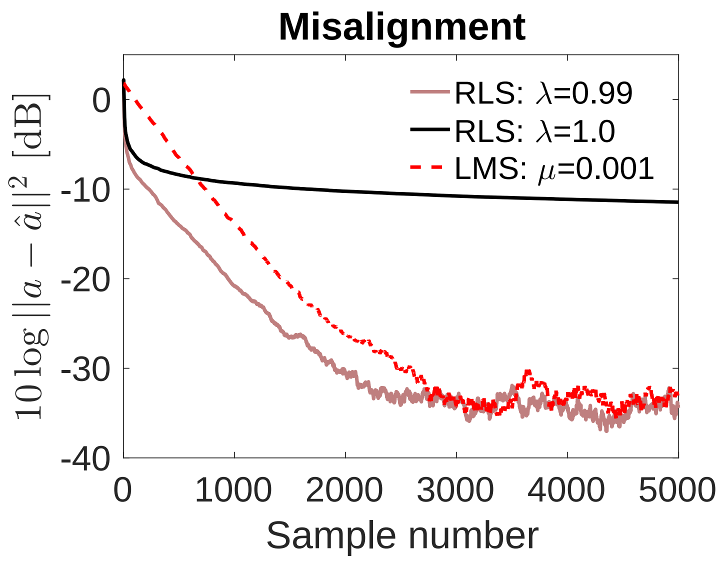
# Comparison between LMS and RLS

## Consider SYS ID of an AR(2) process with coefficients $\mathbf{a} = [1.2728, -0.82]^\top$, driven by $w \sim \mathcal{N}(0,1)$. Simulations performed over 10 independent realisations.









## Forgetting factor

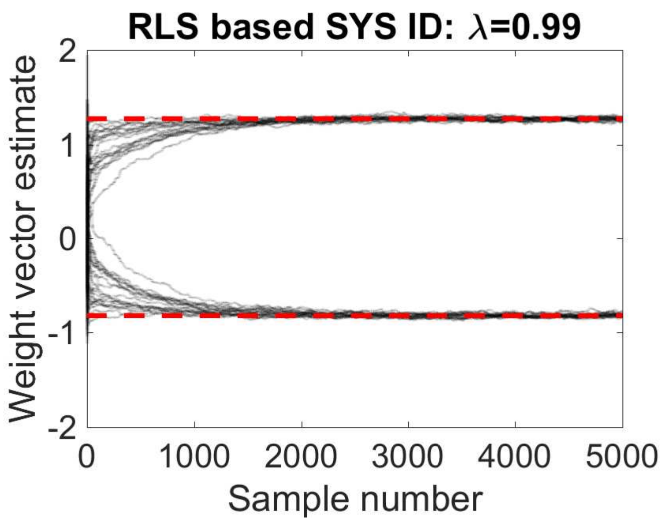
- The basic cost function for the RLS algorithm assumes a statistically stationary environment
- In the original cost function all the errors are weighted equally, which is wrong in the statistically nonstationary environment where distant past is not contributing to learning
- In order to deal with a nonstationary environment, modify the LS error criterion (recall the weighted least squares (WLS))

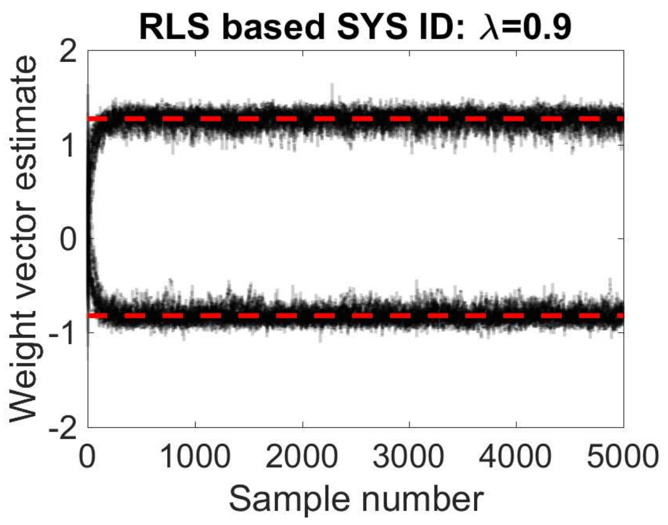
$$J(n) = \sum_{i=0}^{n} \lambda^{n-i} e^2(i) = [e_n, e_{n-1}, \ldots, e_0] \begin{bmatrix} \lambda^0 & 0 & \cdots \\ 0 & \lambda^1 & \cdots \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda^n \end{bmatrix} \begin{bmatrix} e_n \\ e_{n-1} \\ \vdots \\ e_0 \end{bmatrix} = \mathbf{e}^T \mathbf{W} \mathbf{e}$$

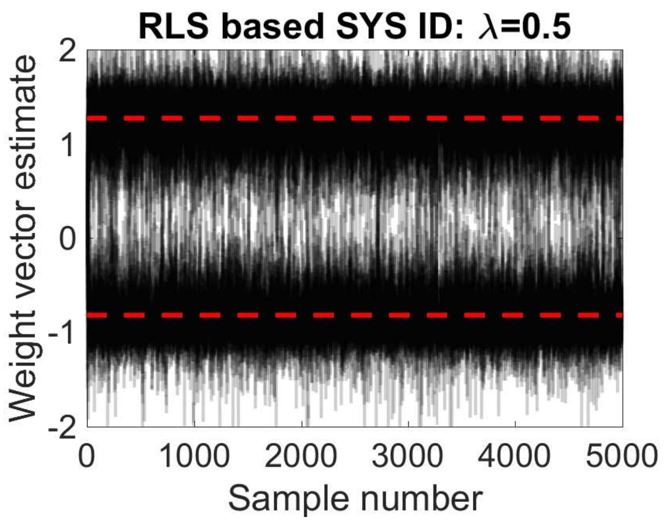
- In this way, the 'old' information is gradually forgotten
- Forgetting factor $\lambda \in (0, 1]$, but typically $> 0.95$
- The forgetting factor introduces an effective window length of $\frac{1}{1-\lambda}$

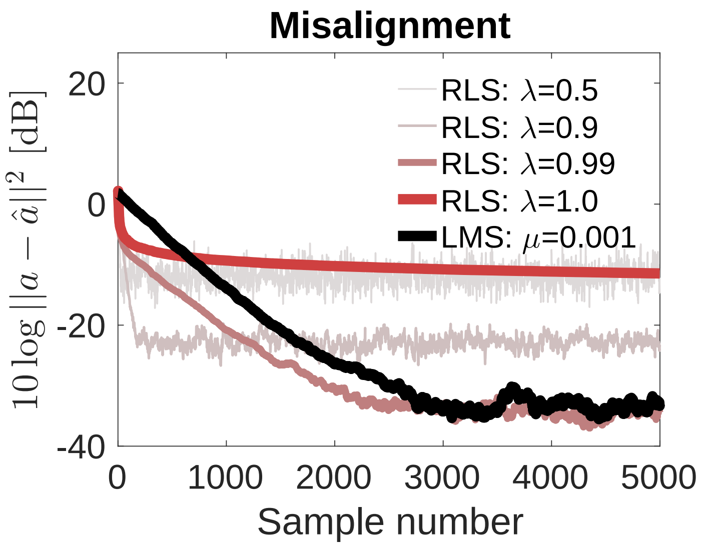
## RLS with forgetting factor $\lambda$

### with a careful choice of $\lambda$, RLS can outperform LMS









## A compact form of RLS and geometry of learning

From $\quad \mathbf{w}(n+1) = \mathbf{R}^{-1}(n+1)\mathbf{p}(n+1)$, we have

$$\mathbf{w}(n+1) = \left[\mathbf{R}^{-1}(n) - \frac{\mathbf{R}^{-1}(n)\mathbf{x}(n+1)\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)}{\mathbf{x}^T(n+1)\mathbf{R}^{-1}(n)\mathbf{x}(n+1)+1}\right]\left[\mathbf{p}(n) + d(n)\mathbf{x}(n)\right]$$

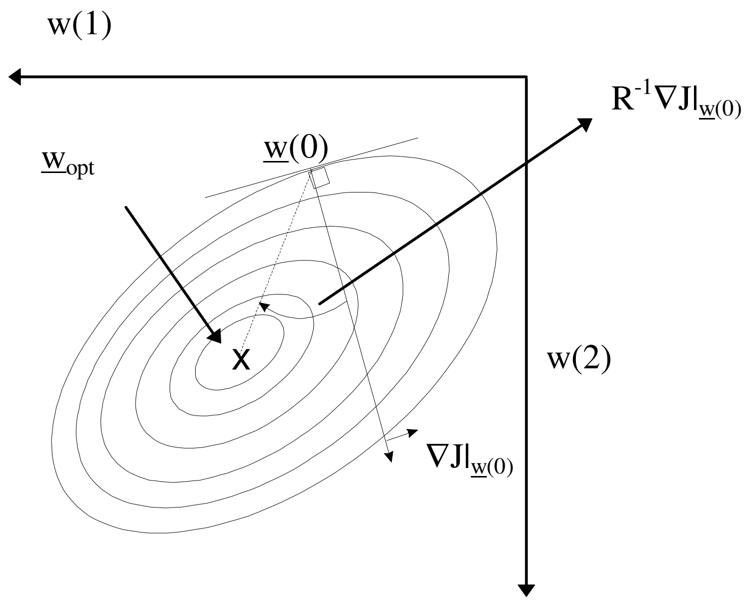
After some grouping of the terms above, we arrive at

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mathbf{R}^{-1}(n)}{1 + \mathbf{x}^T(n)\mathbf{R}^{-1}(n)\mathbf{x}(n)}\, e(n)\mathbf{x}(n)$$

○ the term $\mathbf{R}^{-1}$ “filters” the direction and length of the data vector $\mathbf{x}$

○ The term $1 + \mathbf{x}^T(n)\mathbf{R}^{-1}(n)\mathbf{x}(n)$ is a measure of input signal power, normalised by $\mathbf{R}^{-1}$.

○ This normalisation makes the power proportional to the data length $N$ and not to the actual signal level $\hookrightarrow$ it also decorrelates the data.



w(1)
$R^{-1}\nabla J|_{\underline{w}(0)}$
$\underline{w}_{opt}$
$\underline{w}(0)$
x
w(2)
$\nabla J|_{\underline{w}(0)}$

## Things to remember about the RLS

- Results are exactly the same as for the standard least squares $\looparrowright$ no approximation involved
- RLS does not perform matrix inversion at any stage $\looparrowright$ it calculates the matrix inverse recursively
- The role of $\mathbf{R}^{-1}$ in RLS is to rotate the direction of descent of the LMS algorithm towards the minimum of the cost function independent of the nature, or colouration, of the input
- The operation $\mathbf{R}^{-1}\mathbf{x}[k]$ is effectively a **pre-whitening operation**, compare with transform domain adaptive filtering
- Unlike gradient algorithms, there is no explicit notion of learning rate
- RLS guarantees convergence to the optimal weight vector
- Variants of RLS (sliding window, forgetting factor) deal with pragmatic issues (nonstationarity etc)
- The price to pay is increased computational complexity compared to the LMS
- The convergence of the RLS in a high SNR environment $(> 10dB)$ is of an order $\mathcal{O}(2p)$, whereas for LMS $\mathcal{O}(10p)$
- The misadjustment performance of RLS is essentially zero because it is deterministic and matches the data where $\lambda = 1$.

## From LMS to Kalman Filter

### Kalman filter $\Leftrightarrow$ LMS with an optimal learning "gain"

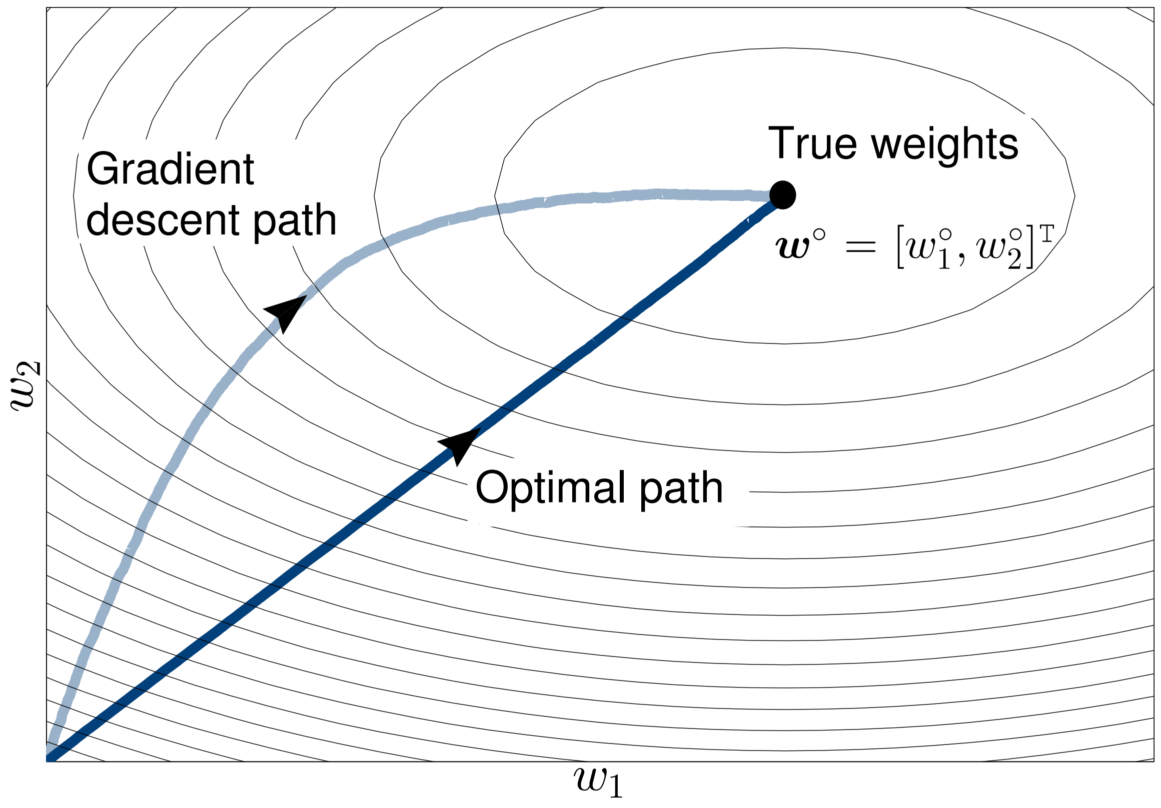
Recall the LMS algorithm: $\mathbf{w}_k = \mathbf{w}_{k-1} + \mu_k \mathbf{x}_k (d_k - \mathbf{w}_{k-1}^T \mathbf{x}_k)$ for the desired signal given by $d_k = \mathbf{x}_k^T \mathbf{w}^\circ + q_k$.

- Introduce more degrees of freedom by replacing the scalar step-size, $\mu_k$, with a positive definite learning gain matrix, $\mathbf{G}_k$, to give
$$\mathbf{w}_k = \mathbf{w}_{k-1} + \mathbf{G}_k \mathbf{x}_k (d_k - \mathbf{w}_{k-1}^T \mathbf{x}_k).$$
- Now, we are able to control both the **magnitude and direction** of the gradient descent adaptation.
- The matrix $\mathbf{G}_k$ is found by minimising the mean square deviation (MSD) in the form:
$$J_k = E\{\|\mathbf{w}^\circ - \mathbf{w}_k\|^2\}$$
$$\partial J_k / \partial \mathbf{G}_k = \mathbf{0} \implies \mathbf{G}_k = \frac{\mathbf{P}_{k-1}}{\mathbf{x}_k^T \mathbf{P}_{k-1} \mathbf{x}_k + \sigma_q^2}$$
with $\mathbf{P}_k = \mathbf{P}_{k-1} - \mathbf{G}_k \mathbf{x}_k \mathbf{x}_k^T \mathbf{P}_{k-1}$ and $\sigma_q^2 = E\{q_k^2\}$.
- The optimal gain vector $\mathbf{g}_k \triangleq \mathbf{G}_k \mathbf{x}_k$ is known as the **Kalman gain.**

## Relationship between the Kalman Filter and the LMS

### Optimal learning gain for stochastic gradient algorithms

Mean trajectories of an ensemble of noisy single-realisation gradient descent paths for correlated data.

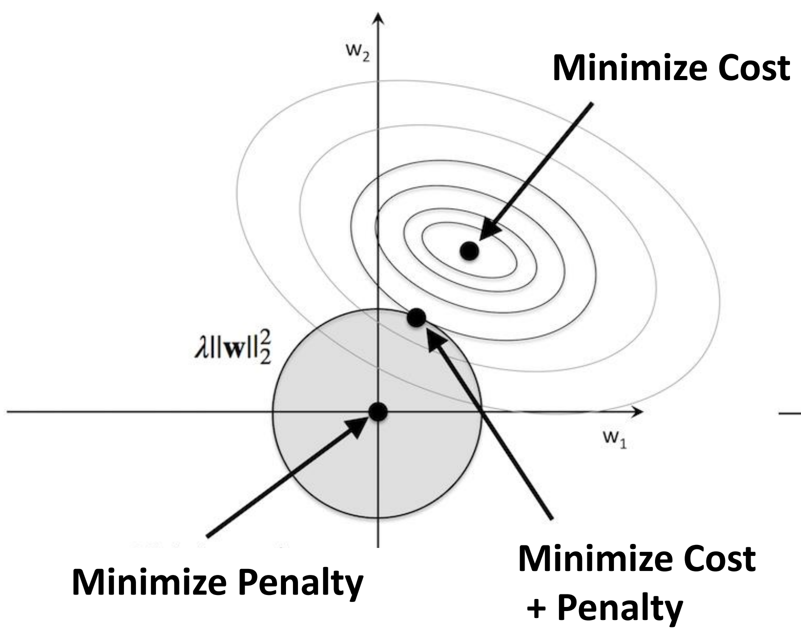


The LMS path is locally optimal but globally slower converging than the optimal path (see DM *at al.*, IEEE Sig. Proc. Magazine, March 2015).

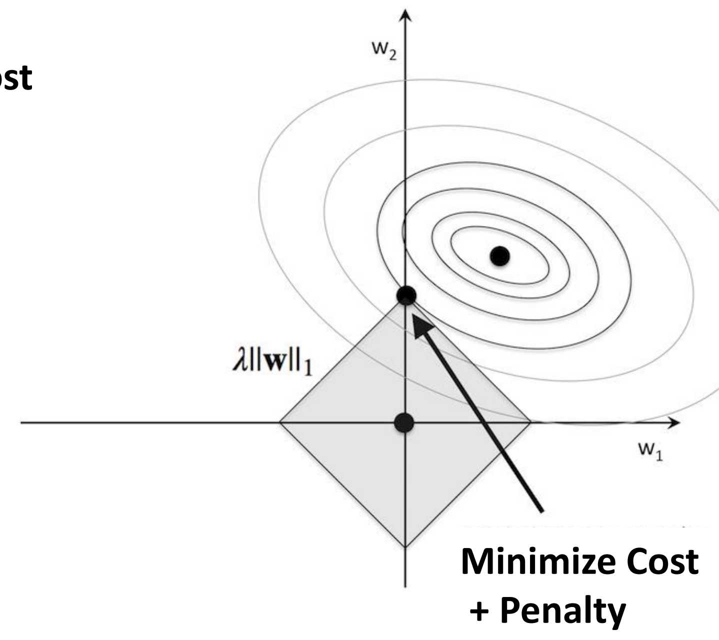
## Exploiting regularisation and sparsity

### Regularised LMS cost function with $\ell_2$-norm (Tikhonov) or $\ell_1$-norm (Lasso)

$$\mathcal{J}_k(\mathbf{w}) = \underbrace{(d_k - \mathbf{w}^T\mathbf{x}_k)^2}_{\text{Cost}} + \underbrace{\lambda\|\mathbf{w}\|_\ell}_{\text{Penalty}}$$ with Regularisation Parameter: $\lambda$.



Tikhonov: $\ell = 2$



Lasso: $\ell = 1$

(Figure credit: Mlxtend, S. Raschka, 2016)

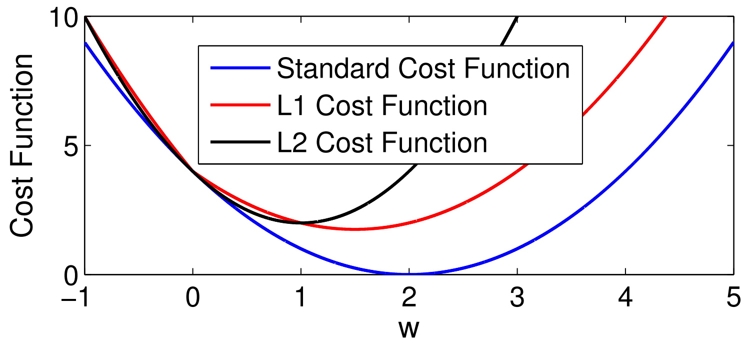
**Sparsity promoting**

For $\lambda \neq 0$ the **minima of the regularised cost functions do not correspond** to the squared error cost.

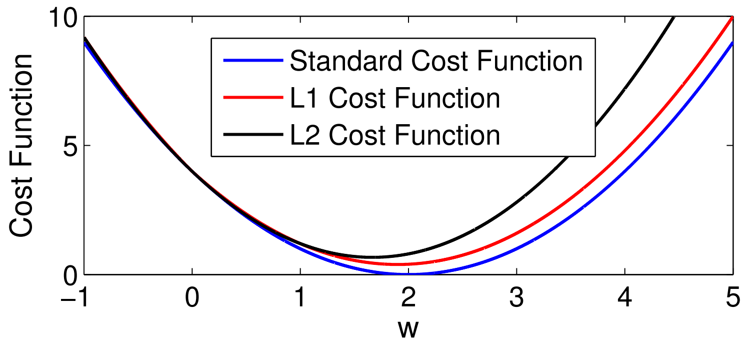
## Regularised LMS aims to avoid overfitting by a penalty for complexity

### The effect of the regularisation parameter

Consider a simple case of: $d_k = w^\circ x_k + q_k$ with $w_\circ = 2$.



$\lambda = 1$



$\lambda = 0.2$

Comparison between the $\ell_1$- and $\ell_2$-norm regularised cost functions

- For $\lambda \neq 0$ the minima of the regularised cost functions do not correspond to the optimum weight $w^\circ = 2$
- The regularisation parameter $\lambda$ controls the **trade-off** between minimising the cost (squared error) and penalty.
- **Smaller values of $\lambda$** correspond to favours **minimising the cost** (squared error) while **large $\lambda$ values** strongly **penalises the norm of $\mathbf{w}$**.

## A framework to involve sparsity in optimisation

The optimisation criterion becomes:

$$\begin{aligned}
\hat{\mathbf{w}} &= arg\min_{\mathbf{w}} \Big\{ E\big[\, \| \, y(n) - \mathbf{w}^T(n)\mathbf{x}(n) \, \|_2^2 \,\big] + \lambda \, \| \, \mathbf{w}(n) \, \|_1 \Big\} \\
&\equiv \min_{\mathbf{w}} \Big\{ E\big[\, \| \, y(n) - \mathbf{w}^T(n)\mathbf{x}(n) \, \|_2^2 \,\big] \Big\} \quad \text{s.t.} \quad \| \, \mathbf{w}(n) \, \|_1 \leq \rho \\
&\equiv \min_{\mathbf{w}} \Big\{ \, \| \, \mathbf{w}(n) \, \|_1 \Big\} \quad \text{s.t.} \quad E\big[\, \| \, y(n) - \mathbf{w}^T(n)\mathbf{x}(n) \, \|_2^2 \,\big] \leq \varepsilon
\end{aligned}$$

For LMS-type stochastic gradient descent, for a filter with length $L$ we then have

$$\begin{aligned}
\hat{\mathbf{w}} &= arg\min_{\mathbf{w}} \Big\{ \frac{1}{2} e^2(n) + \lambda \, \| \, \mathbf{w}(n) \, \|_1 \Big\} = arg\min_{\mathbf{w}} \Big\{ \frac{1}{2} e^2(n) + \lambda \sum_{i=1}^{L} |w_i(n)| \Big\} \\
\mathbf{w}(n+1) &= \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) - \mu\lambda sign(\mathbf{w}(n))
\end{aligned}$$

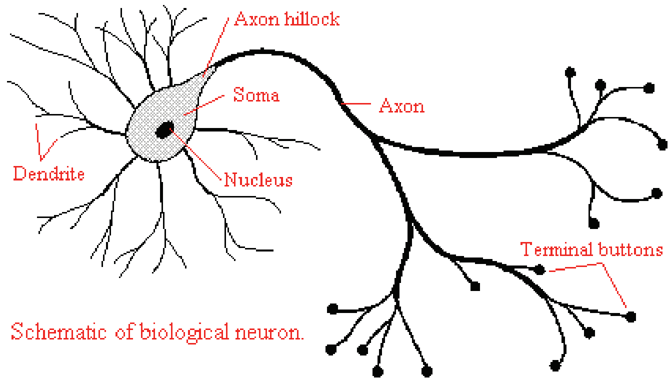
Other constraints can also be included (e.g. smoothness)

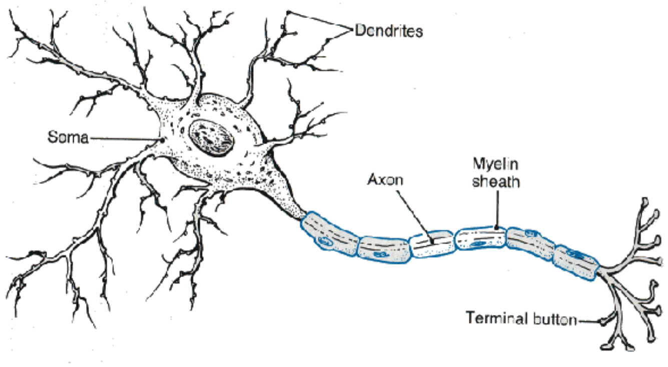
## The need for nonlinear structures

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

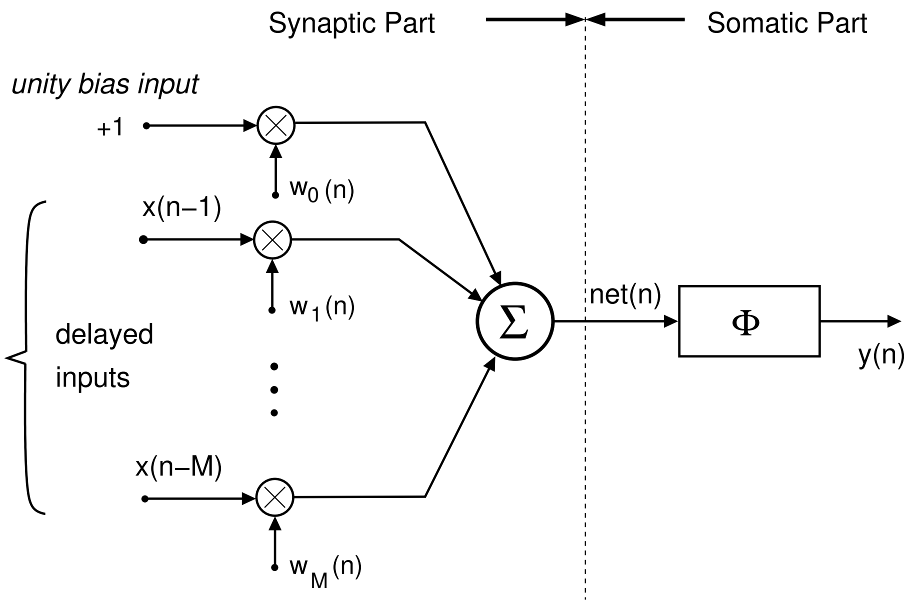
- when trying to identify dynamical signals/systems observed through a saturation type sensor nonlinearity, the use of linear models will be limited
- when separating signals with overlapping spectral components
- systems which are naturally nonlinear or signals that are non-Gaussian, such as limit cycles, 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





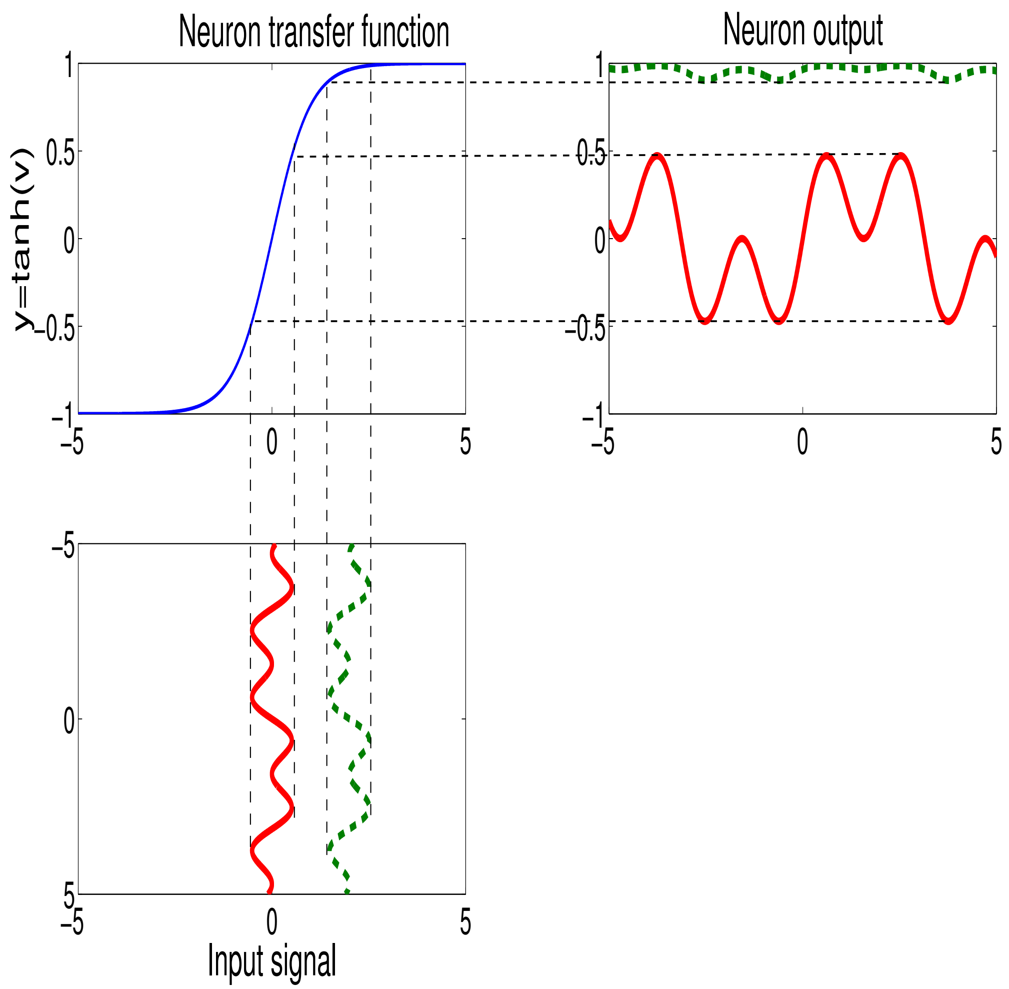
Biological neuron



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**

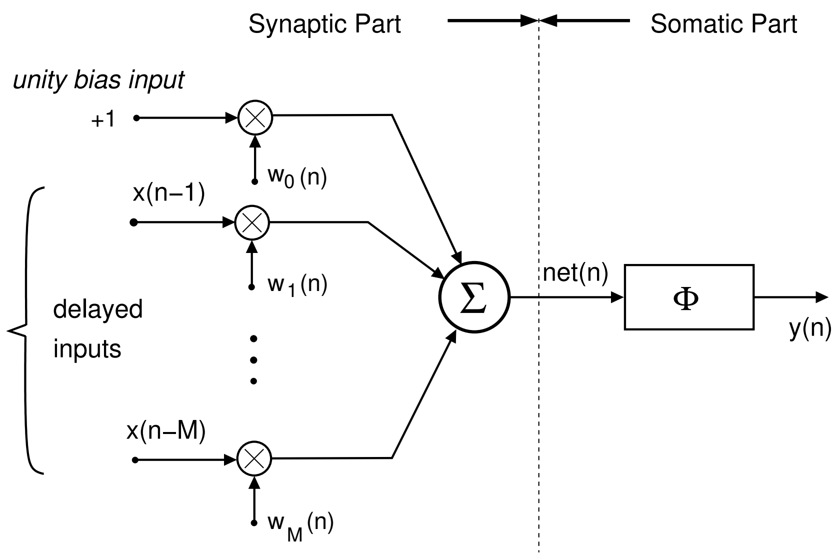
- observe the different behaviour depending on the operating point
- the output behaviour varies from amplifying and slightly distorting the input signal to attenuating and considerably distorting
- From the viewpoint of system theory, neural networks represent nonlinear maps, mapping one metric space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

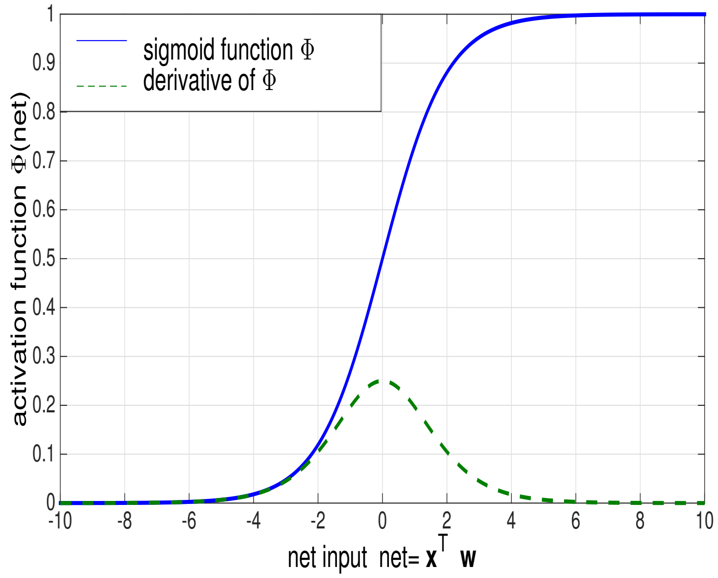
## A simple nonlinear structure, referred to as the perceptron, or dynamical perceptron

- Consider a simple nonlinear FIR filter, with a sigmoid nonlinearity

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

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

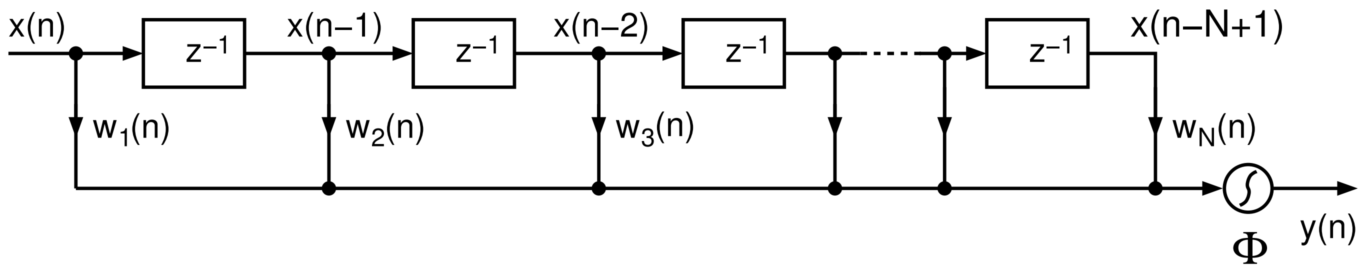




Model of artificial neuron (dynamical perceptron, nonlinear FIR filter)

## 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

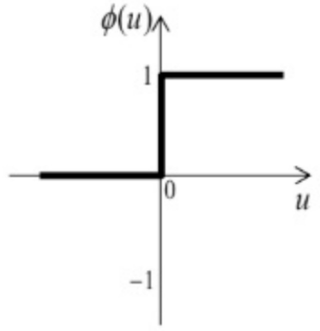
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\Phi'\left(\mathbf{x}^T(n)\mathbf{w}(n)\right)e(n)\mathbf{x}(n) = \mathbf{w}(n) + \mu\Phi'(n)e(n)\mathbf{x}(n)$$

∘ This filter is BIBO (bounded input bounded output) stable, as the output range is limited due to the saturation type of nonlinearity $\Phi$.

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

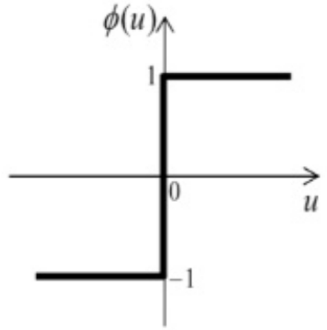
## Nonlinear activation functions in NNs

step function



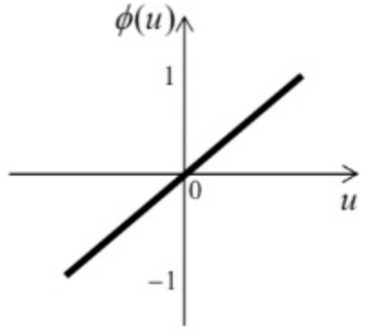
$$\phi_{step}(u) = \begin{cases} 1 & \text{if} \quad u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

sign function

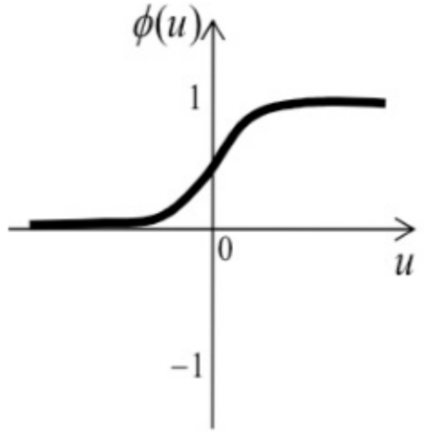


$$\phi_{sign}(u) = \begin{cases} 1 & \text{if} \quad u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

identity function

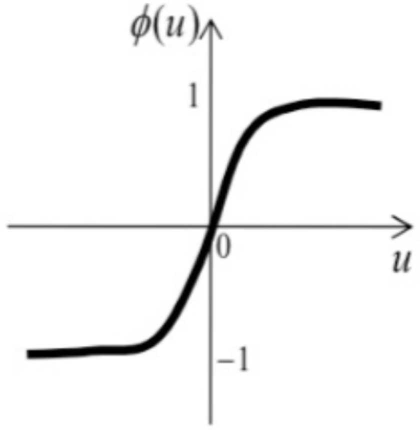


$$\phi_{id}(u) = u$$



$$\phi_{sig}(u) = \frac{1}{1+e^{-u}}$$

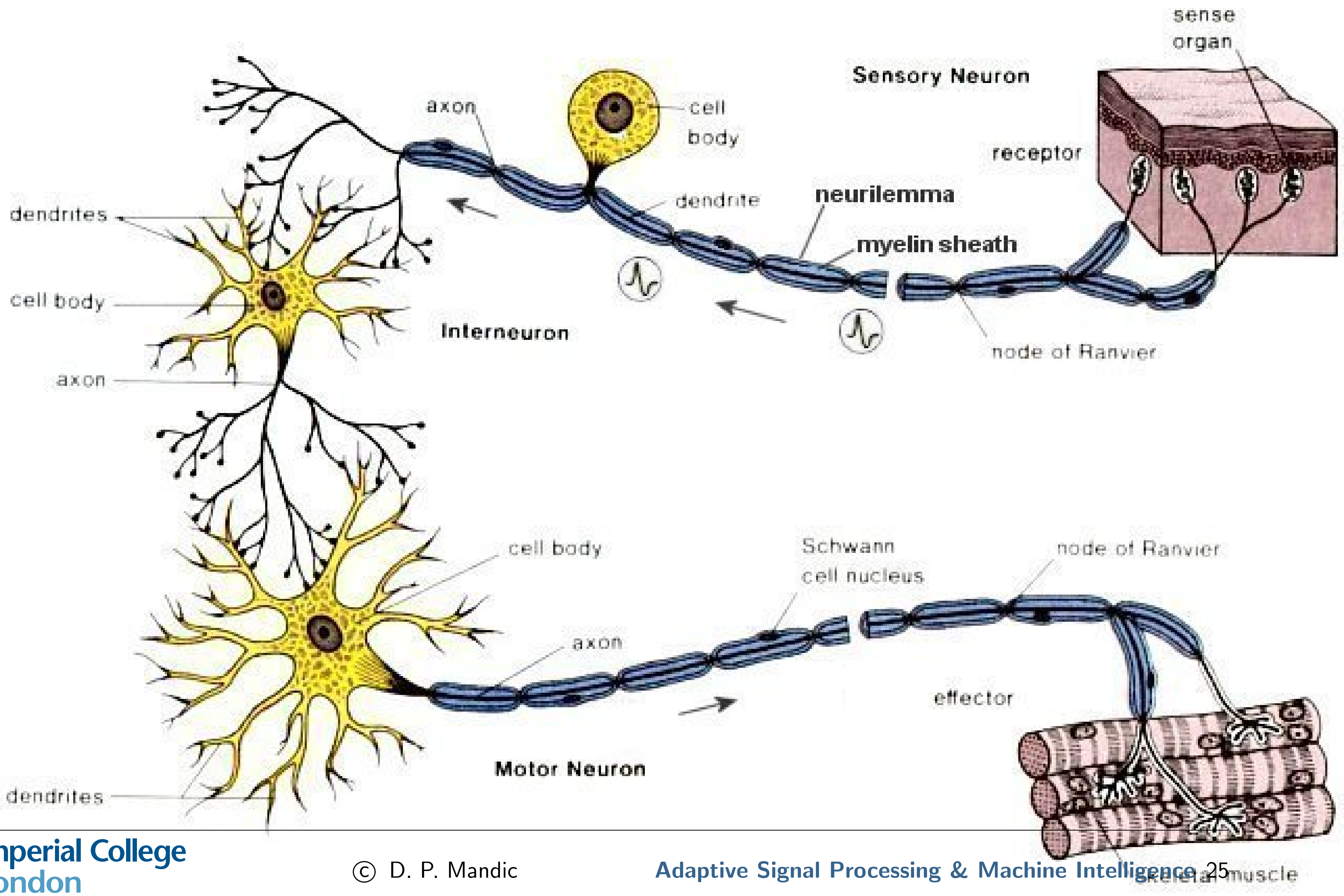
sigmoid function



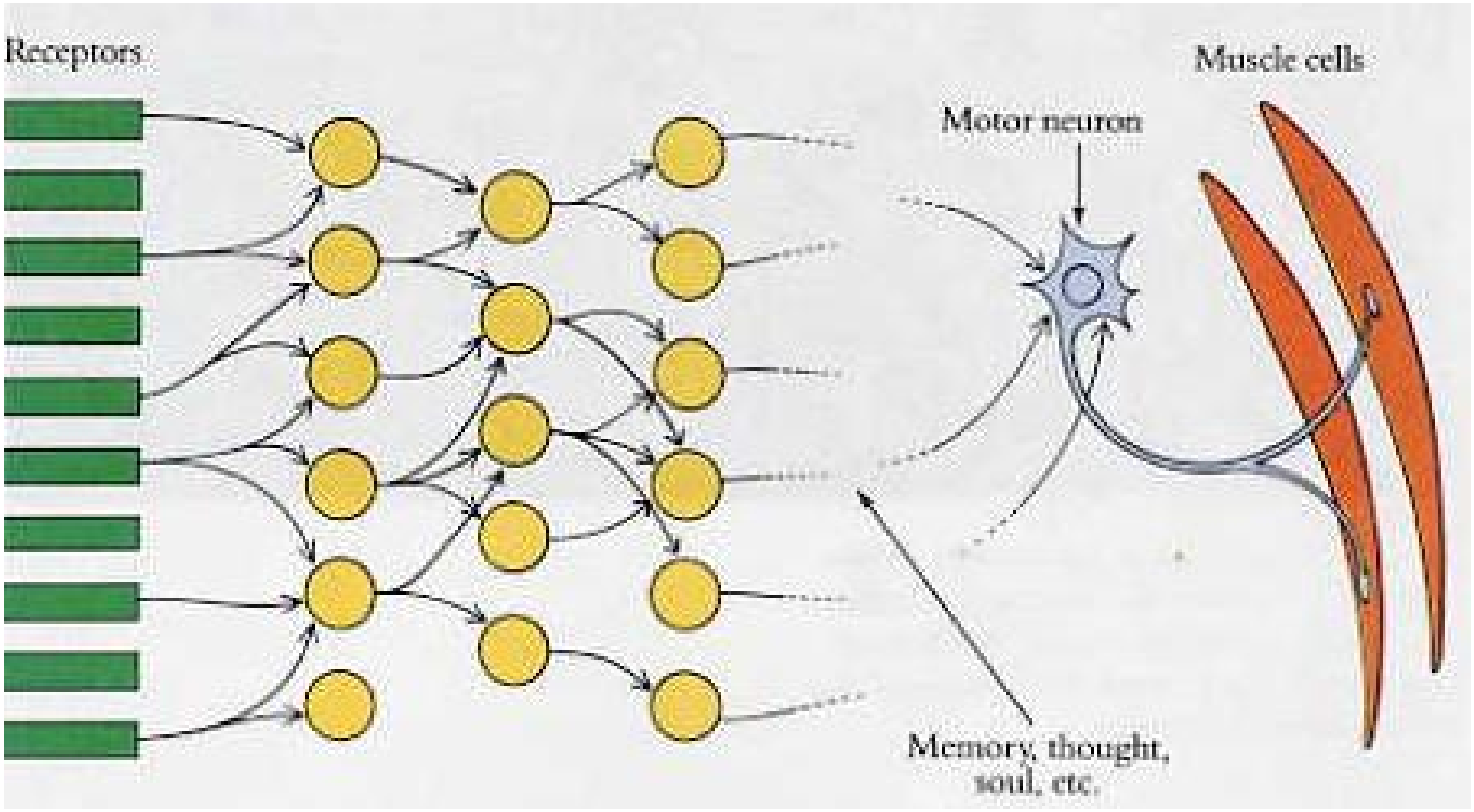
$$\phi_h = \frac{e^u - 1}{e^u + 1}$$

hyper tangent function

# From a neuron to a neural network
## Mapping of sensory inputs



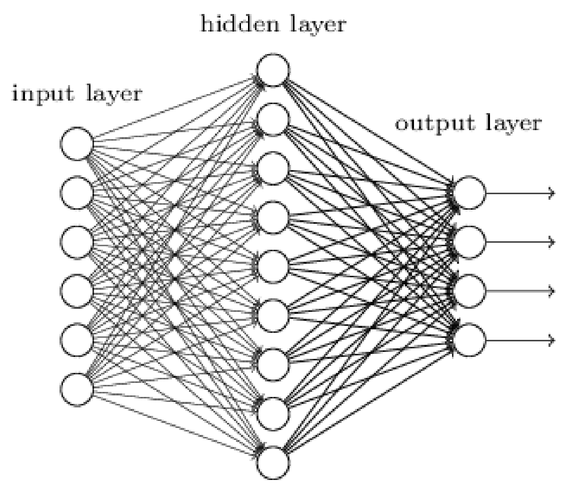
## A biological neural network: Perception $\looparrowright$ action



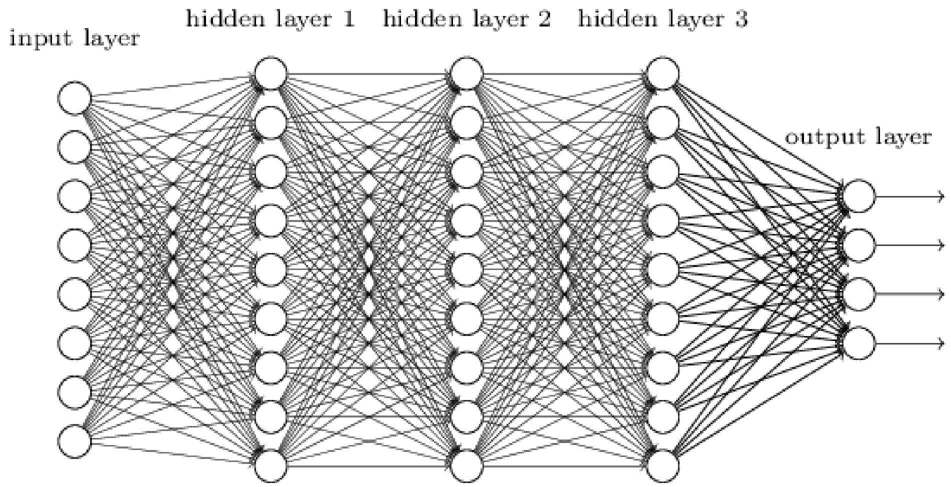
☞ There are multiple layers of neurons which represent, encode, and transmit information at a different scale and in a highly parallel manner

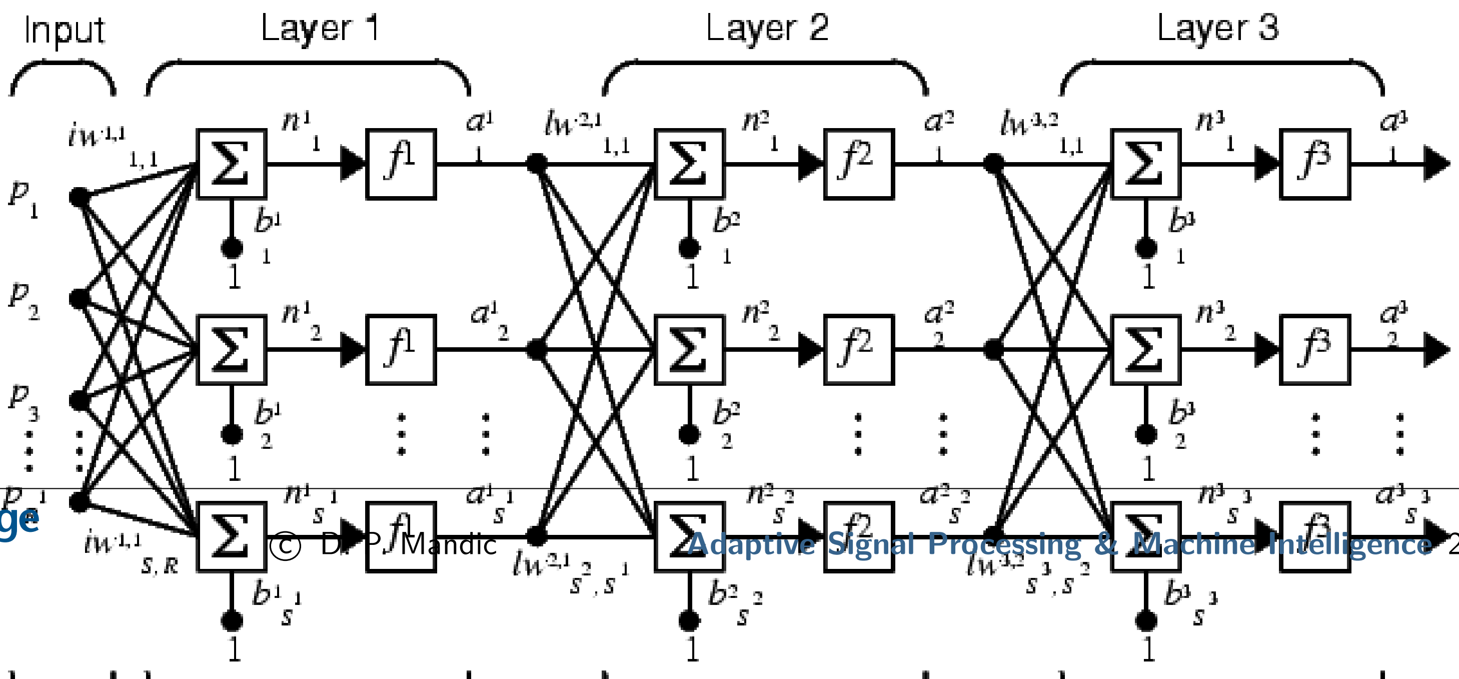
## Feedforward neural networks (NNs)

"Non-deep" feedforward neural network



Deep neural network





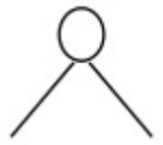
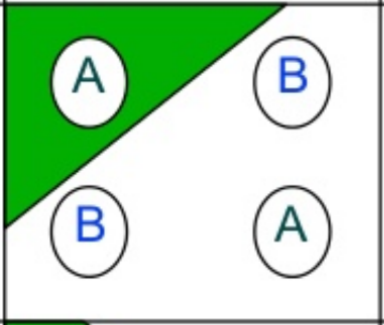
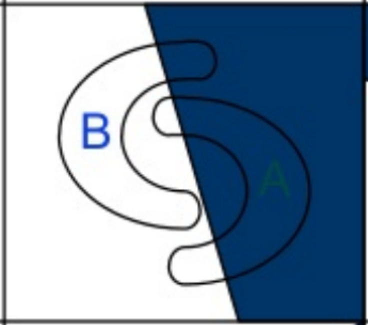
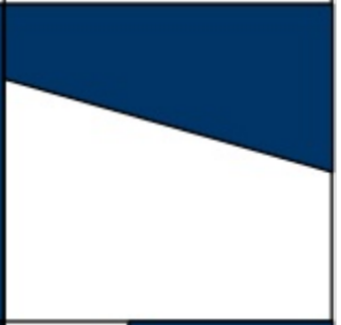
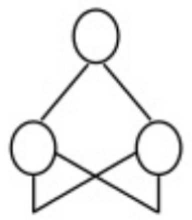
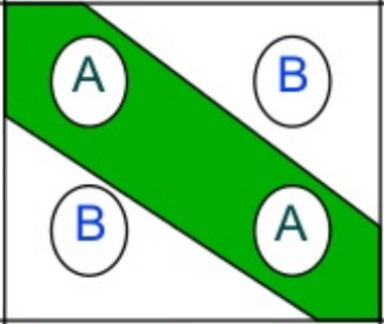
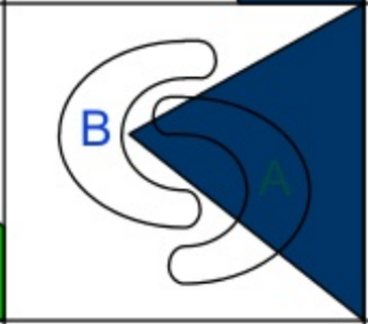
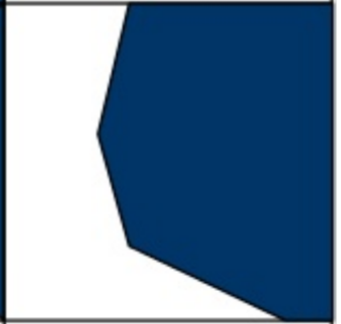
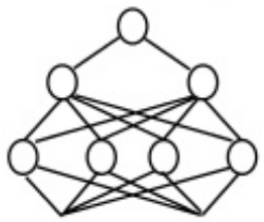
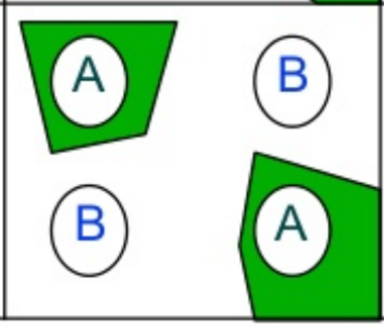
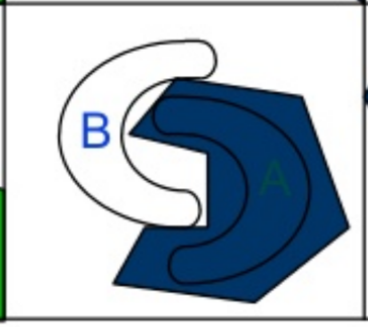
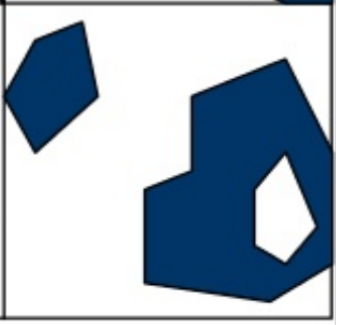
## Universal function approximation property of NNs

- A list of 23 problems in mathematics put forth by David Hilbert at the Paris conference of the International Congress of Mathematicians in 1900 (8 August in the Sorbonne)
- Several of them turned out to be very influential for 20th century mathematics
- The 13th problem is a "universal function approximation" problem (approximate 7th order polynomial with a function of 2 variables)
- In 1956 Kolmogorov has shown that any function of several variables can be constructed with a finite number of three-variable functions (extended by V. Arnold for only two-variable functions)

**The Kolmogorov solution is similar to the architecture known as Radial Basis Function (RBF) Neural Network (a dual of MLP).**

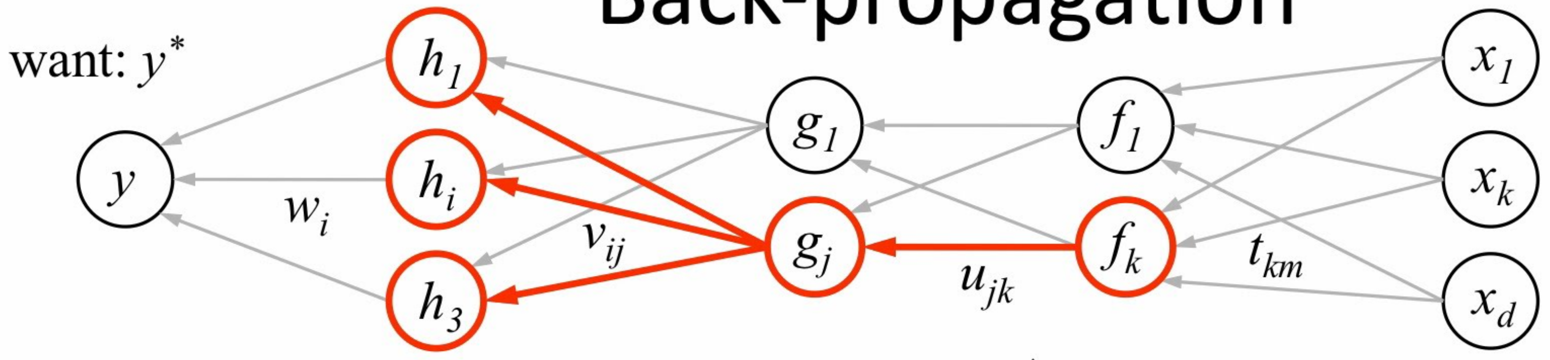
**The universal function approximation capability of MLPs: Cybenko, Hornik; RNNs: various authors**

## Multilayer perceptron with one hidden layer enables nonlinear separation of classes

| *Structure* | *Types of Decision Regions* | *Exclusive-OR Problem* | *Classes with Meshed regions* | *Most General Region Shapes* |
|---|---|---|---|---|
| *Single-Layer*  | *Half Plane Bounded By Hyperplane* |  |  |  |
| *Two-Layer*  | *Convex Open Or Closed Regions* |  |  |  |
| *Three-Layer*  | Arbitrary (Complexity Limited by No. of Nodes) |  |  |  |

## Training feedforward NNs $\looparrowright$ the backpropagation algorithm

Back-propagation



1. receive new observation $\boldsymbol{x} = [x_1 \dots x_d]$ and target $y^*$
2. **feed forward:** for each unit $g_j$ in each layer $1 \dots L$
   compute $g_j$ based on units $f_k$ from previous layer: $g_j = \sigma\left(u_{j0} + \sum_k u_{jk} f_k\right)$
3. get prediction $y$ and error $(y - y^*)$
4. **back-propagate error:** for each unit $g_j$ in each layer $L \dots 1$

(a) compute error on $g_j$

$$\underbrace{\frac{\partial E}{\partial g_j}}_{\text{should } g_j \text{ be higher or lower?}} = \sum_i \underbrace{\sigma'(h_i) v_{ij}}_{\text{how } h_i \text{ will change as } g_j \text{ changes}} \underbrace{\frac{\partial E}{\partial h_i}}_{\text{was } h_i \text{ too high or too low?}}$$

(b) for each $u_{jk}$ that affects $g_j$

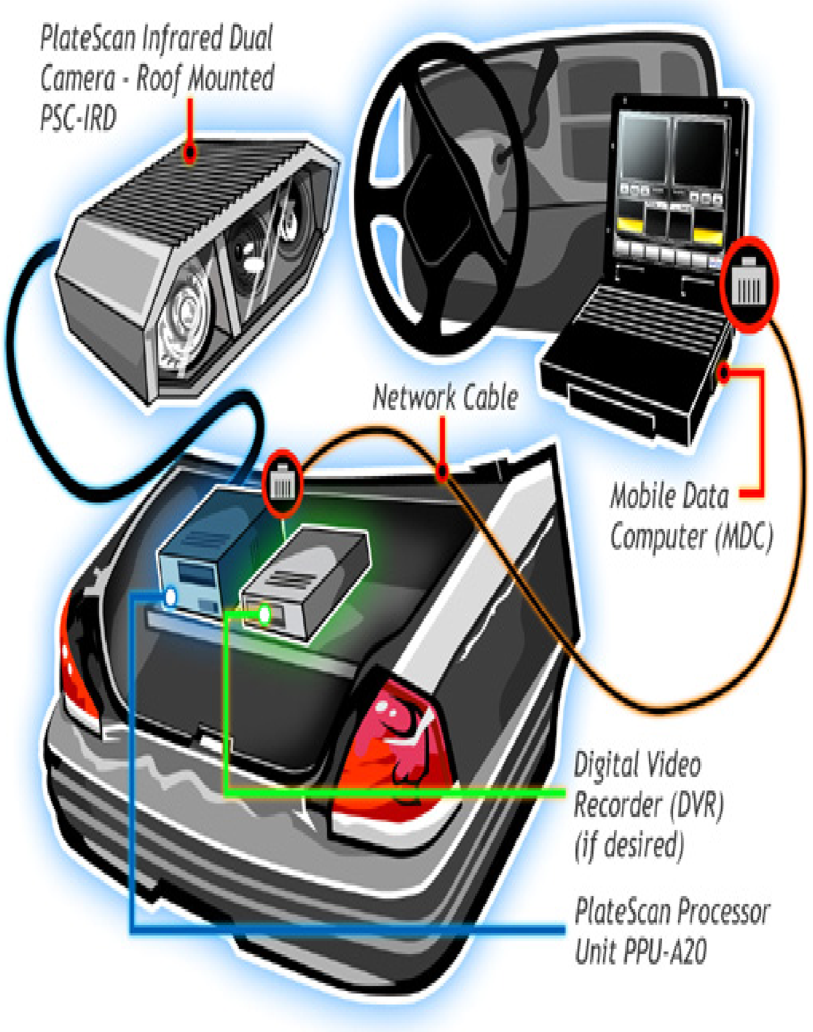
(i) compute error on $u_{jk}$

$$\frac{\partial E}{\partial u_{jk}} = \underbrace{\frac{\partial E}{\partial g_j}}_{\text{do we want } g_j \text{ to be higher/lower}} \underbrace{\sigma'(g_j) f_k}_{\text{how } g_j \text{ will change if } u_{jk} \text{ is higher/lower}}$$

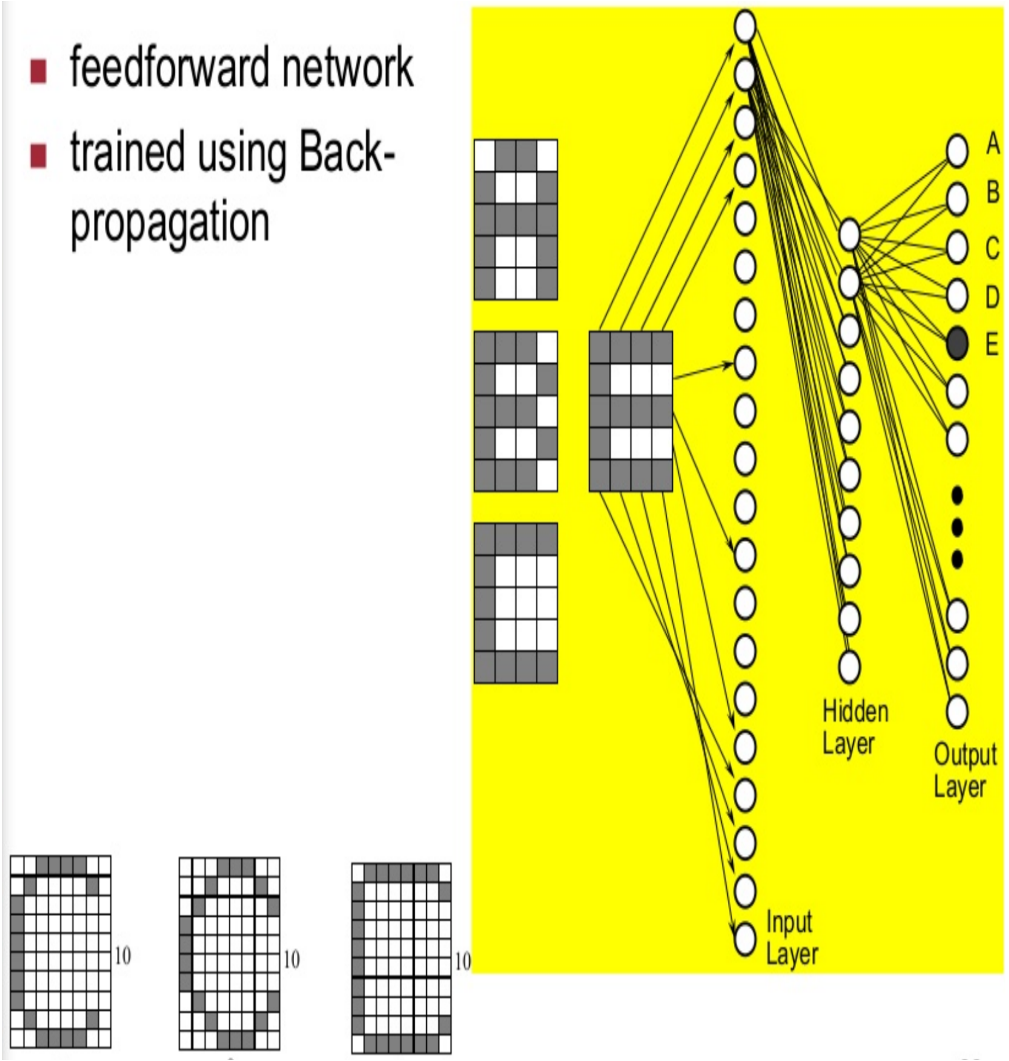
(ii) update the weight

$$u_{jk} \leftarrow u_{jk} - \eta \frac{\partial E}{\partial u_{jk}}$$

## Application: Optical character recognition

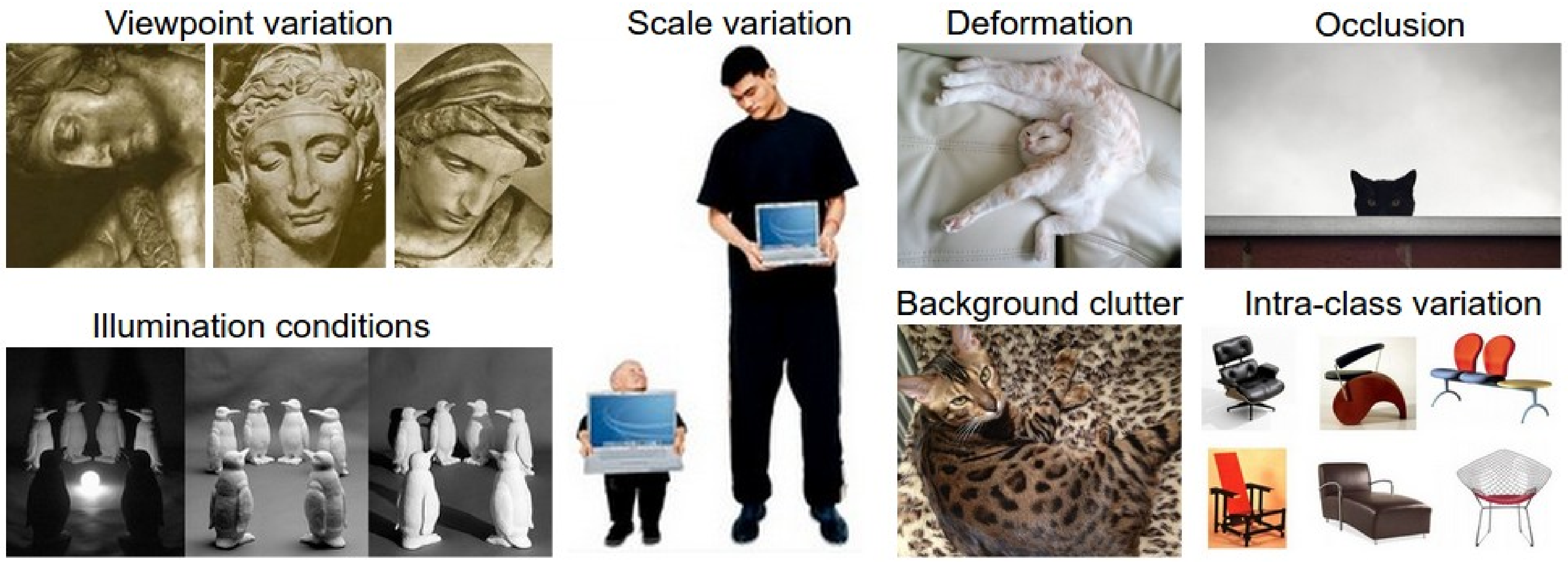

PlateScan Infrared Dual Camera - Roof Mounted PSC-IRD
Network Cable
Mobile Data Computer (MDC)
Digital Video Recorder (DVR) (if desired)
PlateScan Processor Unit PPU-A20

- feedforward network
- trained using Back-propagation

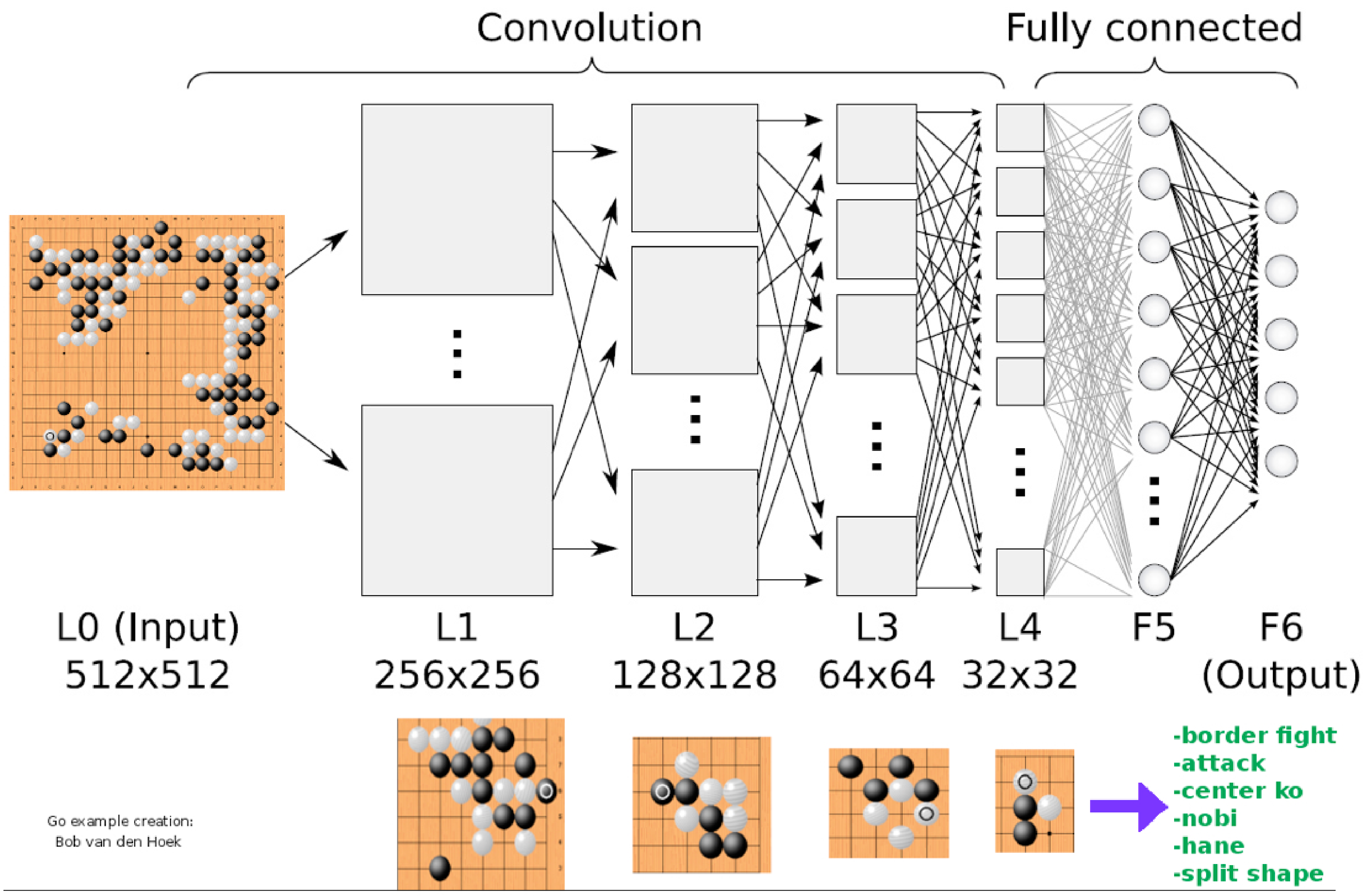


## Applications on massive data volumes

### Top: Challenges Bottom: Classes

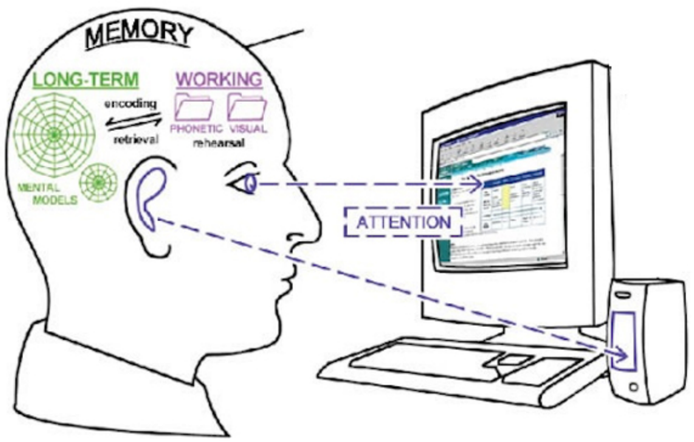


## These huge volumes can be processed using deep NNs, for example the AlphaGo (Nature, 2016)



# Motivation for recurrent neural networks (RNN)

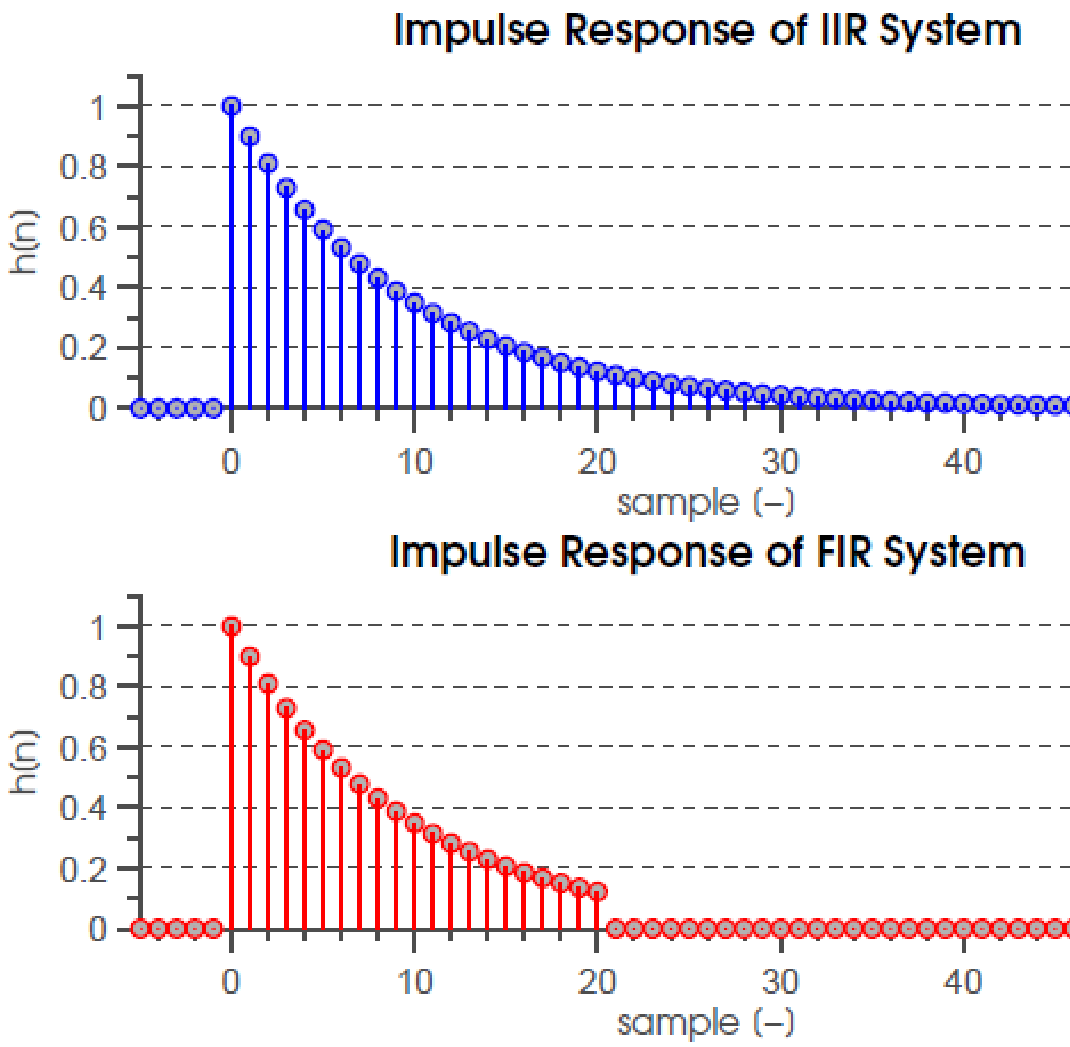
## Even a small-scalle IIR system can produce an infinite impulse response



○ Biological systems typically operate in a feedback loop

○ This also involves both short-term and long-term memory

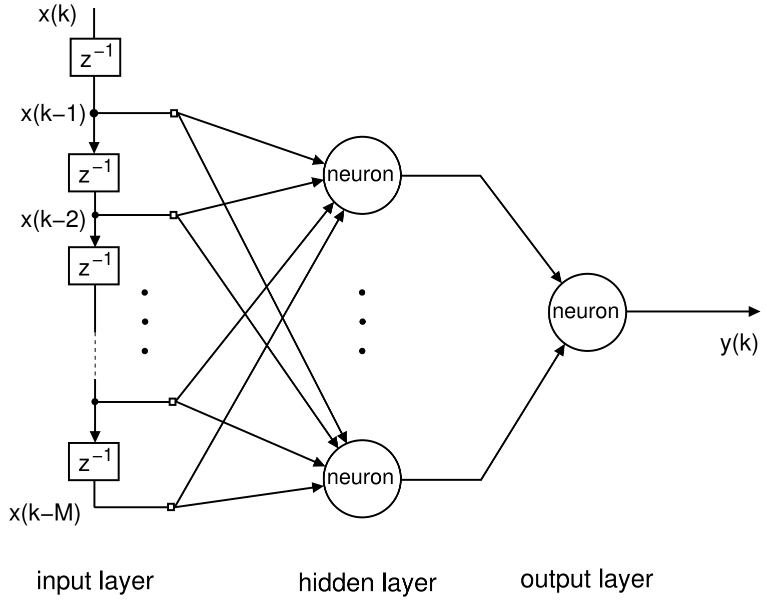
○ Speed of information transfer in the brain is slow [0.6 m/s for pain, 20 m/s for thinking, to 100 m/s for passive attention], but many processing units (neurons) operate in parallel



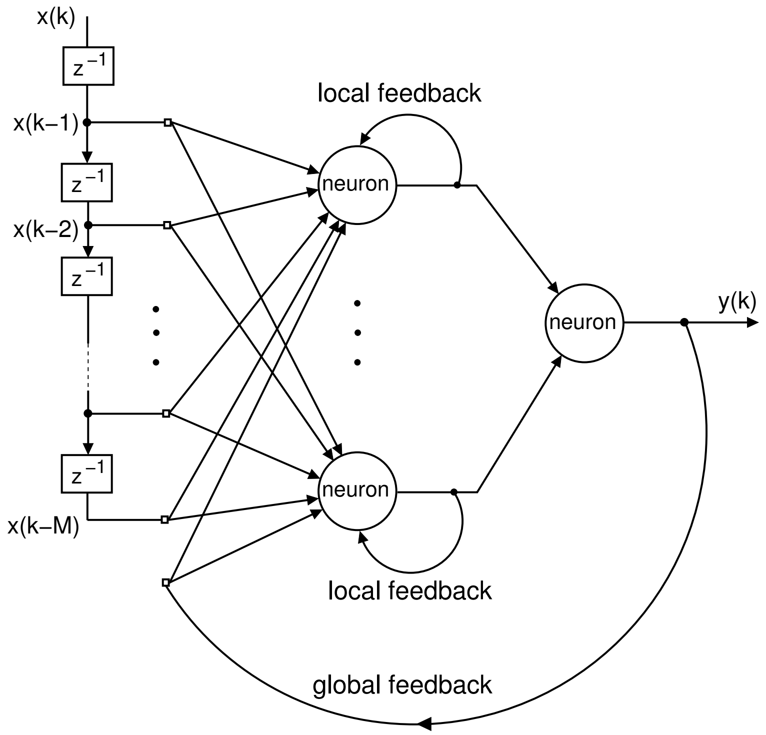
Imp. resp. for $x_n = 0.9x_{n-1} + q_n$

## Feedforward and feedback networks

Single artificial neuron models can be combined to form **neural networks.**



Feedforward network with hidden layer



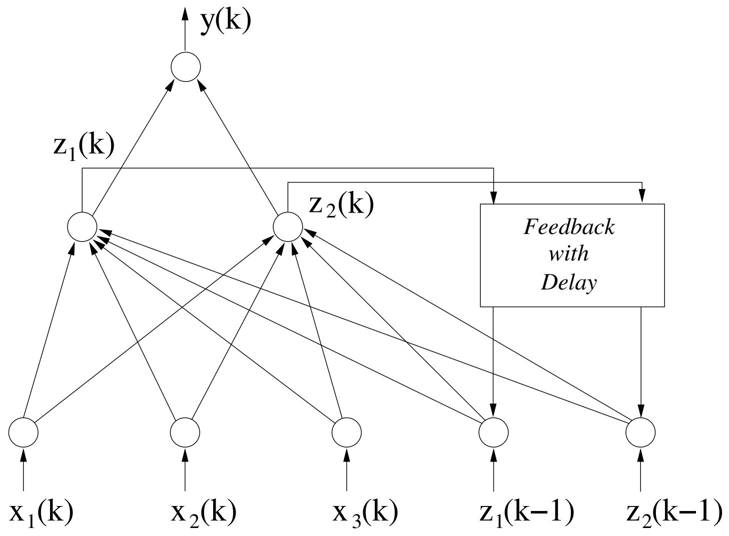
recurrent neural network

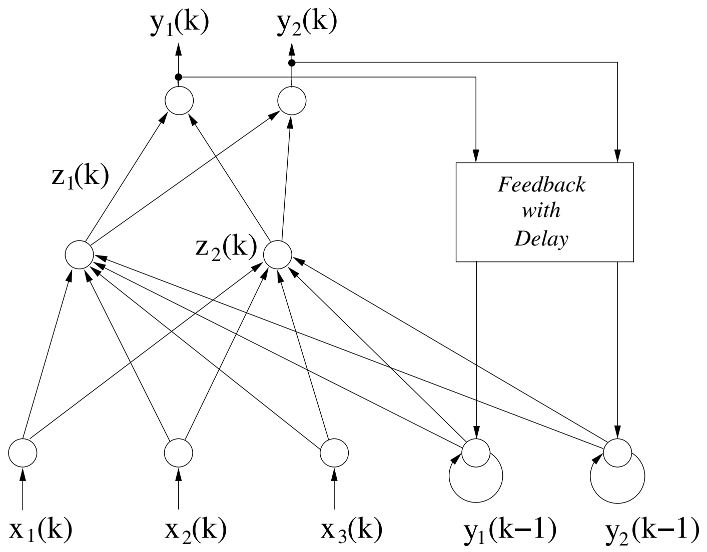
Direct gradient training of is difficult and we resort to “error backpropagation”

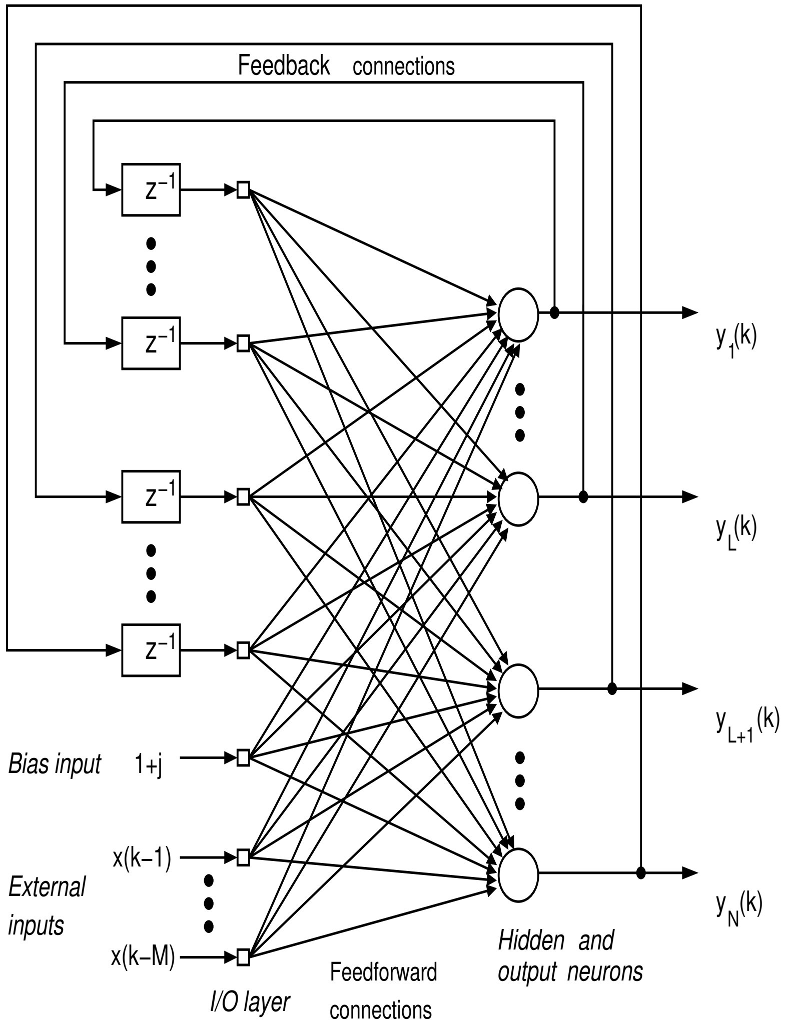
Recurrent ’feedback’ connections can be local or global.

## Some standard RNN architectures

**Top: Elman RNN. Bottom: Jordan RNN**



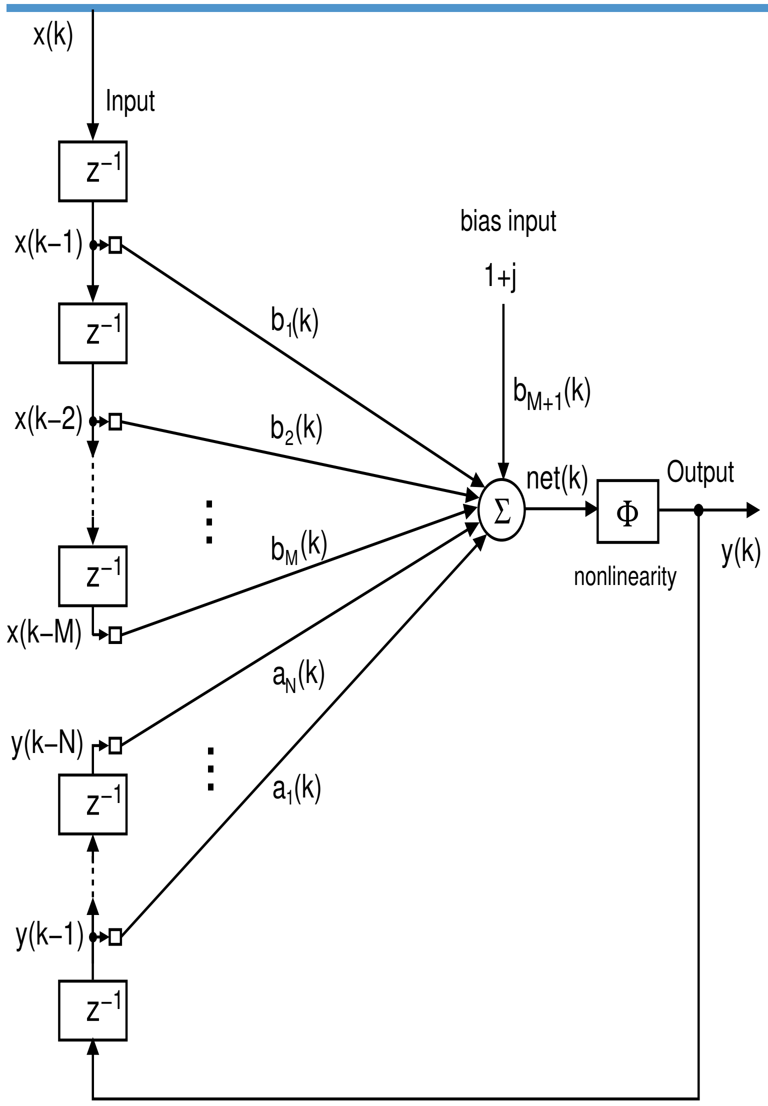




**Multilayer RNN**

## Recurrent perceptron

**this is a complex-valued variant, for the real value one the bias weight $b_{M+1} = 1$**



Denote by:

$$\mathbf{w}_a(k) = [a_1, \ldots, a_N]^T$$
$$\mathbf{w}_b(k) = [b_1, b_2, \ldots, b_M, b_{M+1}]^T$$
$$\mathbf{w}(k) = [\mathbf{w}_b^T(k), \mathbf{w}_a^T(k)]$$
$$\mathbf{u}(k) = [\mathbf{x}(k), \mathbf{y}(k)]$$

Then,

$$y(k) = \Phi\Big(\sum_{j=1}^{M} w_j(k)x(k-j) + w_{M+1}(k) + \\ + \sum_{m=1}^{N} w_{m+M+1}(k)y(k-m)\Big)$$

or $y(k) = \Phi(net(k)) = \Phi(\mathbf{w}^T(k)\mathbf{u}(k))$

**In other words, we have an NARMA(N,M) model.**

## Weight update for recurrent perceptron

$$\Delta w_i(k) = -\eta \frac{\partial E(k)}{\partial w_i(k)} = e(k)\frac{\partial e(k)}{\partial w_i(k)} = -e(k)\frac{\partial y(k)}{\partial w_i(k)} \tag{1}$$

$$\frac{\partial y(k)}{\partial w_i(k)} = \Phi'\left(net(k)\right)\frac{\partial net(k)}{\partial w_i(k)} \tag{2}$$

$$\frac{\partial y(k)}{\partial w_i(k)} \approx \Phi'(net(k))\left[u_i(k) + \sum_{m=1}^{N} w_{m+M+1}(k)\frac{\partial y(k-m)}{\partial w_i(k-m)}\right] \tag{3}$$

where vector $\mathbf{u}(k) = [x(k), \ldots, x(k-M), 1, y(k), \ldots, y(k-N)]^T$
If we introduce notation $\pi_i(k) = \frac{\partial y(k)}{\partial w_i(k)}$, then

$$\pi_i(k) = \Phi'(net(k))\left[u_i(k) + \sum_{m=1}^{N} w_{m+M+1}(k)\pi_i(k-m)\right] \tag{4}$$

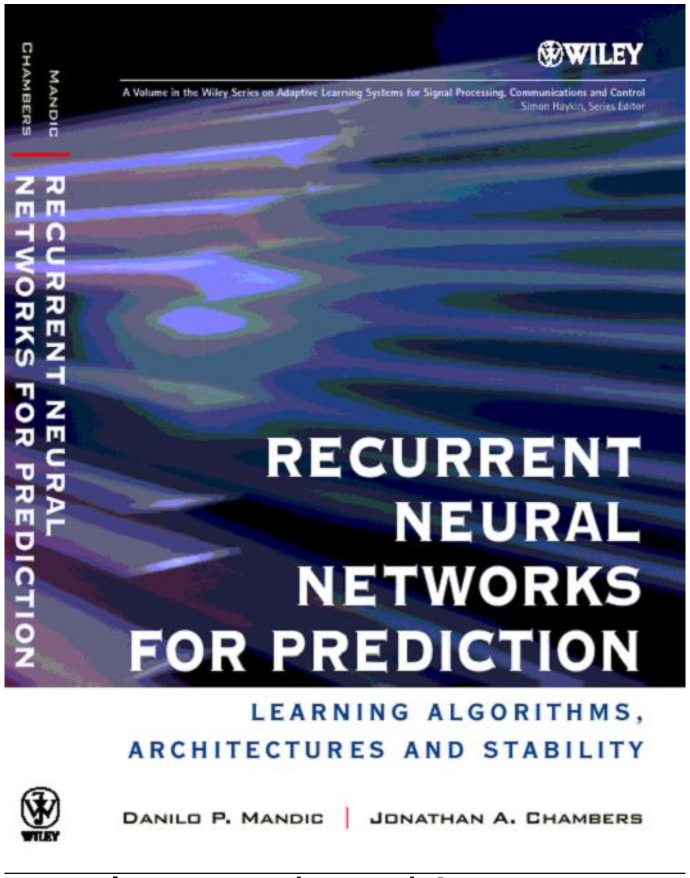
In control theory, coefficients $\pi_i(k)$ are called *sensitivities*.

## Conclusions

- Adaptive processing of signals observed through possibly nonlinear sensors, and at multiple scales
- The recursive least squares (RLS), a 'deterministic' solution to the estimation of streaming data
- A general form of learning algorithms, optimality of the gain factor
- Black box, grey box and white box modelling
- Parametric, non-parametric, and semi-parametric modelling
- How can we 'make sense' of similar data classes which are observed in different lighting conditions, shades, and viewing angles
- Perceptron, neural network, recurrent perceptron
- Neural networks as 'nonlinear ARMA', that is, NARMA systems
- Backpropagation and real time recurrent learning (RTRL) algorithms
- This promises enhanced practical solutions in a variety of applications

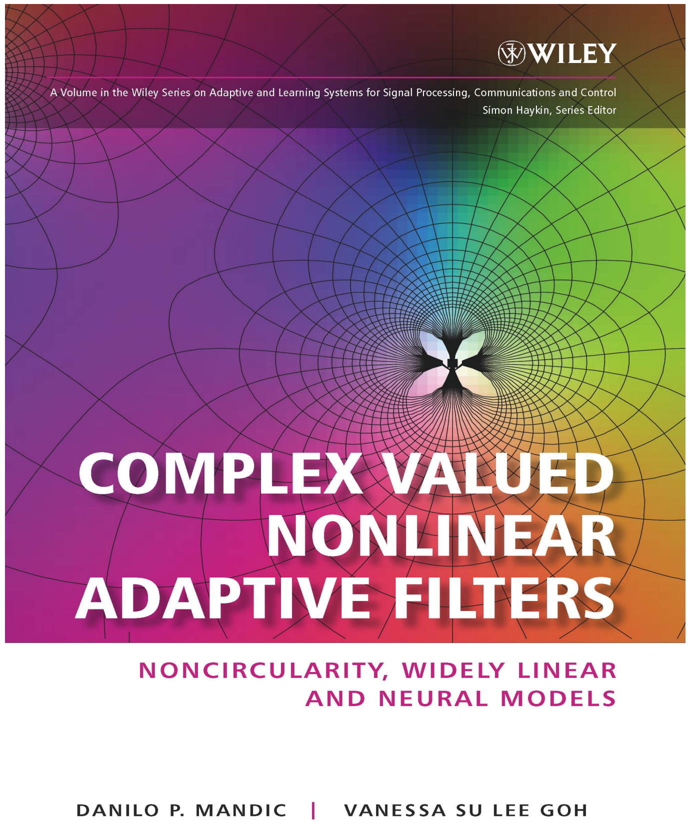
## Additional reading

**D. Mandic & J. Chambers (*RNNs for Prediction*, Wiley 2001).**


WILEY
A Volume in the Wiley Series on Adaptive Learning Systems for Signal Processing, Communications and Control
Simon Haykin, Series Editor
MANDIC CHAMBERS
RECURRENT NEURAL NETWORKS FOR PREDICTION
RECURRENT NEURAL NETWORKS FOR PREDICTION
LEARNING ALGORITHMS, ARCHITECTURES AND STABILITY
DANILO P. MANDIC | JONATHAN A. CHAMBERS

neural network architectures

**D. Mandic and S. Goh (*Complex Adaptive Filters*, Wiley 2009).**


WILEY
A Volume in the Wiley Series on Adaptive and Learning Systems for Signal Processing, Communications and Control
Simon Haykin, Series Editor
COMPLEX VALUED NONLINEAR ADAPTIVE FILTERS
NONCIRCULARITY, WIDELY LINEAR AND NEURAL MODELS
DANILO P. MANDIC | VANESSA SU LEE GOH

real, complex, and neural adaptive filters

## Some back–up material

## Appendix: Inverse of a matrix

### We need to calculate those inverses for our main models

**The structure of an inverse matrix is also important!**

- A symmetric matrix has a symmetric inverse
- A **Toeplitz** matrix has a persymmetric inverse (symmetric about the cross diagonal)
- A Hankel matrix has a symmetric inverse (useful in harmonic retrieval)

A useful formula for matrix inversion is **Woodbury's identity** (matrix inversion Lemma)

$$\left(\mathbf{A}+\mathbf{U}\mathbf{V}^{\mathbf{T}}\right)^{-1}=\mathbf{A}^{-1}-\left[\mathbf{A}^{-1}\mathbf{U}\left(\mathbf{I}+\mathbf{V}^{\mathbf{T}}\mathbf{A}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^{\mathbf{T}}\mathbf{A}^{-1}\right]$$

If $\mathbf{u}$ and $\mathbf{v}$ are vectors, then this simplifies into

$$(\mathbf{A}+\mathbf{u}\mathbf{v}^T)^{-1}=\mathbf{A}^{-1}-\frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1+\mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}$$

which will be important in the derivation of adaptive algorithms.

In a special case $\mathbf{A}=\mathbf{I}$ and we have

$$(\mathbf{I}+\mathbf{u}\mathbf{v}^T)^{-1}=\mathbf{I}-\frac{\mathbf{u}\mathbf{v}^T}{1+\mathbf{v}^T\mathbf{u}}$$

## Example: Matrix inversion lemma $\hookrightarrow$ another approach

Construct an 'augmented' matrix $[A, B; C, D]$ and its inverse $[E, F; G, H]$, so that

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

**Consider the following two products**

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE+BG & AF+BH \\ CE+HC & CF+DH \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

and

$$\begin{bmatrix} E & F \\ G & H \end{bmatrix}\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} EA+FC & EB+FD \\ GA+HC & GB+HD \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

Combine the corresponding terms to get another form

$$\left(\mathbf{A}-\mathbf{B}\mathbf{D}^{-1}\mathbf{C}\right)^{-1} = \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D}-\mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{C}\mathbf{A}^{-1}$$

## Appendix: The RLS algorithms and its variants

**Accelerated LMS:**

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{C}(n)\mathbf{x}(n), \quad \mathbf{C} \text{ a chosen matrix}$$

**Normalised LMS:**

$$\mu(n) = \frac{\mu}{\varepsilon + \mathbf{x}^T(n)\mathbf{x}(n)}, \quad 0 < \mu < 2$$

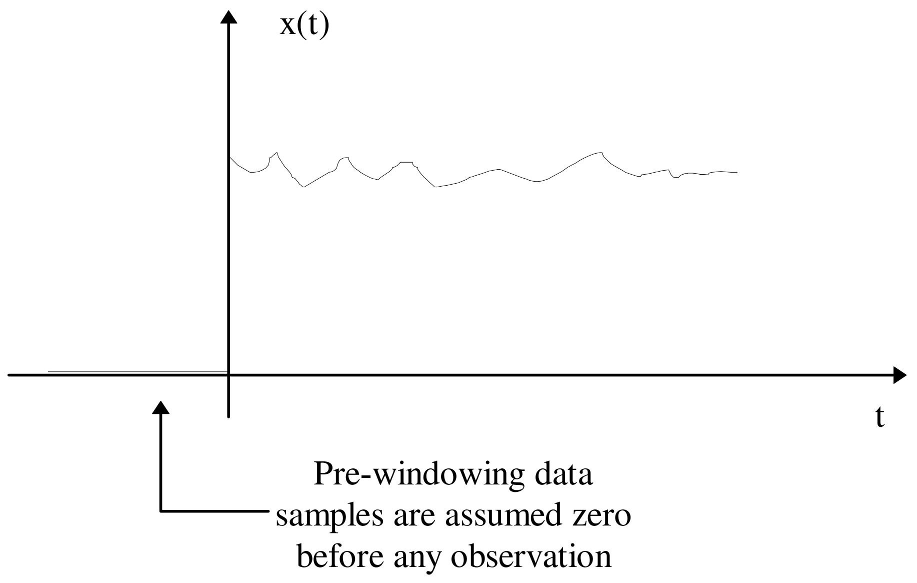
**Exponentially weighted RLS** (RLS with the forgetting factor)

$$J(n) = \sum_{i=0}^{n} \lambda^{n-i} \big| d(i) - y(i) \big|^2, \quad \lambda \text{ is a forgetting factor}$$

- Forgetting factor $\lambda \in (0, 1]$, but typically $> 0.95$
- The forgetting factor introduces an effective window length of $\frac{1}{1-\lambda}$

## Appendix: Pre–windowing

- To initialise RLS we need to make some assumptions about the input data
- The most common one is that of prewindowing, that is $\forall k < 0, x(k) = 0$


x(t)
t
Pre-windowing data samples are assumed zero before any observation

## Appendix: Complete RLS algorithm

- Initialisation $\mathbf{w}(0) = \mathbf{0}$, $\mathbf{P}(0) = \delta\mathbf{I}$, $\mathbf{P}(k) = \mathbf{R}^{-1}$

- For each $k$

$$
\begin{aligned}
\gamma(k+1) &= \mathbf{x}^T(k+1)\mathbf{P}(k)\mathbf{x}(k+1) \\
\mathbf{k}(k+1) &= \frac{\mathbf{P}(k)\mathbf{x}(k+1)}{1+\gamma(k+1)} \\
\mathbf{e}(k+1) &= \mathbf{d}(k+1) - \mathbf{x}^T(k+1)\mathbf{w}(k+1) \\
\mathbf{w}(k+1) &= \mathbf{w}(k)+\mathbf{k}(k+1)\mathbf{e}(k+1)=\mathbf{w}(k)+\mathbf{R}^{-1}(k+1)\mathbf{x}(k+1)\mathbf{e}(k+1) \\
\mathbf{P}(k+1) &= \mathbf{P}(k) - \mathbf{k}(k+1)\mathbf{x}^T(k)\mathbf{P}(k)
\end{aligned}
$$

- Computational complexity of RLS is an $\mathcal{O}(p^2)$ in terms of multiplications and additions as compared to LMS and NLMS which are $\mathcal{O}(2N)$

## A general form of recursive learning algorithms

Let us introduce a new 'rotated' or 'prewhitened' vector

$$\mathbf{k}(k) = \mathbf{R}^{-1}(k)\mathbf{x}(k)$$

and

$$\mathbf{R}^{-1}(k+1) = \mathbf{R}^{-1}(k) - \mathbf{k}(k+1)\mathbf{x}^T(k+1)\mathbf{R}^{-1}(k)$$

**A general form of the RLS algorithm:**

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{k}(k+1)\alpha(k+1)$$

where

$$\alpha(k+1) = d(k+1) - \mathbf{x}^T(k+1)\mathbf{w}(k)$$

Cf. LMS - the term $\mu\mathbf{x}(k)$ has been replaced by a time–varying matrix $\mathbf{k}(k)$

## An even more general framework from LMS to Kalman filter

- Gradient algorithms can track the weights only approximately $\looparrowright$ we use the symbol $\hat{\mathbf{w}}$, to give a general expression for a weight update

$$\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \eta\mathbf{\Gamma}(k)\left(y(k) - \mathbf{x}^T(k)\hat{\mathbf{w}}(k)\right)$$

  where $\mathbf{\Gamma}(k)$ is a gain matrix which determines 'adaptation direction'

- To asses how far an adaptive algorithm is from the optimal solution, consier the weight error vector, $\tilde{\mathbf{w}}(k)$, and sample input matrix $\mathbf{\Sigma}(k)$

$$\tilde{\mathbf{w}}(k) = \mathbf{w}(k) - \hat{\mathbf{w}}(k), \qquad \mathbf{\Sigma}(k) = \mathbf{\Gamma}(k)\mathbf{x}^T(k)$$

- We now have a new weight error equation

$$\tilde{\mathbf{w}}(k+1) = \left(\mathbf{I} - \eta\mathbf{\Sigma}(k)\right)\tilde{\mathbf{w}}(k) - \eta\mathbf{\Gamma}(k)\nu(k) + \Delta\mathbf{w}(k+1)$$

## LMS, RLS and Kalman filter

### For different gains different learning algorithms can be obtained

**The Least Mean Square** (LMS) algorithm

$$\mathbf{\Sigma}(k) = \mathbf{x}(k)\mathbf{x}^T(k)$$

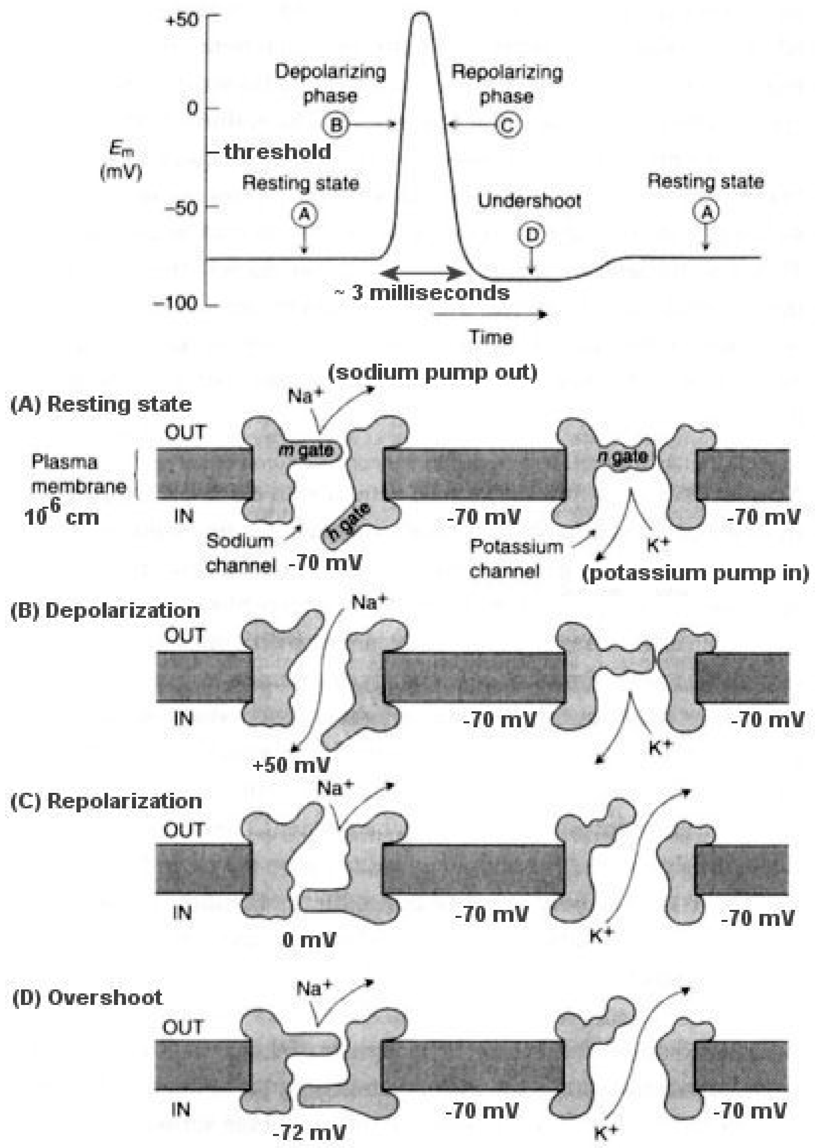
**The Recursive Least Squares** (RLS) algorithm

$$\begin{aligned}\mathbf{\Sigma}(k) &= P(k)\mathbf{x}(k)\mathbf{x}^T(k)\\ P(k) &= \frac{1}{1-\eta}\left[P(k-1) - \eta\frac{P(k-1)\mathbf{x}(k)\mathbf{x}^T(k)P(k-1)}{1-\eta+\eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)}\right]\end{aligned}$$

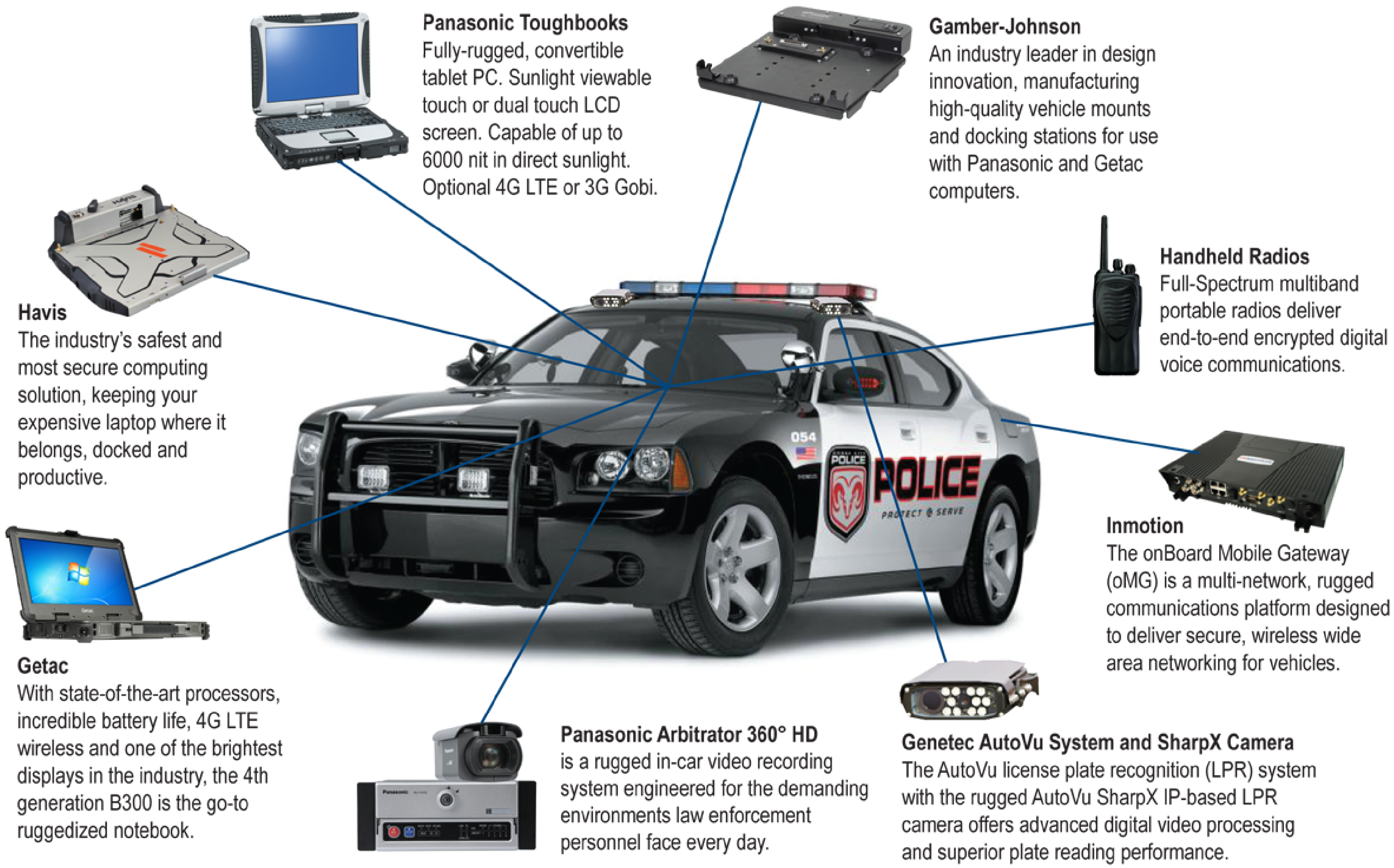
**The Kalman Filter** (KF) algorithm

$$\begin{aligned}\mathbf{\Sigma}(k) &= \frac{P(k-1)\mathbf{x}(k)\mathbf{x}^T(k)}{R+\eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)}\\ P(k) &= P(k-1) - \frac{\eta P(k-1)\mathbf{x}(k)\mathbf{x}^{\mathbf{T}}(k)P(k-1)}{R+\eta\mathbf{x}^T(k)P(k-1)\mathbf{x}(k)} + \eta Q\end{aligned}$$

## Appendix: Action potentials



## Smart sensor nodes $\looparrowright$ a police car



**Panasonic Toughbooks**
Fully-rugged, convertible tablet PC. Sunlight viewable touch or dual touch LCD screen. Capable of up to 6000 nit in direct sunlight. Optional 4G LTE or 3G Gobi.

**Gamber-Johnson**
An industry leader in design innovation, manufacturing high-quality vehicle mounts and docking stations for use with Panasonic and Getac computers.

**Havis**
The industry's safest and most secure computing solution, keeping your expensive laptop where it belongs, docked and productive.

**Handheld Radios**
Full-Spectrum multiband portable radios deliver end-to-end encrypted digital voice communications.

**Inmotion**
The onBoard Mobile Gateway (oMG) is a multi-network, rugged communications platform designed to deliver secure, wireless wide area networking for vehicles.

**Getac**
With state-of-the-art processors, incredible battery life, 4G LTE wireless and one of the brightest displays in the industry, the 4th generation B300 is the go-to ruggedized notebook.

**Panasonic Arbitrator 360° HD**
is a rugged in-car video recording system engineered for the demanding environments law enforcement personnel face every day.

**Genetec AutoVu System and SharpX Camera**
The AutoVu license plate recognition (LPR) system with the rugged AutoVu SharpX IP-based LPR camera offers advanced digital video processing and superior plate reading performance.

## Some notions from estimation theory

- **System:** actual underlying physics[1] that generate the data;
- **Model:** a mathematical description of the system;
- Many variations of mathematical models can be postulated on the basis of datasets collected from observations of a system, and their suitability assessed by various performance metrics;
- Since it is not possible to characterise nonlinear systems by their impulse response, we resort to less general models, e.g. polynomial filters.
- Some of the most frequently used polynomial filters are based upon
  - Volterra series, a nonlinear analogue of the linear impulse response,
  - threshold autoregressive models (TAR)
  - Hammerstein and Wiener models.

[1]Technically, the notions of *system* and *process* are equivalent.

## Basic modes of modelling

- **Parametric** modelling assumes a fixed structure for the model. The model identification problem then simplifies to estimating a finite set of parameters of this fixed model. An example of this technique is the broad class of ARIMA/NARMA models.

- **Nonparametric** modelling seeks a particular model structure from the input data. The actual model is not known beforehand.
  We look for a model in the form of $y(k) = f\left(x(k)\right)$ without knowing the function $f(\cdot)$.

- **Semiparametric** modelling is the combination of the above. Part of the model structure is completely specified and known beforehand, whereas the other part of the model is either not known or loosely specified.

Neural networks, especially recurrent neural networks, can be employed within estimators of all of the above classes of models. Closely related to the above concepts are white, gray, and black box modelling techniques.

## White–, Gray–, and Black–Box modelling

- The idea is to distinguish between three levels of prior knowledge, which have been "colour–coded"
- **White box** $\looparrowright$ the underlying physical process has to be understood (planetary motions, Kepler's laws, gravitation)
- **Black box** $\looparrowright$ no previous knowledge about data production system. The aim is to find a function $F$ that approximates[2] the process $y$ based on the previous observation of the process $y_{PAST}$ and input $u$, as

  $$y = F\left(y_{PAST}, u\right)$$

  This is achieved without any knowledge of the underlying signal generating model
- **Grey box** $\looparrowright$ obtained from black box modelling if some information about the system is known **a priori**

[2]The downside is that it is generally not possible to learn about te true physical process that generates the data, especially if a linear model is used

## RNNs and Polynomial Filters

Perform a Taylor series expansion (TSE) for the output of a multiplicative NN for which input signals are $u(k-1)$ and $u(k-2)$

$$\begin{aligned} y(k) &= c_0 + c_1 u(k-1) + c_2 u(k-2) + c_3 u^2(k-1) + c_4 u^2(k-2) + \\ &\quad c_5 u(k-1)u(k-2) + c_6 u^3(k-1) + c_7 u^3(k-2) + \cdots \end{aligned}$$

☞ the form of a general Volterra series, given by

$$\begin{aligned} y(k) &= h_0 + \sum_{i=0}^{N} h_1(i)x(k-i) + \\ &\quad \sum_{i=0}^{N}\sum_{j=0}^{N} h_2(i,j)x(k-i)x(k-j) + \cdots \end{aligned}$$

**Representation by a neural network is therefore more compact!**

**RNNs have advantages:** Volterra series are not suitable for modelling saturation type nonlinearities, and systems with nonlinearities of a high order.

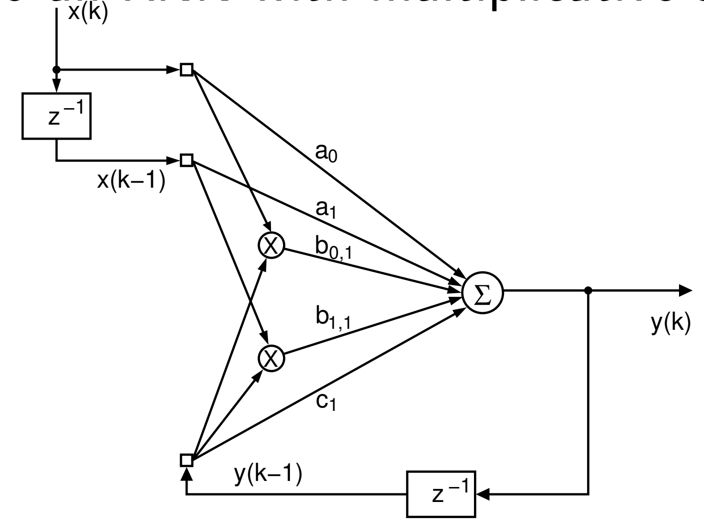
## Bilinear series

**Problem:** Since the Volterra series = TSE with memory, it cannot model systems with discontinuity.

**Solution: Bilinear (truncated Volterra) model**

$$y(k) = \sum_{j=1}^{N-1} c_i y(k-j) + \sum_{i=0}^{N-1}\sum_{j=1}^{N-1} b_{i,j} y(k-j)x(k-i) + \sum_{i=0}^{N-1} a_i x(k-i)$$

which is equivalent to an RNN with multiplicative synapses



$$y(k) = c_1 y(k-1) + b_{0,1} x(k) y(k-1) + b_{1,1} x(k-1) y(k-1) + a_0 x(k) + a_1 x(k-1)$$

## Notes

○

## Notes

∘

## Notes

○

## Notes

○