## Associative Memory Networks

$$y = h(\boldsymbol{x}) = \sum_{q=1}^{Q} w_q \phi_q^*(\boldsymbol{x}), \tag{1}$$

where $\phi_q^*(\boldsymbol{x})$, $(q = 1, \ldots, Q)$ are the basis functions.

- Non-Lattice
  - RBF networks,
- Lattice
  - B-splines,
  - Neuro-Fuzzy,
  - CMAC.

Prior to the phase of parameter estimation it is necessary to decide about some structural parameters:

- the number of basis functions (i.e. number of parameters to be estimated);
- the shape of each basis function;
- the location of each basis function;

**But how is possible to decide about these issues ?**

These issues are of course problem dependent and it is highly conected with the amount of prior knowledge available.

## Radial Basis Function Networks

$\phi_q^*(\boldsymbol{x}) = f(\parallel \boldsymbol{c}_i - \boldsymbol{x} \parallel_2)$

Let $r = \parallel \boldsymbol{c}_i - \boldsymbol{x} \parallel_2$, for the basis function which has a centre in $\boldsymbol{c}$, then several different choices for $f(.)$ are given by:

- the radial linear function: $f(r) = r$,
- the radial cubic function: $f(r) = r^3$,
- the Gaussian function: $f(r) = \exp(-r^2/(2\sigma^2))$,
- the thin plate function: $f(r) = r^2 \log(r)$,
- the multi-quadratic function: $f(r) = (r^2 + \sigma^2)^{0.5}$,
- the shifted logarithm function: $f(r) = \log(r^2 + \sigma^2)$.

- The Gaussian function is the one that is the most used;
- It is the only one that can be written as the product of univariate functions:

$$f_q(\boldsymbol{x}) = \exp\left(-\frac{\| \boldsymbol{c}_q - \boldsymbol{x} \|_2^2}{2\boldsymbol{\sigma}_q^2}\right) = \prod_{l=1}^{n} \exp\left(-\frac{(c_{ql} - x_l)^2}{(2\sigma_q^2)}\right) \tag{2}$$

- Centers of RBFs can be chosen using an unsupervised learning algorithm. Like for example, $k-$means clustering, which partitions the training data in $Q$ clusters where the total squared Euclidian distances between the training inputs $\boldsymbol{x}_i$ and their corresponding cluster centers $\boldsymbol{c}_q$'s are minimised,

$$E_{k-means} = \sum_{i \in N} \sum_{q \in Q} \alpha_{iq} \parallel \boldsymbol{x}_i - \boldsymbol{c}_q \parallel^2, \tag{3}$$

In this Equation, $\alpha_{iq} = 1$, if the Euclidian distance from $\boldsymbol{x}_i$ to $\boldsymbol{c}_q$ is the smallest one. Otherwise, $\alpha_{iq} = 0$.

- The shape factor $\boldsymbol{\sigma}_q$ of each Gaussian function can be chosen as a mean distance between the $\boldsymbol{c}_j$ and the first few neighbouring nodes, por exemplo, $M_s = 2$, is usually adopted,

$$\boldsymbol{\sigma}_q = \frac{1}{M_s} \sum_{m=1}^{M_s} \parallel \boldsymbol{c}_{q_m} - \boldsymbol{c}_q \parallel, \tag{4}$$

where $q_m$ denotes the first $m$ nearest units the $q$th Gaussian centre.

## Lattice Associative memory networks

- B-splines,
- Neuro-fuzzy models,
- CMAC networks.

## Disadvantages of lattice based networks

Both neuro-fuzzy models and B-spline networks are lattice networks making the design of the model easy but introducing some drawbacks. The first step is to define the unidimensional membership functions or basis functions. Then multidimensional basis functions are constructed by tensor multiplication of uni-dimensional basis functions.

The total number of basis functions is defined by the product:

$$Q = \prod_{l=1}^{n} r_l, \tag{5}$$

where $r_l$ is the number of uni-dimensional basis functions for each variable $x_l$.

Each additional uni-dimensional basis functions in one of the variables adds many multi-dimensional basis functions. For example, if one basis function is added to $x_p$ then the new number of basis functions will be:

$$Q_{new} = Q_{old} + \prod_{l=1, l \neq p}^{n} r_l.$$

The effect is illustrated in Figure 1. This sometimes sometimes leads to unnecessary complexity of the model and overparameterization which may result in poor generalisation.

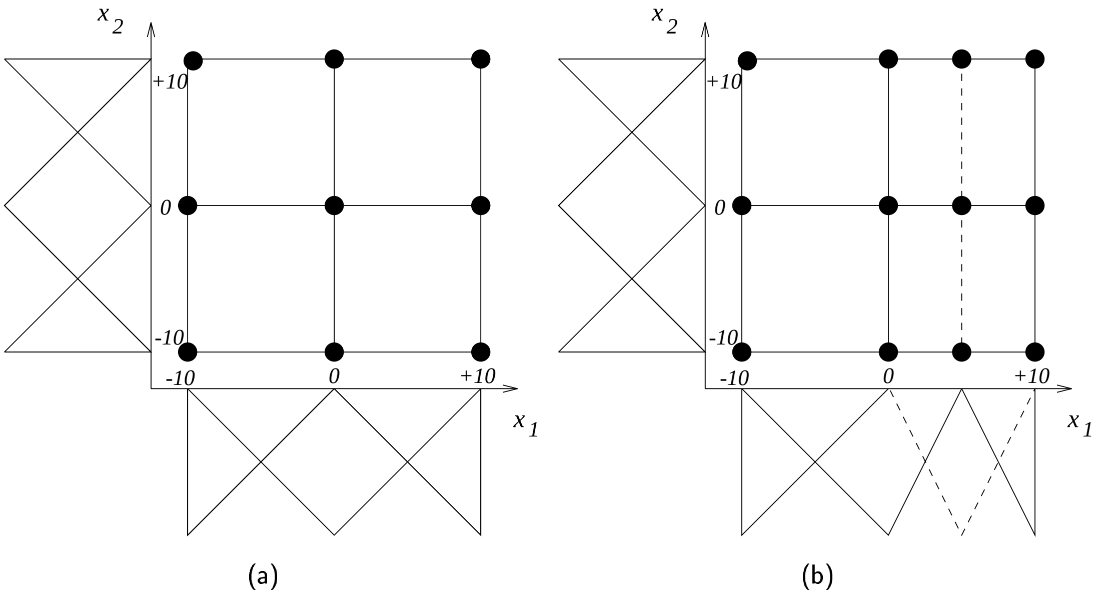

Figura 1: (a) Three basis functions are defined for each variable that results in nine bi-dimensional basis functions. (b) One basis function is added to $x_1$ resulting in 12 bi-dimensional basis functions.

- Another issue is that the number of basis functions and the associated amount of training data required grow exponentially as the number of input variables grows linearly.
- This property is known as *curse of dimensionality* and it limits the use of these models to low dimensional problems.
- Recently the research has focused in alternatives that can result in parsimonious models using techniques like Adaptive B-spline Basis Modelling of Observation Data (ASMOD) and Adaptive B-spline Basis function Modelling of Observation Data (ABB-MOD) algorithms.

# Linear Equations

$$y(t) = \mathbf{\Phi}(\boldsymbol{x}, t)\boldsymbol{W}, \tag{6}$$

where:

$$\mathbf{\Phi}(\boldsymbol{x}) = \left[ \begin{array}{ccccc} \phi_1^*(\boldsymbol{x}) & \dots & \phi_q^*(\boldsymbol{x}) & \dots & \phi_Q^*(\boldsymbol{x}) \end{array} \right], \tag{7}$$

and:

$$\boldsymbol{W} = \left[ \begin{array}{ccccc} w_1 & \dots & w_q & \dots & w_Q \end{array} \right]^T. \tag{8}$$

This property allows the use of the well known Least Squares algorithms. The Batch Least Squares algorithm is the simplest algorithm to use. For the set of training data $\mathcal{D} = \{(\boldsymbol{x}(t), y(t)), t = 1, \ldots, N\}$, it is possible to estimate the parameters using the following algorithm:

$$\hat{\boldsymbol{W}} = (\overline{\boldsymbol{\Phi}}^T \overline{\boldsymbol{\Phi}})^{-1} \overline{\boldsymbol{\Phi}}^T Y, \tag{9}$$

where:

$$\begin{aligned} Y &= \begin{bmatrix} y(1) & \ldots & y(N) \end{bmatrix}^T, \quad \text{and} && \text{(10a)} \\ \overline{\boldsymbol{\Phi}} &= \begin{bmatrix} \overline{\phi}_1^* & \ldots & \overline{\phi}_q^* & \ldots & \overline{\phi}_Q^* \end{bmatrix}, \quad \text{and} && \text{(10b)} \\ \overline{\phi}_q^* &= \begin{bmatrix} \phi_q^*(\boldsymbol{x}(1)) & \ldots & \phi_q^*(\boldsymbol{x}(N)) \end{bmatrix}^T. && \text{(10c)} \end{aligned}$$

The matrix, $\overline{\boldsymbol{\Phi}}$, is a $N \times Q$ matrix, and it becomes difficult to handle when $Q$ and $N$ become very large. Also, the method is ill-conditioned if $(\overline{\boldsymbol{\Phi}}^T \overline{\boldsymbol{\Phi}})^{-1}$ is nearly singular. For example, if one of the basis functions is not activated by any data, matrix $\overline{\boldsymbol{\Phi}}$ will have one of the columns equal to zero and estimation will be impossible.

- Usually, Associative Memory networks contain a large number of rules or weights $Q$.
- In this case, algorithms based on instantaneous gradient descent rules are good alternatives. They can also be implemented on-line due to low cost computations.

## LMS (Least Mean Square) Algorithm

$$\Delta \boldsymbol{w}(t-1) = \delta \epsilon_y(t) \boldsymbol{\Phi}(t), \tag{11}$$

where $\delta$ is the learning rate,

$$\epsilon_y(t) = \hat{y}(t) - y(t) \tag{12}$$

and

$$\boldsymbol{\Phi}(\boldsymbol{x}) = \left[ \begin{array}{ccccc} \phi_1^*(\boldsymbol{x}) & \ldots & \phi_q^*(\boldsymbol{x}) & \ldots & \phi_Q^*(\boldsymbol{x}) \end{array} \right], \tag{13}$$

## NLMS (Normalised Least Mean Square) Algorithm

$$\Delta \boldsymbol{w}(t-1) = \delta \epsilon_y(t) \frac{\boldsymbol{\Phi}(t)}{\parallel \boldsymbol{\Phi}(t) \parallel_2^2}, \tag{14}$$

where,

$$\parallel \boldsymbol{\Phi}(t) \parallel_2^2 = \boldsymbol{\Phi}^T(t) \boldsymbol{\Phi}(t). \tag{15}$$

To ensure faster convergence, the Recursive Least Squares (RLS) algorithm can be used:

$$
\begin{aligned}
\hat{y}(t) &= \boldsymbol{\Phi}(\boldsymbol{x}, t)\hat{\boldsymbol{W}}(t), && (16)\\
e(t) &= y(t) - \hat{y}(t), && (17)\\
\hat{\boldsymbol{W}}(t) &= \hat{\boldsymbol{W}}(t-1) + \frac{P(t-1)\boldsymbol{\Phi}(t)}{1 + \boldsymbol{\Phi}^T(t)P(t-1)\boldsymbol{\Phi}(t)} e(t), && (18)\\
P(t) &= P(t-1) - \frac{P(t-1)\boldsymbol{\Phi}(t)\boldsymbol{\Phi}^T(t)P(t-1)}{1 + \boldsymbol{\Phi}^T(t)P(t-1)\boldsymbol{\Phi}(t)}, && (19)\\
P(0) &= \sigma I, && (20)
\end{aligned}
$$

where $\sigma$ is a large number.

The memory required to store the covariance matrix $P(t)$, and the matrix computations to update it and the vector of parameters $\hat{\boldsymbol{W}}(t)$ make this algorithm computationally expensive and not suitable for use with high-complexity models.

For neuro-fuzzy models, the complex matrix computations can be avoided if a special feature of the model is exploited. For each data point the vector of the basis functions $\mathbf{\Phi}(t)$ has only a few non-zero elements. Then, if the algorithm initially detects the non-zero elements, a substantial amount of computation can be saved during the evaluation of $P(t-1)\mathbf{\Phi}(t)$ and $\mathbf{\Phi}^T(t)P(t-1)$.

For very commonly used triangular basis function (B-splines of order 2), one data point activates two basis functions for each variable of the inputs of the model. Thus, for $n$ input variables, each data point activates $2^n$ basis functions as can seen in Figure.

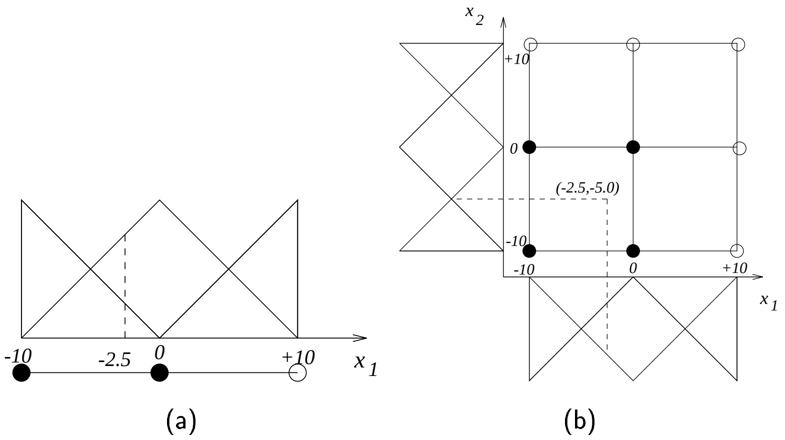

Figura 2: (a) Each point activates two uni-dimensional basis functions. (b) Each pair of points activates 4 bi-dimensional basis functions.

Other Least Squares algorithms which have more robust numerical properties can also be used, such as the Bierman's $UD$ algorithm.

For this case, the matrix $P(t-1)$ factored as:

$$P(t-1) = U(t-1)D(t-1)U(t-1)^T,$$

where $U(t-1)$ is upper triangular and $D(t-1)$ is a diagonal matrix. But the storage requirements and matrix computations are greater than those of the RLS algorithm. Also, the sparseness of vector $\mathbf{\Phi}(t)$ can not be fully exploited.