Computational Intelligence

Master in Artificial Intelligence

Lluís A. Belanche and René Alquézar

belanche@cs.upc.edu, alquezar@cs.upc.edu

Soft Computing Research Group
Dept. de Ciències de la Computació (Computer Science)

Universitat Politècnica de Catalunya

2019-2020

Artificial Neural Networks: the RBFNN

The Gaussian Distribution

A continuous d-variate random vector $\boldsymbol{X} = (X_1, \ldots, X_d)^\top$ is **normally distributed**, written $\boldsymbol{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, when its joint pdf is:

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu}) \right\}$$

where $\boldsymbol{\mu}$ is the *mean vector* and $\Sigma_{d\times d} = (\sigma_{ij}^2)$ is the (real symmetric and PD) *covariance matrix*.

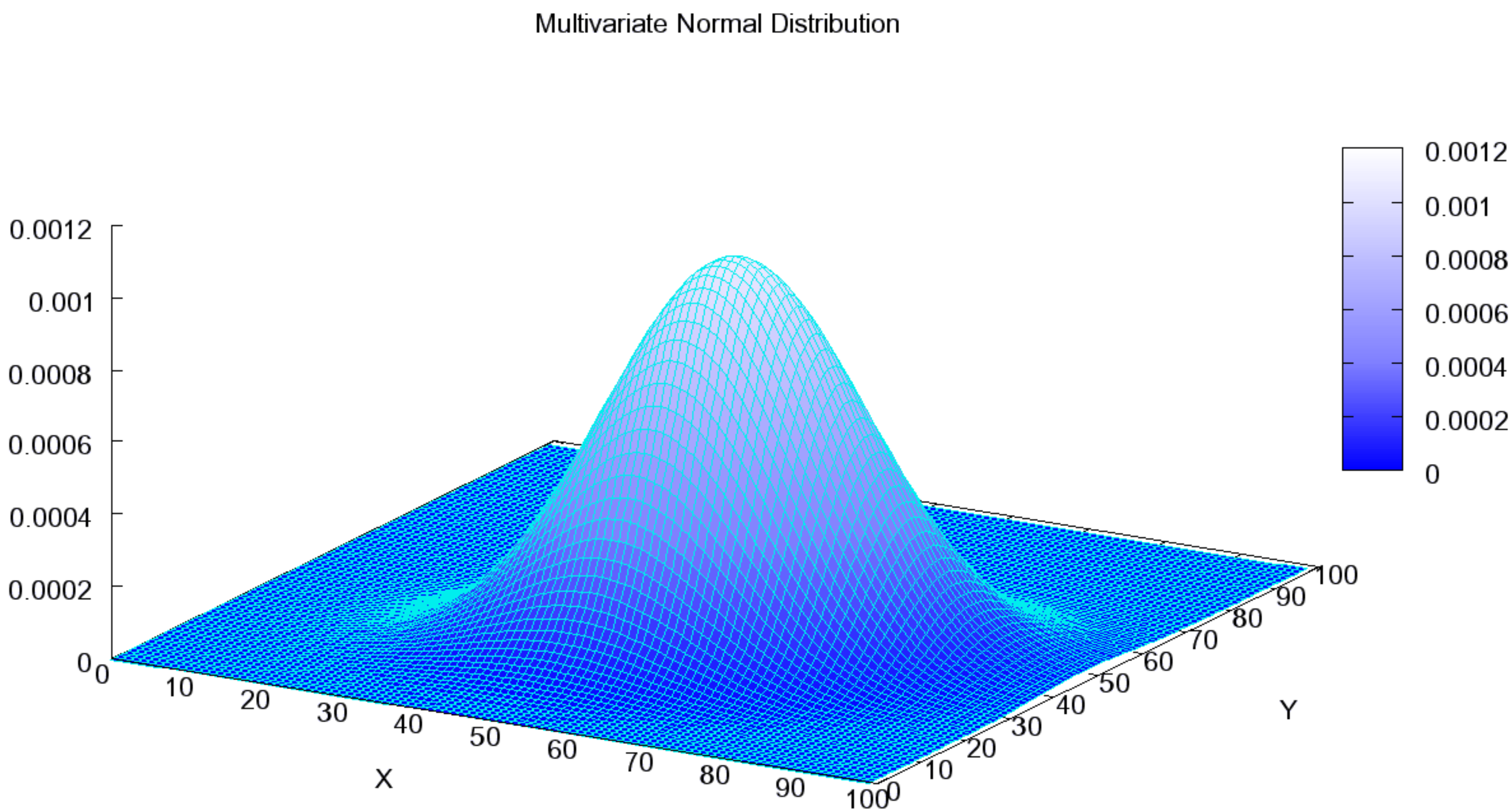
- $\mathbb{E}[\boldsymbol{X}] = \boldsymbol{\mu}$ and $\mathbb{E}[(\boldsymbol{X} - \boldsymbol{\mu})(\boldsymbol{X} - \boldsymbol{\mu})^\top] = \Sigma$.
- $CoVar[X_i, X_j] = \sigma_{ij}^2$ and $Var[X_i] = \sigma_{ii}^2$

---

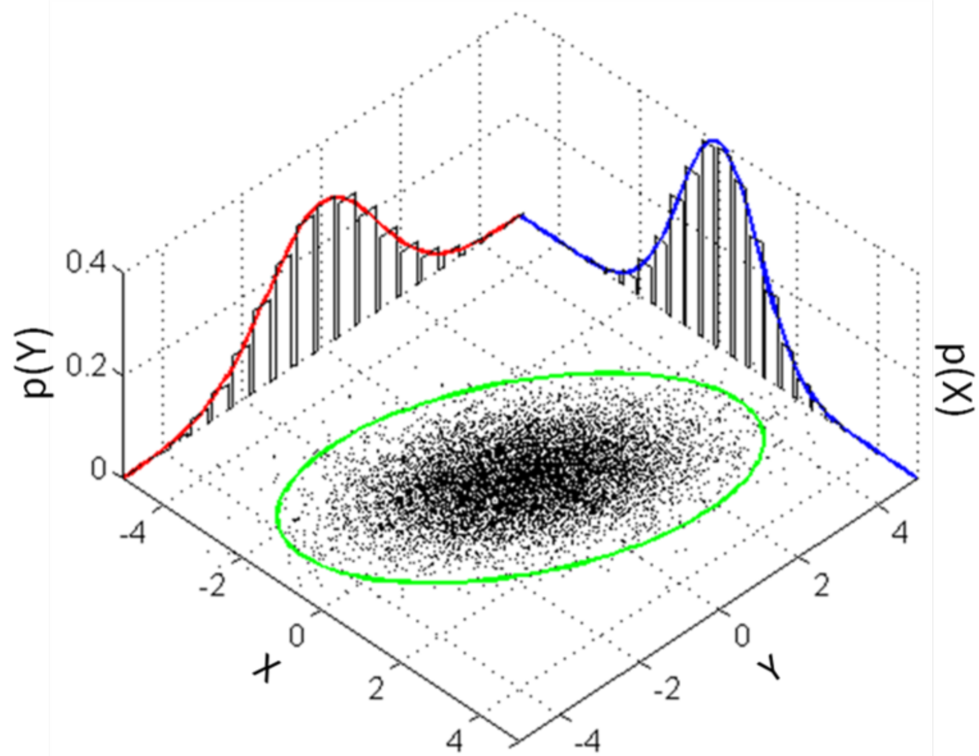
if $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, then X_i, X_j are independent $\Longleftrightarrow CoVar[X_i, X_j] = 0$

(in general, only the left-to-right implication holds)

The Gaussian Distribution ($d = 2$)

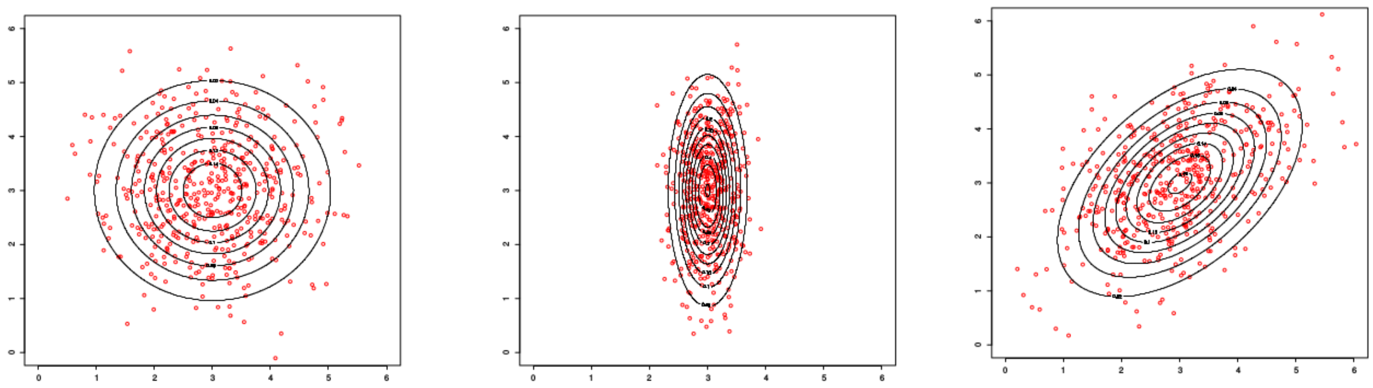


The Gaussian Distribution ($d = 2$)



Observations from a bivariate normal distribution, a contour ellipsoid, the two marginal distributions, and their histograms (images from the Wikipedia)

The Gaussian Distribution ($d = 2$)



$\boldsymbol{\mu} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$ $\quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\qquad \boldsymbol{\Sigma} = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix}$ $\qquad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$

- The principal directions (a.k.a. PCs) of the hyperellipsoids are given by the *eigenvectors* $\boldsymbol{u}_i$ of Σ, which satisfy $\boldsymbol{\Sigma u}_i = \lambda_i \boldsymbol{u}_i$.

- The lengths of the hyperellipsoids along these axes are proportional to $\sqrt{\lambda_i}$ (note $\lambda_i > 0$)

<u>The Gaussian Distribution</u>

Conceptual view

- What is behind the choice of a **multivariate Gaussian**?

  Examples from a class are noisy versions of an ideal class member (a *prototype*):

  – Prototype: modeled by the mean vector

  – Noise: modeled by the covariance matrix

- The quantity

$$d(\boldsymbol{x}) := \sqrt{(\boldsymbol{x} - \boldsymbol{\mu})^{\mathsf{T}} \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{\mu})}$$

  is called the **Mahalanobis distance** for $\boldsymbol{x}$

- Very important! the number of parameters is $\frac{d(d+1)}{2} + d$

The Gaussian Distribution

Mathematical view

Positive definiteness: for a Gaussian distribution to be well-defined, Σ has to be real symmetric and positive definite (PD): for all non-null vectors $\boldsymbol{x} \in \mathbb{R}^d$, $\boldsymbol{x}^\mathsf{T}\Sigma\boldsymbol{x} > 0$ must hold true.

Examples: are these matrices PD in $d = 2$?

$$a. \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad b. \begin{pmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{pmatrix}$$

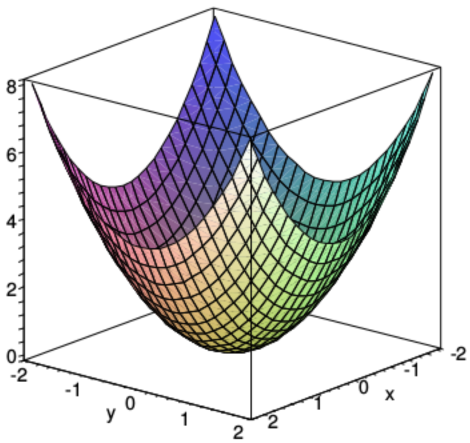
$$c. \begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix} \qquad d. \begin{pmatrix} 1 & 4 \\ \frac{1}{2} & 1 \end{pmatrix}$$

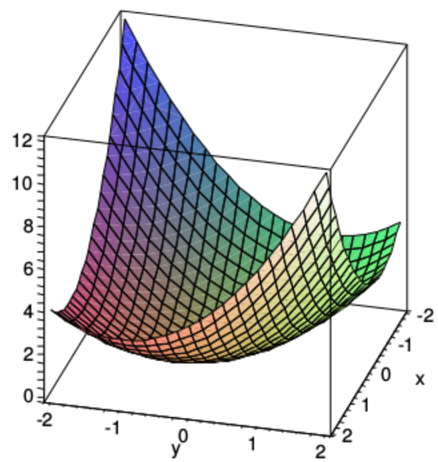
a. YES; b. YES

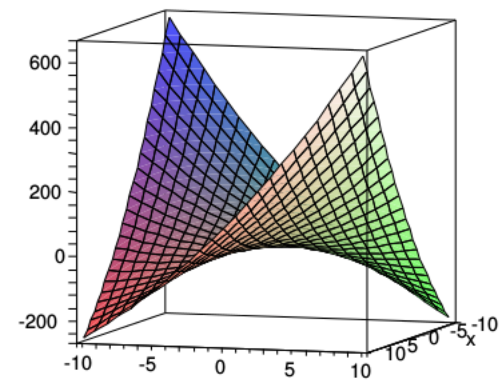
c. YES; d. NO

The Gaussian Distribution

Mathematical view







$a.\ x_1^2 + x_2^2;$ $b.\ x_1^2 + x_1 x_2 + x_2^2;$ $d.\ x_1^2 + \frac{9}{2} x_1 x_2 + x_2^2$

Artificial neural networks: the RBFNN

Introduction

- **Radial Basis Funtion** (RBF) neural networks have their roots at exact function interpolation (the formulation as a neural network came later)

- The output of a hidden neuron is determined by the **distance** between the input and the neuron's center (seen as a **prototype**)

- This latter fact has two important consequences:

  1. It allows to give a precise interpretation to the network output

  2. It allows to design de-coupled training algorithms

Artificial neural networks: the RBFNN

Introduction

- Exact function interpolation:

$$h(\boldsymbol{x}_n) = t_n \qquad \boldsymbol{x}_n \in \mathbb{R}^d, t_n \in \mathbb{R},\ n = 1, ..., N$$

- The function h is expressed as a combination of **basis functions**:

$$\phi_n(\boldsymbol{x}) := \phi(\|\boldsymbol{x} - \boldsymbol{x}_n\|)$$

Artificial neural networks: the RBFNN

Introduction

- The combination is linear w.r.t. the basis functions:

$$h(\boldsymbol{x}) = \sum_{n=1}^{N} w_n \phi_n(\boldsymbol{x}) = \sum_{n=1}^{N} w_n \phi(\|\boldsymbol{x} - \boldsymbol{x}_n\|)$$

  which we will force to be exact for all the data points: $h(\boldsymbol{x}_n) = t_n$

- The function $\|\cdot\|$ is any norm in $\mathbb{R}^d$ (most often an **Euclidean norm**)

- Because of the norm, the ϕ_n are functions that exhibit **radial** contours of constant value **centered** at the data points $\boldsymbol{x}_n$

Artificial neural networks: the RBFNN

Introduction

In matrix notation:

$$\begin{pmatrix} \phi_1(\boldsymbol{x}_1) & \phi_2(\boldsymbol{x}_1) & \cdots & \phi_N(\boldsymbol{x}_1) \\ \phi_1(\boldsymbol{x}_2) & \phi_2(\boldsymbol{x}_2) & \cdots & \phi_N(\boldsymbol{x}_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\boldsymbol{x}_N) & \phi_2(\boldsymbol{x}_N) & \cdots & \phi_N(\boldsymbol{x}_N) \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ \vdots \\ t_N \end{pmatrix}$$

$$\boldsymbol{\Phi w} = \boldsymbol{t}$$

Note that the matrix Φ is $N \times N$ and symmetric (if all ϕ_i have a single common width parameter).

Artificial neural networks: the RBFNN

Introduction

- Assuming that Φ is non-singular, $\boldsymbol{w}$ can be found as $\boldsymbol{w} = \Phi^{-1}\boldsymbol{t}$ (*e.g.*, using LU decomposition: $\Phi = LU$ where L is lower triangular and U is upper triangular)

- It can be shown that indeed Φ is non-singular for various choices of the basis functions (**Micchelli's theorem**), including:

  1. $\phi(z) = \exp(-z^2/\sigma^2)$
  2. $\phi(z) = (z^2 + \sigma^2)^\alpha, \ \alpha \in (-\infty, 0) \cup (0, 1)$
  3. $\phi(z) = z^3$
  4. $\phi(z) = z^2 \ln z$

Artificial neural networks: the RBFNN

Introduction

- If the interpolation problem has codomain in $\mathbb{R}^m$ (i.e., $t_n \in \mathbb{R}^m$), the generalization is straightforward:

$$h_k(\boldsymbol{x}) = \sum_{n=1}^{N} w_{kn}\phi_n(\boldsymbol{x}) = \sum_{n=1}^{N} w_{kn}\phi(\|\boldsymbol{x} - \boldsymbol{x}_n\|),\ 1 \leq k \leq m$$

  that we will force to be exact for all the data points: $h_k(\boldsymbol{x}_n) = t_{nk}$

- This problem leads to $\Phi W = T$, solved again by simple matrix inversion as $W = \Phi^{-1}T$

Note the dimensions: Φ is $N \times N$, but W, T are $N \times m$

Artificial neural networks: the RBFNN

Regularization

- Very often, in ML, the exact function interpolation setting is not attractive at all!

  1. High number (N) of interpolation points $\rightarrow$ complex and unstable solutions

  2. The outputs t_n depend stochastically on the inputs $\boldsymbol{x}_n$ $\rightarrow$ overfit solutions

  3. The interpolation matrix Φ can be singular or ill-conditioned

  4. The inversion of Φ grows as $O(N^3)$
     (for symmetric PD matrices, Cholesky decomposition takes some $N^3/3$ steps)

- We are in need of a tighter control of complexity of the solution

Artificial neural networks: the RBFNN

Regularization

- In one-hidden-layer neural networks, **regularization** penalizes the size of the weight matrix:

$$E_{emp}(W) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{m} (t_{nk} - h_k(\boldsymbol{x}_n))^2 + \frac{\lambda}{2} \sum_{k=1}^{m} \|\boldsymbol{w}_k\|^2$$

  which results in $W = (\Phi + \lambda I_N)^{-1} T$; the value of $\lambda > 0$ is proportional to the amount of noise in the data

- Another way of obtaining much simpler solutions is to use a **subset** of the data points to center the basis functions; more generally, they can be centered at a carefully selected set of points in $\mathbb{R}^d$

Artificial neural networks: the RBFNN

RBF networks

With these modifications, we obtain the so-called RBF network:

$$h_k(\boldsymbol{x}) = \sum_{i=0}^{H} w_{ki}\phi_i(\boldsymbol{x}) = w_{k0} + \sum_{i=1}^{H} w_{ki}\phi(\|\boldsymbol{x} - \boldsymbol{c}_i\|), \;\; 1 \leq k \leq m$$

which is a **two-layer neural network**:

1. The first (hidden) layer of $H \ll N$ neurons computes the basis functions $\phi_i(\boldsymbol{x})$, centered at the vectors $\boldsymbol{c}_i$

2. A constant basis function $\phi_0(\boldsymbol{x}) = 1$ is included to be associated with the bias weights w_{k0} in the output layer

3. The second (output) layer implements the linear combinations of basis functions

Artificial neural networks: the RBFNN

RBF networks

A very popular choice for the ϕ_i is a simple Gaussian:

$$\phi_i(\boldsymbol{x}) = \exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{c}_i\|^2}{\sigma_i^2}\right)$$

- The new matrix $\Phi_{N\times(H+1)}$, is sometimes known as the **design** matrix; now the weight matrix is $W = (\Phi^\top\Phi)^{-1}\Phi^\top T$

- If the original $\Phi_{N\times N}$ matrix was non-singular, then the matrix $\Phi_{N\times(H+1)}$ is also non-singular (very important result!)

If we also regularize the solution, then $W = (\Phi^\top\Phi + \lambda I_{H+1})^{-1}\Phi^\top T$

Artificial neural networks: the RBFNN

In summary

RBF network training is typically performed in a decoupled way:

1. The first stage finds $H, \{\boldsymbol{c}_i\}, \{\sigma_i^2\}$ using a **clustering** algorithm

2. The second stage finds W by any of the usual (linear) methods:

   - Using the pseudo-inverse (via the SVD), for **regression** (linear output activations)

   - Using logistic regression, for **binary classification** (logistic output activations)

   - Using multinomial regression, for **multiclass classification** (softmax output activations)

Artificial neural networks: the RBFNN

Comparison to the MLP (I)

- MLPs perform a global and distributed approximation of the underlying function, whereas RBFNN perform a local and non-distributed one

- The distributed representation of MLPs causes the error surface to have multiple local minima

- Training times for MLPs are usually orders of magnitude larger than those for RBFNNs

- MLPs generalize better than RBFNNs in regions of input space outside of the local neighborhoods defined by the training set*

*On the other hand, extrapolation far from training data is oftentimes unjustified and risky.

Artificial neural networks: the RBFNN

Comparison to the MLP (II)

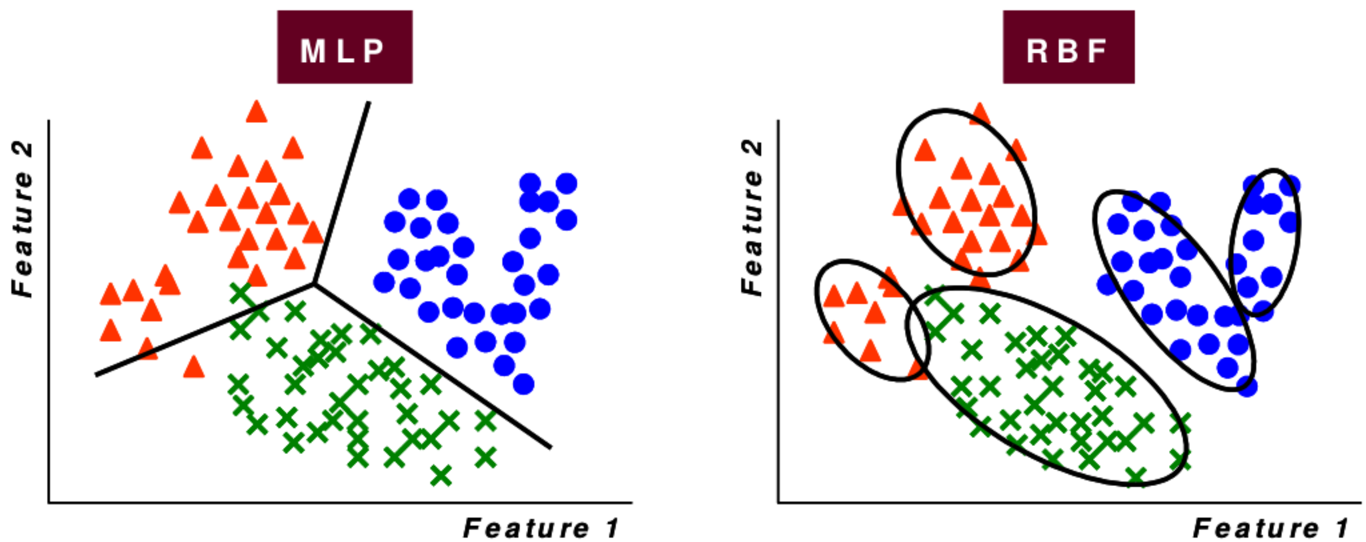
- MLPs typically require fewer neurons than RBFNNs to approximate a non- linear function with the same accuracy

- All the parameters in an MLP are trained simultaneously; parameters in the hidden and output layers of an RBFNN network are typically trained separately using very efficient hybrid algorithms

- MLPs may have multiple hidden layers with complex connectivity, whereas RBFNNs typically have only one hidden layer and full connectivity

Artificial neural networks: the RBFNN

Comparison to the MLP (III)

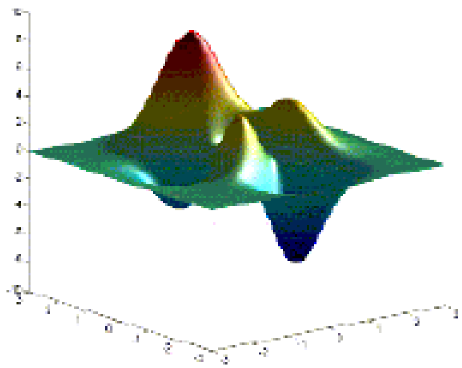


- The hidden neurons of an MLP compute the inner product between an input vector and their weight vector; RBFNNs compute the Euclidean distance between an input vector and the RBF centers
- MLP partition feature space with hyper-planes; in RBFNN, constant activation boundaries of hidden units are hyper-ellipsoids (usually hyper-spheres)

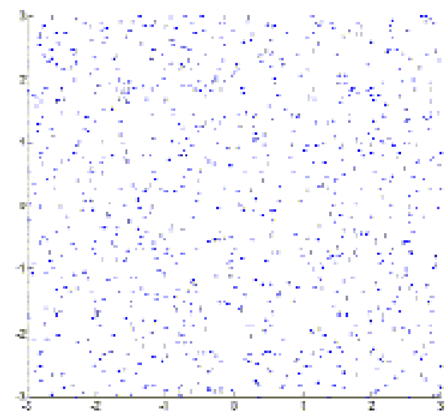
Picture credit: R. Gutiérrez-Osuna

Artificial neural networks: the RBFNN

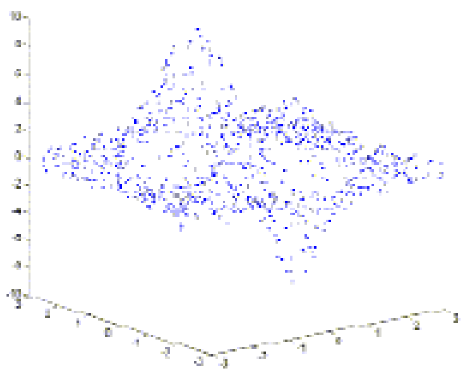
An example



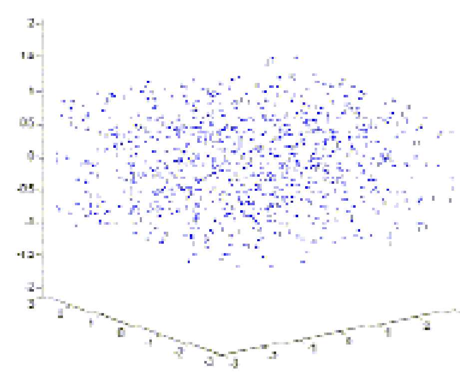
a: Deterministic function



b: Uniform distribution of data points



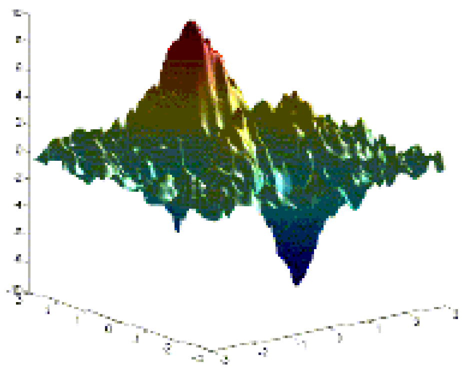
c: The data sample with noise



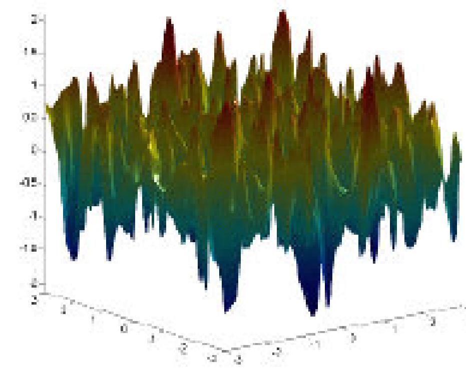
d: The U(-1,1) noise component

Artificial neural networks: the RBFNN

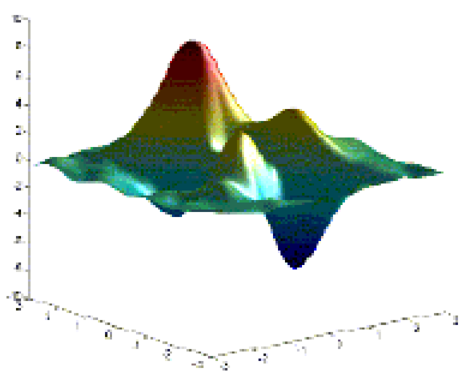
Example



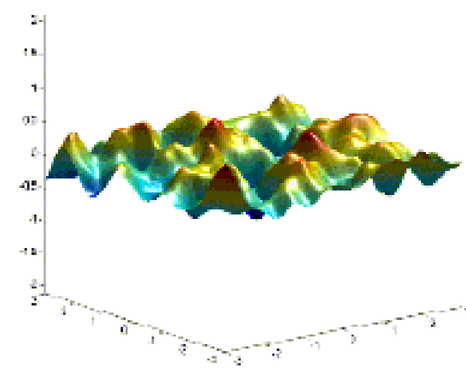
e: Exact fit to data points in (c)



f: (e)-(a), i.e., exactly fitting the data in (d)



g: Approximating RBF



h: (g)-(a)