INTERNATIONAL COMPUTER SCIENCE INSTITUTE

1947 Center St. • Suite 600 • Berkeley, California 94704-1198 • (510) 643-9153 • FAX (510) 643-7684

Learning from data: general issues and special applications of Radial Basis Function networks

A. Baraldi*, N. A. Borghese†

TR-98-028

August 1998

Abstract

In the first part of this work some important issues regarding the use of data-driven learning systems are discussed. Next, a special category of learning systems known as artificial Neural Networks (NNs) is presented. Our attention is focused on a specific class of NNs, termed Radial Basis Function (RBF) networks, which are widely employed in classification and function regression tasks. A constructive RBF network, termed Hierarchical RBF (HRBF) model, is proposed. An application where the HRBF model is applied to reconstruct a continuous 3-D surface from range data samples is presented.

Key words: Inductive and deductive types of inference, learning from data, predictive learning, supervised and unsupervised learning, actual risk and empirical risk, curse of dimensionality, basis function, kernel function, neural networks, Multi-Layer-Perceptron, Radial Basis Function network, data-driven and error-driven learning, hybrid learning, one- and two-stage learning, grid-partitioning and scatter-partitioning network, constructive network, Hierarchical Radial Basis Function network.

*International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA 94704-1198, Ph.: +1+510+643-9153, Fx.: +1+510+643-7684, baraldi@icsi.berkeley.edu

†Laboratory of Human Motion Study and Virtual Reality Istituto Neuroscienze Bioimmagini - C.N.R. via f.lli Cervi 93 - 20090 Segrate (Milano), Italy Ph.: +39+2+; Fax: +39+2+, borghese@inb.mi.cnr.it

1 Learning from data

Modern science describes physical, biological and social systems in terms of *first-principle* models (e.g., Newton's laws of mechanics) to be verified on the basis of experimental data [1]. However, in many applications the underlying first principles are unknown or the systems under study are too complex to be described analytically. In the absence of first principles, experimental data can be used to derive unknown (input, output) dependencies between a system's variables. In recent years the use of low-cost sensors and computers has increased the availability and usability of large experimental data sets. Thus, there has been a shift from classical modelling, based on first principles, to the development of models from data [1]. The latter approach is known as data-driven learning [1], or learning-by-examples [2]. In humans, this type of learning mechanism is involved, for example, with the simplest cognitive processes in infants and with commonsense reasoning and spontaneous generalization in adults [2].

All data-driven learning systems are based on the following concepts: i) information extracted from a finite set of observed examples, termed *training data set*, can be used to answer questions either about unobserved samples belonging to a so-called *test set*, or about unknown properties hidden in the training data; and ii) the goal of the learning process is to minimize a risk functional (theoretically computed over an infinite data set, see Section 2.1) by adapting system parameters on the basis of the finite training set.

Training data belong to two main categories: i) *labeled data*, consisting of known (input, output) vector pairs sampled from an unknown (input, output) mapping between two multidimensional spaces; and ii) *unlabeled data* consisting of observed input samples exclusively. In the first case, the training set is termed *supervised* because the value of the desired output vector is specified for each input vector. In the latter case, the training set is termed *unsupervised*. A supervised training set is described as a set of multidimensional vector pairs, $(\mathbf{x_h}, \mathbf{y_h})$, $h = 1, ..., M$, where M is the finite size of the training set, input sample $\mathbf{x_h} \in R^{n_i}$, output sample $\mathbf{y_h} \in R^{n_o}$, and n_i and n_0, belonging to $\mathcal{I}^+$, are the dimensionality of the input and output space respectively.

Data-driven learning tasks include the following:

1. *Predictive learning* tasks. Whereas *deductive learning* provides special cases (e.g., output values) from general models, the goal of predictive learning, also called *inductive learning* or *specific-to-general type of inference*, is to provide good generalization (i.e., a good model) from a finite set of particular labeled cases (supervised training examples) [1]. In detail, on the basis of the supervised training set, a predictive learning system selects the approximating function that minimizes a cost function for unobserved (future) samples from a wide class of candidated functions [1]. Two classes of predictive learning tasks are:

   (a) *Function regression*, where the task is to estimate an unknown continuous function, capable of giving good predictions for unobserved data, from eventually noisy supervised training samples. When training examples are not affected by noise, regression problems are called *function interpolation* problems [1]. When the mapping between two multidimensional spaces is viewed as a function defined over the multidimensional input space R^{n_i}, then the problem of function

regression becomes equivalent to finding a surface providing the best fit to the training data points in the multidimensional output space R^{n_o}, according to an optimization criterion.

(b) *Classification (pattern recognition)*, where the task is to assign new inputs to one of a finite number of discrete output values, equivalent to output classes or categories, on the basis of the supervised training set.

2. Probability density function estimation from unsupervised training samples.

3. Clustering/vector requantization from unsupervised training samples, where clusters are intended as "natural" structures or statistical regularities capable of characterizing the unsupervised training set.

In the first part of this work some important issues regarding the use of data-driven learning systems are discussed. Next, a particular category of learning systems known as artificial Neural Networks (NNs) is presented. Our attention is focused on a specific class of NNs, termed Radial Basis Function (RBF) networks, which are widely employed in classification and function regression tasks. A constructive RBF network, termed Hierarchical RBF (HRBF) model, is presented. An application where the HRBF model is applied to reconstruct a continuous 3-D surface from range data samples is proposed [3].

2 Issues in learning from data

This section briefly discusses some important issues regarding the use of data-driven learning systems.

2.1 Actual risk and empirical risk

In Section 1, an inductive learning machine is described as an adaptive system whose goal is to select, from a set of candidated functions, the estimated function that best approximates the experimental responses of the system under study on the basis of a training set of observed examples. Let us identify as $\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})$ the class of function estimators supported by the learning machine, where $\mathbf{w}$ identifies the vector of adjustable parameters to be determined on the basis of the training data. Under the hypothesis that observed (input, output) samples are independent and identically distributed (i.i.d), the supervised training data set can be described by the joint probability density function

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{x}|\mathbf{y}). \tag{1}$$

For any measured pair $(\mathbf{x_h}, \mathbf{y_h})$, $h = 1, ...M$, the quality of the approximating function $\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})$ is measured by the loss or discrepancy function $L(\mathbf{y_h}, \hat{\mathbf{f}}(\mathbf{w}, \mathbf{x_h}))$. The expected value of the loss function is called *actual (or true or expected or prediction) risk functional*

$$R_{act}(\mathbf{w}) = \int L(\mathbf{y}, \hat{\mathbf{f}}(\mathbf{w}, \mathbf{x}))p(\mathbf{x}, \mathbf{y})d\mathbf{x}d\mathbf{y}. \tag{2}$$

Learning is the process of estimating, based only on the training data, the vector of parameters $\mathbf{w}$ *such that the estimated function* $\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})$ *minimizes the actual risk functional*

$R_{act}(\mathbf{w})$. The problem is that since $p(\mathbf{x}, \mathbf{y})$ is unknown, then $R_{act}(\mathbf{w})$ is also unknown. This is tantamount to saying that *the finite number of training samples implies that any estimate of the (unknown) actual risk function is inaccurate (biased), i.e., any predictive learning problem is ill-posed.* The straightforward approach is to minimize the *empirical risk* or *sample error* computed over the finite samples belonging to the supervised training set as a substitute for unknown expected risk. The empirical risk is

$$R_{emp}(\mathbf{w}) = \frac{1}{M} \sum_{h=1}^{M} L(\mathbf{y_h}, \hat{\mathbf{f}}(\mathbf{w}, \mathbf{x_h})). \tag{3}$$

Unfortunately, empirical risk minimization based on a finite training set does not guarantee a small actual risk. This is evident when we consider that the solution that minimizes R_{emp} is not unique as there is an infinite number of continuous functions that can interpolate M data points exactly. In other words, the predictive learning problem is ill-posed in the absence of any assumption about the nature of the function to be estimated. The key observation is that meaningful estimation on the basis of a finite data set is possible only for sufficiently smooth target functions. This is tantamount to saying that any inductive learning process is based on an *inductive principle* or *inference method*, which is a framework for selecting a unique solution from a wide class of candidated functions using finite data. An inductive principle combines the evidence provided by the training data with smoothness constraints considered as *a priori* assumptions (i.e., *background knowledge independent of the training data*) about the learning process at hand. Note that the inductive principle and the optimization procedure required by any learning system feature two different and complementary roles: while the inductive principle identifies the class of cost functions to be minimized, the optimization procedure specifies how to estimate parameters to minimize the given cost function.

To improve the performance of learning systems in minimizing the actual risk on the basis of a limited amount of data, quantitative bounds and equations relating the actual risk to the empirical risk, model complexity and the number of training examples have been investigated in recent years [1], [4].

2.2 Inductive principles

The complexity of a learning system increases with the number of independent and adjustable parameters, also termed *degrees of freedom*, to be adapted during the learning process. According to the qualitative principle of *Occam's razor*, a sound basis for generalizing beyond a given set of examples is to prefer the simplest hypothesis that fits observed data [4], [6]. This principle states that to be effective, the cost function minimized by an inductive learning system should provide a trade-off between how well the model fits the training data and *model complexity.* This also means that model complexity must be controlled by *a priori* (background) knowledge, i.e., *subjective knowledge available before any evidence (e.g., empirical risk) provided by the training data is observed.*

Different inductive principles provide cost functions considered as different quantitative formulations of Occam's qualitative principle. For example, when the supervised training data are affected by noise, exact interpolation functions tend to become highly oscillatory. Since they are unlikely to provide good predictions for new examples, highly oscillatory

functions are considered neither useful nor desirable in function regression. To restrict possible solutions, one common assumption is to add a penalty (regularization) term based on *a priori* knowledge to the empirical risk to be minimized. Then, according to the *penalization (regularization) inductive principle*, we have

$$R_{pen}(\mathbf{w}) = R_{emp}(\mathbf{w}) + \lambda\omega[\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})], \tag{4}$$

where $\lambda > 0$ is a regularization parameter that controls the strength of the *a priori* knowledge included in the form of the penalty functional $\omega[\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})]$. To avoid the risk of *overfitting* training data, the penalty term should be chosen as a non-decreasing function of model complexity in such a way that minimization of Eq. (4) is equivalent to minimizing actual risk (2). In this case minimization of Eq. (4) provides a trade-off between fitting training data to minimize the empirical risk, and fitting background knowledge to minimize model complexity. This is tantamount to saying that the choice of λ and $\omega[\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})]$ is a problem of *model selection*, i.e., a task in which a model of optimal complexity is chosen for the given finite data set [1]. For example, in function regression tasks, the regularization term may penalize candidated functions that are not smooth, i.e., those presenting large oscillations, by computing [6]

$$\omega[\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})] = \int \left(\frac{d^2 \hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})}{d\mathbf{x}^2} \right)^2 d\mathbf{x}. \tag{5}$$

Under the classical Bayesian approach, both λ and $\omega[\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})]$ are chosen on the basis of *a priori* knowledge exclusively, which is to say that by definition the observed data are not used for model selection. As correct specification of priors is difficult to accomplish in practice, predictive learning systems become more robust and flexible if the regularization parameter λ is chosen on the basis of the training set (for example, by means of *cross-validation* [1], [5]) for a given type of user-defined penalty functional $\omega[\hat{\mathbf{f}}(\mathbf{w}, \mathbf{x})]$.

Several statistical techniques, besides the regularization inductive principle, have proposed different bounds and equations relating actual risk to empirical risk, model complexity and the number of training examples within the framework of predictive learning. For example, it has been shown that in the limit of an infinite data set, a cost function computed as the sum-of-squared differences between target and estimated outputs is proportional to the sum of a bias term with a variance term [6], [8], i.e.,

$$R_{act}(\mathbf{w}) \propto [\text{bias}(\mathbf{w})]^2 + \text{variance}(\mathbf{w}). \tag{6}$$

In Eq. (6) the bias term measures the extent to which the average (over an infinite data set) of the approximating function differs from the target function. For example, an approximating function that is closely fitted to training data (e.g., that exactly interpolates them) will tend to have a small bias. Conversely, in Eq. (6) the variance term measures the extent to which the approximating function is sensitive to the particular choice of the data set [6], [7]. For example, an approximating function that is closely fitted to training data will tend to have a large variance. Thus, minimization of Eq. (6) provides a natural trade-off between minimizing both bias and variance: by smoothing the approximating function we can decrease the variance, but if the smoothing is taken too far then the bias becomes large. This quantitative principle is called the *bias-variance trade-off* [8].

Additional quantitative properties of predictive learning systems have been proposed by statistical learning theory, also known as the Vapnik-Chervonenkis (VC) theory, in the last 25 years [9]. For example, in the case of two-class (binary) pattern recognition problems, the bound

$$R_{act}(\mathbf{w}) \leq R_{emp}(\mathbf{w}) + \phi\left[\frac{h}{M}\right], \tag{7}$$

holds if parameter h is finite. This parameter, known as the *VC dimension*, is a characteristic of the set of approximating functions supported by the learning system and is a measure of model complexity [1], [10]. In Eq. (7), function $\phi[\frac{h}{M}]$, called the *VC confidence term or confidence interval*, decreases monotonically with size M of the training set and monotonically increases with h. Eq. (7) means that if M is large, then the value of the confidence term becomes small and the empirical risk can be safely employed as an estimate of the true risk. Vice versa, when M is small, Eq. (7) shows that to minimize the actual risk the combination of the empirical risk with the confidence term ought to be minimized, i.e., the VC-dimension h should also be minimized [9], [10].

For function regression problems VC theory provides the following bound, which is useful for model selection [1]

$$\frac{h}{M} \leq 0.8 \quad \text{for } \eta \geq \min\left\{\frac{4}{\sqrt{M}}, 1\right\}, \tag{8}$$

where $(1-\eta) \in [0,1]$ is called the confidence level. Eq. (8) provides an upper limit for model complexity for a given sample size and confidence level or, equivalently, a lower limit for the sample size for a given model complexity and confidence level.

2.3 Semiparametric models

To model training data, *parametric data-driven learning algorithms* employ specific functional forms selected by the application developer on a *a priori* basis (i.e., before examining the data). One typical example of a parametric learning system is the normally-distributed density function estimator. Parametric learning systems employ simple learning procedures, but lack robustness.

Non-parametric learning systems, such as kernel-based methods for density function estimation, are not restricted to specific functional forms. Like parametric learning systems, they employ simple learning procedures, but their complexity increases with the size of the training set.

Learning systems where the problem is to estimate both optimal model complexity and model parameters are called *semiparametric* [1], [6]. Unlike parametric learning algorithms, semiparametric systems are not restricted to specific functional forms. Moreover, the complexity of semiparametric systems must increase with the complexity of the problem being solved rather than with the size of the training set as for non-parametric systems. The disadvantage of semiparametric learning systems is that they are computationally intensive compared to the simple procedures required by both parametric and non-parametric systems [6]. One typical example of a semiparametric learning system is the mixture distribution model for density function estimation [6].

A special case of semiparametric systems is the class of *constructive procedures*, also called *growing algorithms*, where model complexity increases dynamically and the choice

among candidated solutions is based on their performance over the entire collection of training examples [1], [4].

2.4 Curse of dimensionality

In previous sections it has been shown why, in predictive learning, meaningful function estimation is possible only for sufficiently smooth target functions. A smoothness constraint essentially defines possible function behaviors in local neighborhoods of the input space [1]. It is obvious that the accuracy of function estimation depends on having enough training samples within the local neighborhood specified by a smoothness constraint in the input space. Unfortunately, the number of samples yielding the same density increases exponentially with the dimensionality of the input space [1], [6]. This effect is known as the "curse of dimensionality."

As an example of the "curse of dimensionality," consider that any function estimator increases its number of adjustable parameters with the dimensionality of input space. As a consequence, the size of training data required to compute a reliable estimate of adaptive parameters may become huge in practical problems. A specific relationship between sample size and model complexity is shown in Eq. (8). A heuristic rule between sample size and model complexity requires sample size to be at least 10 times the number of free parameters in the model [16].

2.5 Classification of methods for regression

One possible way to classify methods for estimating continuous-valued functions from noisy samples is to develop a taxonomy based on *dictionary representation* versus *kernel representation* [1].

In dictionary representation, a set of parameterized candidated functions is expressed as a *weighted combination of basis functions*

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{j=1}^{C} \mathbf{w_j} g_j(\mathbf{x}, \mathbf{v_j}), \tag{9}$$

where $g_j(\mathbf{x}, \mathbf{v_j})$ is called *Basis Function* (BF) and vectors $\mathbf{w} = [\mathbf{w_1}, ..., \mathbf{w_C}]$ and $\mathbf{v} = [\mathbf{v_1}, ..., \mathbf{v_C}]$ consist of adaptive parameters. The set of BFs, $g_j(\mathbf{x}, \mathbf{v_j})$, $j = 1, ..., C$, is called a *dictionary*, and the number C of dictionary entries (BFs) is often considered a regularization (complexity) parameter. The goal of a predictive learning system that employs Eq. (9) for function regression is to adjust degrees of freedom $\mathbf{w}$, $\mathbf{v}$ and C on the basis of a (finite) supervised training set in such a way that the approximating function, selected from the set of candidated functions, provides minimum prediction risk (2). In other words, *Eq. (9) is a function estimator designed to perform well over the entire instance (input) space, i.e., model selection with a dictionary representation is global because parameters are adjusted by a learning procedure on the basis of the entire training set, and "permanent" because, after completion of the learning phase, training examples can be dismissed.* If basis functions are fixed, parameterization (9) becomes linear with respect to parameters $\mathbf{w}$ which can be estimated from the training set by means of linear least squares methods [1], [6].

In kernel representation, a continuous-valued function estimator is expressed as a *distance-weighted combination of observed output values*

$$\hat{\mathbf{y}}(\mathbf{x}) = \sum_{h=1}^{M} \mathbf{y_h} k_h(d(\mathbf{x}, \mathbf{x_h})), \tag{10}$$

where (input, output) pairs $(\mathbf{x_h}, \mathbf{y_h})$, $h = 1, ..., M$, are the observed examples that make up the supervised training set. Unlike Eq. (9), whose computational complexity increases with regularization parameter C, note that the computational complexity of Eq. (10) increases with the size of the training set. In Eq. (10) weighting function $k_h(d(\mathbf{x}, \mathbf{x_h}))$, called *kernel function*, is monotonically non-increasing with distance $d(\mathbf{x}, \mathbf{x_h})$. For example, distance $d(\mathbf{x}, \mathbf{x_h})$ can be computed according to the L_p norm (Minkowsi metric) which, in a general multidimensional case, is

$$d(\mathbf{a}, \mathbf{b}) = (\sum_{q=1}^{n_i} |a_q - b_q|^p)^{\frac{1}{p}}, \tag{11}$$

where $\mathbf{a}$ and $\mathbf{b}$ are two points belonging to the input space whose dimensionality is $n_i \geq 1$. If $p = 2$, Eq. (11) provides the Euclidean distance. Kernel function $k(d(\mathbf{a}, \mathbf{b}))$ usually (but not always) satisfies the following properties [1]

$k(d(\mathbf{a}, \mathbf{b})) \geq 0.$	Nonnegative.	$(12a)$
$k(d(\mathbf{a}, \mathbf{b})) = k(d(\mathbf{b}, \mathbf{a})).$	Radially symmetric.	$(12b)$
$k(d(\mathbf{a}, \mathbf{a})) =$ max.	Takes on its maximum when $\mathbf{a} = \mathbf{b}$.	$(12c)$
$\lim_{t \to \infty} k(t) = 0.$	Localized function [6].	$(12d)$

Kernel methods are examples of *instance-* or *memory-based learning strategies* because they are based on storing all training data [4], [11]. The basic idea behind kernel representation is that when a new query instance is presented, the target value is estimated only on the basis of training examples near the query point, as training examples are individually weighted by their distance from the query point. This also means that unlike Eq. (9), which is designed to perform well over the entire instance space, instance-based learning approaches actually construct a different approximation to the target function for each distinct query instance [4]. For the sake of simplicity, let us consider a mapping between two 1-D spaces, i.e., $x_h \in \mathcal{R}$, $y_h \in \mathcal{R}$, $h = 1, ..., M$. When unobserved function values are estimated by fitting the nearby training points well, with less concern for distant training points, the cost function to be minimized on the basis of the supervised training set becomes

$$C(q) = \sum_{h=1}^{M} (\hat{y}(q) - y_h)^2 k(d(q, x_h)), \tag{13}$$

where q is a query input point (unobserved example) and $k(d(q, x_h)]$ is a distance-weighting function constrained by Eq. (12). The best estimate $\hat{y}(q)$ is the one that minimizes $C(q)$ such that

$$\frac{\partial C(q)}{\partial \hat{y}(q)} = \sum_{h=1}^{M} (\hat{y}(q) - y_h) k(d(q, x_h)) = 0,$$

therefore,

$$\hat{y}(q) = \frac{\sum_{h=1}^{M} y_h k(d(q, x_h))}{\sum_{h=1}^{M} k(d(q, x_h))}. \tag{14}$$

According to Eq. (14), if $k(d(q,x_h)) \rightarrow \infty$ when $d(q,x_h) \rightarrow 0$, i.e., when $q \rightarrow x_h$, then $\hat{y}(q) \rightarrow y_h$, i.e., exact interpolation of the supervised training data is pursued. If data are noisy, exact interpolation is not desirable, and Eq. (14) should employ a weighting scheme based on finite values of the kernel function to guarantee smooth interpolation between training points [11]. For example, let us consider the case in which Eq. (14) employs a Gaussian weighting function

$$k(d(q,x_h)) = e^{-\frac{(d(q,x_h))^2}{2\sigma_w}}. \tag{15}$$

In this case, if $d(q,x_h) \rightarrow 0$, i.e., if $q \rightarrow x_h$, then $k(d(q,x_h)) \rightarrow 1$ and, as a consequence, $\hat{y}(q) \neq y_h$. This means that when Eq. (14) employs a Gaussian weighting function then no exact interpolation is pursued because not every observed input value is mapped exactly onto its corresponding target value.

It is important to stress that a clear distinction between dictionary and kernel methods as proposed in [1] is rather obscure in the rest of the literature where the term "kernel function" is often employed to identify localized basis functions in dictionary methods and density function estimators.

2.6 Orthogonal basis functions

An interesting case where the computation of the weights of a dictionary representation (9) is significantly simplified occurs when the function estimator employs a set of orthonormal basis functions. Basis functions $g_j(\mathbf{x})$, $j = 1, ..., C$, are orthonormal if

$$\int g_i(\mathbf{x}) g_j(\mathbf{x}) dx = \delta_{i,j}, \quad \forall i,j \in \{1, C\}, \tag{16}$$

where $\delta_{i,j}$ is the Kronecker delta, such that $\delta_{i,j} = 1$ iff $i = j$ and $\delta_{i,j} = 0$ otherwise. If an orthonormal basis is employed in Eq. (9), and equally-spaced samples are collected in the supervised training set, then minimization of the sum-of-squares error with respect to weight $\mathbf{w_i}$, $i \in \{1, C\}$, yields [1]

$$\frac{\partial R(\mathbf{w})}{\partial \mathbf{w_i}} = \frac{\partial \int \left[\mathbf{y}(\mathbf{x}) - \sum_{j=1}^{C} \mathbf{w_j} \mathbf{g_j}(\mathbf{x})\right]^2 dx}{\partial \mathbf{w_i}} = -2 \int \left[\mathbf{y}(\mathbf{x}) - \sum_{j=1}^{C} \mathbf{w_j} \mathbf{g_j}(\mathbf{x})\right] \mathbf{g_i}(\mathbf{x}) dx =$$

$$-2 \int \mathbf{y}(\mathbf{x}) \mathbf{g_i}(\mathbf{x}) dx + 2 \sum_{j=1}^{C} \mathbf{w_j} \int \mathbf{g_j}(\mathbf{x}) \mathbf{g_i}(\mathbf{x}) dx = -2 \int \mathbf{y}(\mathbf{x}) \mathbf{g_i}(\mathbf{x}) dx + 2\mathbf{w_i} = 0,$$

i.e.,

$$\mathbf{w_i} = \int \mathbf{y}(\mathbf{x}) \mathbf{g_i}(\mathbf{x}) dx. \tag{17}$$

Eq. (17) shows that in *signal processing, where the goal is to find a compact and accurate representation of a known signal*, analysis of the weights employed in the linear combination (9) becomes simple when: i) a fixed set of orthonormal basis is employed; and ii) equally-spaced samples are collected. When these conditions are satisfied the signal-specific weighting coefficients $\mathbf{w_j}$, $j = 1, ..., C$, define uniquely signal $\mathbf{y}(\mathbf{x})$) [12], [13]. However, our interest is focused on estimating an unknown continuous-valued signal from noisy samples.

When the target function is unknown, Eq. (17) cannot be employed directly but must be estimated from the training set of equally-spaced (input, output) examples, i.e.,

$$\mathbf{w_i} = \frac{1}{M} \sum_{h=1}^{M} \mathbf{y_h g_i}(\mathbf{x_h}). \tag{18}$$

Examples of an orthonormal basis include Fourier series, where a stationary signal is synthesized (reconstructed) as a combination of sinusoidal waveforms, Hermite polynomials employed in polynomial transforms [14], and wavelets which are employed, for example, in 1-D and 2-D signal compression [15].

3 Introduction to neural networks

Artificial Neural Networks (NNs) are parametrized and distributed systems capable of learning from data. In this context the term distributed means that NNs consist of a multeplicity of simple and mutually interconnected processing elements. Owing to these properties NNs are also termed complex systems [2]. The role of NNs is to determine (input, output) dependencies that are unknown, or too complex to be described analytically, on the basis of a finite training set of examples.

Since NNs are based on data-driven learning mechanisms, they do not derive their organization from an external design, i.e., the background knowledge involved in NNs does not regard their connectionist architecture as a whole, but only their individual processing elements. Instead, the organization of NNs naturally emerges from elementary interactions of the processing elements while the data set of observed examples is presented to the network. An important consequence of this observation is that, *in the connectionist approach, data-driven learning cannot be studied independently of the physical support of the cognitive system.* This is in contrast with the study of syntactic (symbolic) high-level information processing systems based on learning-by-rule mechanisms [2]. In these systems a set of decision rules representing application-domain specific background knowledge (i.e., independent of training data) may be explicitly taught to the system by an external supervisor. Unlike the connectionist approach, the symbolic approach implies that *high-level cognitive systems can be studied independently of the structure and functioning of their physical support.*

3.1 Advantages and drawbacks of neural networks

Many of the important issues concerning the application of NNs can be introduced by considering the simpler case of polynomial estimators [6]. Let us consider a set of M (input, output) data pairs $(\mathbf{x_h}, y_h)$, $h = 1, ..., M$, where $\mathbf{x_h} = (x_{h,1}, ..., x_{h,n_i}) \in R^{n_i}$ and $y_h \in R$. The problem is to fit a Pth-order polynomial to the given set of multidimensional data points so that a given risk function is minimized. In our example, the Pth-order polynomial has the form

$$y(\mathbf{x}) = w_0 + \sum_{i_1=1}^{n_i} w_{i_1} x_{i_1} + \sum_{i_1=1}^{n_i} \sum_{i_2=1}^{n_i} w_{i_1,i_2} x_{i_1} x_{i_2} + ... + \sum_{i_1=1}^{n_i} \sum_{i_2=1}^{n_i} ... \sum_{i_P=1}^{n_i} w_{i_1,i_2,...,i_P} x_{i_1} x_{i_2} ... x_{i_P}. \tag{19}$$

Eq. (19) is an example of a *linear model*, i.e., a learning system supporting candidated functions that depend linearly on the free parameter vector $\mathbf{w}$, although they may not depend linearly on input variables $\mathbf{x}$ [6]. In the Pth-order polynomial the number of degrees of freedom grows as n_i^P, which represents a dramatic growth in the complexity of the model as the dimensionality of the input space, n_i, increases. This implies that the number of data points, M, required to well-determine the adaptive parameters of a linear model also increases dramatically with the number n_i of input variables.

The *problem of scaling with dimensionality* is efficiently solved by non-linear NNs where non-linear functions of adjustable parameters approximate general (non-linear) mappings between multidimensional spaces. The key idea of non-linear NNs is to estimate an approximating function as a superposition of non-linear 'hidden functions' of adjustable parameters that are adapted to the data during the training process, so that the number of such hidden functions increases with the complexity of the mapping rather than with the dimensionality of the input space [6]. As a consequence, the number of free parameters in non-linear NN models typically increases linearly, or quadratically, with the number of input variables n_i [6].

To provide efficient scaling between model complexity and dimensionality of the input space, non-linear NN models exploit two common properties of real data. First, input variables tend to be correlated in some way, i.e., data points tend to be located in some specific regions of the input space and/or in some sub-spaces featuring a so-called true or intrinsic dimensionality which is usually much lower than n_i. Second, in many real problems output variables vary smoothly as the input variables change by small quantities. Thus, non-linear NN models usually enforce smoothness constraints on their candidated solutions.

The main drawback of non-linear NNs is that their optimization is computationally intensive and technically difficult because of the presence of multiple local minima in the error function.

The two major classes of non-linear NN models, Multi-Layer Perceptrons (MLPs) and Radial Basis Function (RBF) networks, are further discussed in the next sections.

3.2 Comparison between MLPs and RBF networks

MLPs and RBF networks are feed-forward networks employed in predictive learning tasks. Their approximating function is

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}, \mathbf{v}) = \sum_{j=1}^{C} \mathbf{w_j} g_j(\mathbf{x}, \mathbf{v_j}) + \mathbf{w_0}, \tag{20}$$

where $g_j(\mathbf{x}, \mathbf{v_j})$ is a BF (see Eq. (9)) also termed *activation function*, vectors $\mathbf{w}$ and $\mathbf{v}$ consist of adaptive parameters, and $\mathbf{w_0}$ is a bias term.

Although MLPs and RBF networks play similar roles, their architectural properties as well as their supervised learning techniques are quite different.

In MLPs [6]:

1. The feed-forward network architecture consists of up to three layers of processing units, termed input, hidden and output layer respectively. Hidden layers are all the layers between the input and output layer.

2. BFs are non-linear functions of the scalar product between input vector $\mathbf{x}$ and weight vector $\mathbf{v_j}$

$$g_j(\mathbf{x}, \mathbf{v_j}) = \mathbf{s}\left(v_{j,0} + \sum_{d=1}^{n_i} x_d v_{j,d}\right) = \mathbf{s}(\mathbf{x} \bullet \mathbf{v_j}), \tag{21}$$

where symbol "$\bullet$" identifies the dot product and $\mathbf{s}(\cdot)$ is a non-linear 's-shaped' (*squashing*) function that maps the interval $(-\infty, \infty)$ onto range (0,1) (e.g., logistic function) or (-1,1) (e.g., hyperbolic tangent). For example, the logistic sigmoid function is

$$\mathbf{s}(t) = \frac{1}{1 + e^{-t}}, \quad s(t) \in (0,1), \forall t \in (-\infty, \infty). \tag{22}$$

3. As a consequence of Eqs. (21) and (22), activation values of every hidden unit are significant in large portions of the input domain, i.e., many hidden units typically have significant activations for a given input. This is tantamount to saying that many hidden units will typically contribute to the determination of the network output value for a given input pattern.

4. All parameters of an MLP network are adjusted according to a single-stage global training strategy, termed error back-propagation learning algorithm, which is based on the supervised training set [16].[1]

In RBF networks [6], [17], [18]:

1. The feed-forward network architecture is made up of only two layers. The first hidden layer consists of C processing units while the second output layer linearly combines the activations provided by the hidden layer to form the outputs.

2. BFs are RBFs, i.e., radially symmetric functions (see Eq. (12b)) centered on parameter vector $\mathbf{v_i}$:

$$g_j(\mathbf{x}, \mathbf{v_j}) = g_j(d(\mathbf{x}, \mathbf{v_j})), \tag{23}$$

where $d(\mathbf{x}, \mathbf{v_j})$ is the distance between input pattern $\mathbf{x}$ and *prototype or template vector* $\mathbf{v_j}$. In the input space, prototype vector $\mathbf{v_j}$ identifies the center of the receptive field of processing unit j. The *receptive field* of a processing unit is defined as the subset of the input domain where the processing unit features activation values greater than a small threshold $0 < \epsilon << 1$. An RBF whose output tends to zero when distance $d(\mathbf{x}, \mathbf{v_j})$ goes to infinity is termed *localized* RBF [6]. A popular localized RBF is the Gaussian Radial Basis Function (GRBF)

$$g_j(d(\mathbf{x}, \mathbf{v_j})) = e^{-\frac{|\mathbf{x} - \mathbf{v_j}|^2}{2\sigma_j^2}}, \tag{24}$$

where σ_j, $j = 1, ..., C$, are scale parameters determining the size of the receptive fields of the network's GRBFs, and $|\mathbf{x} - \mathbf{v_j}|$ is the Euclidean distance. Thus, GRBFs

[1] This optimization algorithm is called back-propagation because the output error is "back-propagated" through the network all the way down to the free parameters of the input layer [16]. It is their simple and easily implementable learning strategy that makes conventional MLPs the most popular NN model [34].

are equivalent to small patches (Gaussian data windows) covering the input space. Among localized RBF types, GRBFs are preferred for two main reasons: a) they have a number of useful analytical properties, e.g., a Gaussian function is factorizable; as a consequence, their implementation in parallel hardware is straightforward, which makes GRBFs particularly attractive for real-time network implementations [19]; and b) they are claimed to constitute a processing module common in the human nervous system [20]. Whenever GRBFs are adopted, Eq. (9) becomes

$$\hat{\mathbf{y}}(\mathbf{x}, \mathbf{w}, \mathbf{v}, \sigma) = \sum_{j=1}^{C} \mathbf{w_j} g_j(\mathbf{x}; \mathbf{v_j}, \sigma_j), \tag{25}$$

where the adjustable parameters are termed as follows: i) *structural parameters* of the hidden layer are number C of GRBFs, position (prototype vector) $\mathbf{v_j}$ and standard deviation (spread parameter) σ_j, $j = 1, ..., C$, of the GRBFs; and ii) *output weights* of the output layer are parameters $\mathbf{w_j}$, $j = 1, ..., C$.

3. When hidden units employ a localized RBF type, e.g., GRBF, then only a few hidden units typically have significant activations for a given input vector.

4. Free parameters are typically trained according to a two-stage *hybrid learning procedure* combining an *unsupervised first stage* with a *supervised second stage*. The first stage is termed *unsupervised or data-driven* because it assigns hidden unit parameters on the basis of the density of the input vectors exclusively, i.e., without relating to the (input, output) vector pairs of the supervised training set. The second stage is termed *supervised or error-driven* because it determines the output weights of the output layer on the basis of the supervised training set.

5. To provide a smooth interpolation function in which the number of RBFs is determined by the complexity of the mapping to be approximated rather than by the size of the training data set, RBF networks employ a number C of basis functions that must be less than number M of training samples [6], [17]. This is tantamount to stating that C is considered a *regularization parameter* [1].

Both MLPs and RBF networks have been proved to work as universal approximators, i.e., they are capable of approximating any arbitrary mapping between multidimensional spaces to any degree of precision if they are provided with a sufficient number C of processing units [21]. Note that when the distance between two consecutive Gaussians becomes vanishingly small, in the 1-D case where $n_i = n_o = 1$ (for the sake of simplicity), Eq. (25) can be written as

$$\hat{y}(x) = \int_{\mathcal{R}} w(v) g(x - v; \sigma) dv = w(x) * g(x; \sigma), \tag{26}$$

where symbol " $*$ " identifies the convolution product.

3.3 Links between RBF networks and other function approximation techniques

An important property of RBF networks is that they form a unifying link between a number of different learning systems including:

(a) Zero-order Sugeno fuzzy systems (where the ouput of each fuzzy rule is a constant), which are equivalent to RBF networks under certain mild restrictions [22], [23].

(b) Support Vector Machines (SVMs), whose training always finds a global minimum of a cost function for a classification task. An SVM is largely characterized by the choice of a *kernel function* (i.e., a weighting function). When this kernel function is a Gaussian, then SVM is called SVM RBFs, where the number of RBFs, their centers and their interpolation coefficients are all determined automatically by the SVM training and have a simple geometric interpretation [24]. The traditional view of RBF networks has been one in which the centers of RBFs are regarded as templates or prototypes. In line with this view it is reasonable to exploit a clustering heuristic to train the first layer of RBF networks. In contrast, *SVMs consider the centers of the RBFs as those training examples that are critical for a given predictive task.* These critical training examples are termed *support vectors.*

(c) Regularization theory, in which a mapping function is determined by minimizing a cost function designed to penalize mappings that are not smooth [6], [17].

(d) Noise interpolation theory, in which the observable output data is generated by a target function, which is smooth and noise-free, while the input data is corrupted by additive Gaussian noise [6], [25];

(e) Kernel regression, in which a noise-affected function is estimated by means of the Gaussian kernel-based method for density estimation [6].

(f) Distance-weighted regression methods, also termed instance-based [4], or memory-based learning methods [11] (see Section 2.5). Instance-based methods construct only a local approximation to the target function that applies in the neighborhood of the new query instance, and never construct an approximation designed to perform well over the entire input space [4]. RBF networks employing localized RBFs to construct approximations that perform well over the entire input space can be considered an interesting bridge between instance-based and neural network learning algorithms [4].

(g) Bayes optimal classifiers, where the density of the input data is expressed in terms of a mixture distribution whose components are the basis functions [4], [6].

4 Learning techniques for RBF networks

The traditional view of RBF networks where the centers of RBFs are regarded as templates has led to the development of hybrid learning schemes whose first stage performs unsupervised clustering. Advantages and drawbacks of two-stage hybrid learning procedures in comparison with error-driven adaptation strategies for RBF networks are analyzed hereafter.

4.1 Hybrid learning techniques

Hybrid learning techniques used to train RBF networks can be significantly faster than error-driven methods used to train MLPs. In the first stage of a hybrid learning procedure,

parameters of the hidden units are determined using relatively fast, unsupervised methods. One example of an unsupervised method is the Expectation-Maximization (EM) procedure for density function estimation. The EM algorithm tries to maximize the likelihood of input vectors, $p(\mathbf{x})$, represented as a mixture distribution of basis functions $p(j)$, $j \in \{1, C\}$, according to the expansion rule $p(\mathbf{x}) = \sum_j^C p(\mathbf{x}|j)p(j)$, where the number of basis functions C is user-defined [6], [26]. Other examples of unsupervised methods are the clustering algorithms based on histogram analysis where, from the sequence of input patterns, prototype vectors are extracted as statistical regularities capable of minimizing a requantization error (e.g., Hard c-means [27], Fuzzy c-means [28], LBG-U [29]).

Once parameters of the hidden layer have been estimated by the first stage, the second training stage of the hybrid learning procedure determines the values of the output weights by minimizing a cost functional on the basis of the supervised training set. When the cost function is the sum-of-squares error, output weights can be determined by using a pseudo-inverse solution of a linear problem, which is therefore also fast [6]. Otherwise, a slower gradient descent approach can be employed.

RBF networks employing a two-stage hybrid learning procedure feature the following advantages over MLPs:

i) they employ simpler architectures to perform complex mappings;

ii) they can be trained faster;

iii) they may be particularly attractive for applications where input patterns are readily available but (input, output) sample pairs are difficult to gather;

iv) they are easily interpretable if RBFs are well localized (e.g., a GRBF is well localized when its adjustable parameter σ does not tend to either zero or infinity);

and v) the use of unsupervised learning methods can be quite successful in practice when the distribution of the input patterns is highly non-uniform and/or its effective dimensionality is small, i.e., correlation between input variables is high [1].

Disadvantages of hybrid learning techniques for RBF networks are that:

i) most unsupervised clustering techniques require the user to fix several parameters of the hidden layer in advance; this is typically the case for the regularization parameter C (number of RBFs) and/or spread parameters of GRBFs;

and ii) the distribution of RBFs in the input space as computed by the unsupervised technique does not reflect the local complexity of the classification or regression problem at hand; e.g., unsupervised methods may form clusters of input vectors that are closely spaced in the input space but belong to different classes [6], [23], [30], [31].

These disadvantages imply that when the number of RBFs (i.e., the number of adjustable parameters) is increased either by the user or by a constructive clustering algorithm in practical applications [32], [33], there is no guarantee of improving the system's performance on a test set, i.e, on a set of unobserved labeled examples, because the unsupervised algorithm may locate the additional RBFs in regions of the input space where they are either useless or harmful in implementing the desired (input, output) mapping [31].

To avoid this problem, the density of RBFs must be made independent of input vector density but dependent on the complexity of the desired (input, output) mapping. Thus, several error-driven (supervised) learning algorithms for RBF networks, either one- or two-stage, have been proposed in recent years. In this learning framework the density of RBFs is computed on the basis of the supervised training set according to a cost function minimization criterion.

4.2 Error-driven learning techniques

In error-driven learning, all adjustable parameters of a predictive learning system are adapted to minimize a cost function computed on the basis of the training set as a difference between target outputs and system outputs. While for MLPs a simple optimization algorithm based on gradient descent of a sum-of-squares error has been devised to determine the network weights, it has been proved that when this technique is applied, for example, to GRBF networks, it is unsuccessful because [34]: i) it does not ensure that GRBFs will remain localized, i.e., spread parameters may become very large and GRBF responses may become very broad [18]; and ii) it has practically no effect on the positions (centers) of GRBFs.

To overcome these limitations Karayiannis has introduced new types of localized RBFs suitable for gradient descent learning [34]. Drawbacks of this supervised learning approach are that it does not guarantee convergence to the absolute minimum of the cost function and that it does not select model complexity (i.e., number C of hidden units must be user-defined).

A different supervised learning approach is offered by error-driven growing RBF networks. In this approach the number of hidden units is allowed to grow until a convergence criterion is met, and when a new hidden unit is generated, an error-driven insertion criterion locates the center of the RBF in the input space. According to their insertion criterion, growing RBF networks can be divided into *scatter-partitioning RBF networks* [23], [31], where centers and spread parameters of RBFs are adjustable, and *grid-partitioning RBF networks*, where the input space is partitioned into a regular grid of identical patches whose size is known *a priori*, i.e., center and spread parameter values of the hidden layer are neither error- nor data-driven [35].

Unlike constructive grid-partitioning RBF networks, traditional (non-growing) grid-partitioning schemes, where all structural parameters are set on the basis of a rigid gridding of the input space, tend to be inefficient and not robust. However, they offer the advantage of employing the input space gridding mechanism to reduce the number of free parameters in the network [25].

This advantage is fully exploited in the Hierarchical Radial Basis Functions (HRBF) network model where an error-driven mechanism identifies useful RBFs at the crossings of a hierarchy of grid partitions of the input space [35]. In other words, RBFs belonging to higher grids are dynamically added to the basic (first) layer of the HRBF network only when required by an error-driven local convergence criterion. Moreover, in the basic layer of the HRBF network the number of RBFs and the size of the grid is not set heuristically but is data-driven according to linear filtering theory.

Table 1 shows a brief review of some learning strategies employed by RBF networks.

Note the wide range of learning solutions that makes the comparison of these predictive systems extremely difficult. This observation suggests the idea that to make RBF networks more popular, these networks must be provided with a single-stage simple and easily implementable error-driven learning algorithm analogous to the error back-propagation algorithm for MLPs [34].

5 Introduction to Hierarchical Radial Basis Function networks

In this section a function estimator called Hierarchical Radial Basis Function (HRBF) network is described. HRBF can be employed to approximate mappings between multidimensional spaces. It applies a growing mechanism, based on linear filtering theory, to achieve a theoretically sound and computationally fast adaptation of the parameters in both the hidden and output layers on the basis of the supervised training set. This section is organized as follows. Firstly, a simple criterion is driven from linear filtering theory to compute the position and the spread parameter of hidden units. Next, output weights are determined according to a locally distance-weighted regression method, and an error-driven procedure that allocates a hierarchy of GRBFs to selected areas of the input domain is described. Finally, experimental results on surface reconstruction from 3-D range data are presented and discussed.

5.1 Inter-Gaussian distance and Gaussian spread according to linear filtering theory

For simplicity's sake the following analysis is carried out in the case in which both input and output spaces are continuous and 1-D. In this case the supervised training set consists of the (input, output) pairs of real values (x_h, y_h), $h = 1, ..., M$. Results can be extended to multidimensional spaces by observing that multidimensional Gaussians are obtained by factorizing 1-D Gaussians. Note that when 1-D GRBFs featuring the same structural parameter σ are employed, Eq. (25) is simplified as

$$\hat{y}(x) = \sum_{j=1}^{C} w_j g_j(x; \mu_j, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \sum_{j=1}^{C} w_j e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}, \tag{27}$$

where symbol μ_j is now employed to identify the center of the jth Gaussian basis function.

5.1.1 The Gaussian low-pass filter

Let us consider a target function $y(x)$ whose Fourier Transform (FT) is represented as $\mathcal{F}(y(x)) = Y(\nu)$. The maximum frequency content of $y(x)$, identified as ν_{MAX}, is defined as the frequency above which values of the FT amplitude spectrum $|Y(\nu)|$ are trivial. An ideal low-pass filter for signal $y(x)$ is defined as the function $h_i(x)$ whose FT amplitude spectrum, $|\mathcal{F}(h_i(x))| = |H_i(\nu)|$, is equal to 1 for $\nu \leq \nu_{MAX}$, and 0 elsewhere. According to this definition, frequency ν_{MAX} separates the so-called Pass Band from the Stop Band of the filter. In real implementations of a low-pass filter the transition between the two bands

Table 1:

Learning strategy	Algorithm	Structural Parameters			Synaptic weights
		no. of RBFs	$\{\mu\}$	{Spread}	
one-stage	[41][1], [42][1]	EGI and ER	D	D	EGD[8]
	[34][2]	F	EGD[6]	F	EGD[6]
	[24]	ES[3]	ES[3]	F	E
	[35]	ES[4]	ES[4]	F[4]	E
two-stage	[31]	EGI[5]	EGD[7]	EGD[7]	EPI or EGD[6]
	[18]	F	D	H	EPI

D: (Input) Data-driven;
E: Error-driven;
ER: Error-driven Removal of localized RBFs;
ES: Error-driven Selection of localized RBF;
EGD: Error Gradient Descent;
EGI: Error-driven Generation/Insertion of localized RBFs;
EPI: Error-driven Pseudo-Inverse solution of a linear problem;
F: Fixed (user-defined);
H: Heuristic criterion;

[1] Fritzke's algorithm is termed Supervised Growing Neural Gas (SGNG). Its hidden layer, termed Growing Neural Gas (GNG), is described in [41]; its output layer is described in [42].
[2] The type of RBFs is feasible for gradient descent learning.
[3] Centers of selected RBFs correspond to support vectors belonging to the training data set.
[4] Centers of selected RBFs correspond to grid-crossings of a hierarchy of regular grids featuring increasing sampling rate. The sampling rate of the regular grid at level one is eventually computed from the data on the basis of linear filtering theory; for next layers, the sampling rate doubles the sampling rate value of the lower level.
[5] Stopping criterion: back-tracking through cross-validation (non-greedy algorithm) [4].
[6] (Batch) Gradient Descent of the sum-of-squares error;
[7] Gradient Descent of the Class-conditional Variance;
[8] Stochastic (on-line) Gradient Descent of the sum-of-squares error;

does not consist of a step edge nor is the filter response within each band flat. Therefore, a realistic low-pass filter, identified as $h(x)$, features a monotonically decreasing transfer function that can be described as follows.

Definition 1. The Pass Band of a realistic low-pass filter $h(x)$ is defined as the interval in which the amplitude spectrum of filter FT, $|H(\nu)|$, is bounded in range $[d_1, 1]$, i.e.,

$$|H(\nu)| \in [d_1, 1] \text{ when } \nu \in [0, \nu_{cut-off}], \tag{28}$$

where $\nu_{cut-off}$, called cut-off frequency, is such that $|H(\nu_{cut-off})| = d_1$, where $0 << d_1 < 1$ is a functional parameter. For example, a common choice in digital filtering theory is $d_1 = \sqrt{2}/2$, corresponding to a maximum attenuation in the Pass Band equal to 3 db [11].

Definition 2. The Stop Band of a realistic low-pass filter is defined as the interval in which the amplitude spectrum of filter FT, $|H(\nu)|$, is bounded in range $[0, d_2]$, i.e.,

$$|H(\nu)| \in [0, d_2] \text{ when } \nu \in [\nu_{max}, \infty), \tag{29}$$

where ν_{max}, called maximum frequency, is such that $|H(\nu_{max})| = d_2$, where $0 < d_2 << d_1$ is a functional parameter. A conservative choice, used in this work, is $d_2 = 0.01$. As a consequence of definitions 1 and 2, relationship $\nu_{max} > \nu_{cut-off} > \nu_{MAX}$ holds true. The interval between $\nu_{cut-off}$ and ν_{max} is termed Transition Band.

Our analysis considers a GRBF written as

$$g(x;\sigma) = \frac{e^{-\frac{(x)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}}, \tag{30}$$

whose FT is

$$\mathcal{F}(g(x;\sigma)) = G(\nu;\sigma) = e^{-\pi^2\sigma^2\nu^2}. \tag{31}$$

Eq. (31) shows that the Gaussian function $g(x;\sigma)$ is a low-pass filter whose amplitude spectrum response $|G(\nu;\sigma)|$ is a monotone decreasing function of both frequency ν and structural parameter σ. If the values of thresholds d_1 and d_2 are chosen as recommended above, the decreasing monotonicity of $G(\nu;\sigma)$ allows a relating of the values of $\nu_{cut-off}$ and ν_{max} to σ as follows

$$\begin{cases} e^{-\pi^2\sigma^2\nu^2_{cut-off}} = d_1 \Longrightarrow \nu_{cut-off} = \frac{\sqrt{-ln(d_1)}}{\pi\sigma} = \frac{\sqrt{-ln(\sqrt{2}/2)}}{\pi\sigma} = \frac{0.1874}{\sigma}, & (32a) \\ e^{-\pi^2\sigma^2\nu^2_{max}} = d_2 \Longrightarrow \nu_{max} = \frac{\sqrt{-ln(d_2)}}{\pi\sigma} = \frac{\sqrt{-ln(0.01)}}{\pi\sigma} = \frac{0.6831}{\sigma}, & (32b) \end{cases}$$

On the basis of Eq. (32) we can also write

$$\nu_{cut-off} = 0.2743\nu_{max}. \tag{33}$$

5.1.2 Discrete convolution of GRBFs

From the sampling theorem introduced by Shannon and applied to information theory it is known that when a band-limited signal $y(x)$ (i.e., a signal featuring finite ν_{MAX}) is sampled at a sampling interval $\triangle x < 1/2\nu_{MAX}$, then the signal can be (perfectly) reconstructed by

using an ideal low-pass filter (see Section 5.1.1), whose response in the input domain is a sinc function, according to the following equation [36]

$$y(x) = \sum_{j=-\infty}^{+\infty} y_j \frac{sin\left(\pi \frac{(x-x_j)}{\triangle x}\right)}{\pi \frac{(x-x_j)}{\triangle x}} = \sum_{j=-\infty}^{+\infty} y_j sinc\left(\frac{x-x_j}{\triangle x}\right), \tag{34}$$

where (x_j, y_j) are the (input, output) observed examples such that $x_j = x_0 + j\triangle x$, $j \in \{-\infty, +\infty\}$, are equally-spaced sampling points, and kernel functions $sinc\,(x - x_j/\triangle x)$ form an orthogonal basis. In real problems it is necessary to truncate the series in Eq. (34); moreover, the ideal low-pass filter is unrealizable and must be approximated adequately.

The HRBF network, starting from Eq. (26), adapts the sampling theorem (34) to non-ideal function regression tasks as follows. Let us identify with $\triangle\mu$ the distance between two consecutive centers of equally-spaced Gaussians and with $\nu_G = 1/\triangle\mu$ their spatial frequency. In the discrete case of a finite number of equally-spaced Gaussians, Eq. (26) becomes [35]

$$\hat{y}(x) = \frac{\triangle\mu}{\sigma\sqrt{2\pi}} \sum_{j=1}^{C} w(\mu_j) e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}. \tag{35}$$

To provide a reasonable approximation of target function $y(x)$, Eq. (35) must employ a uniform sampling interval $\triangle\mu$ and a spread parameter σ which are constrained as follows. The first constraint is driven by the observation that all frequency components of target function $y(x)$ should be replicated. This means that the cut-off frequency of the Gaussian filter should be greater than the frequency content of signal $y(x)$, i.e.,

$$\nu_{cut-off} > \nu_{MAX}. \tag{36}$$

The second observation is that since Eq. (35) is a discrete, finite sum, then its FT amplitude spectrum is infinite and periodical of period $\triangle\nu = \nu_G$, such that replicas of the main spectrum are equally spaced by an interval equal to ν_G on the frequency axis. To avoid overlapping of the spectrum components, the condition

$$\nu_{max} < \nu_G/2, \tag{37}$$

must hold true [35]. By combining Eqs. (32) and (33) with inequalities (36) and (37), the following relations are obtained

$$\begin{cases} \nu_{MAX} < \nu_{cut-off} = \frac{0.1874}{\sigma} = 0.2743\nu_{max} < 0.2743\nu_G/2 = 0.1371\nu_G, & (38a) \\ \frac{\nu_{MAX}}{0.2743} < \frac{\nu_{cut-off}}{0.2743} = \nu_{max} = \frac{0.6831}{\sigma} < \nu_G/2. & (38b) \end{cases}$$

Eq. (38) can be written as

$$\frac{2 \cdot 0.6831}{\nu_G} = \frac{1.3662}{\nu_G} = 1.3662\triangle\mu = \sigma_{min} \leq \sigma \leq \sigma_{max} = \frac{0.1874}{\nu_{MAX}}. \tag{39}$$

Eq. (39), where $\sigma > 1.3662\triangle\mu$, is more restrictive than the empirical criterion $\sigma = \triangle\mu$, which is typically employed in Parzen window-based density function estimators [7], [37]. The difference is significant in terms of amount of overlap (oversampling [12]) between two

consecutive Gaussians, which increases from 68.2% to 73.3%. On the other hand, our result is consistent with the heuristic criterion employed by Bishop, where the spread parameter σ of hidden units in RBF networks is roughly equal to twice the average spacing between the centres of unequally spaced GRBFs [6]. Constraint (39), equivalent to constraint (38), suggests that with a maximum frequency content of target function $y(x)$, ν_{MAX}, the smallest Gaussian sampling rate, $\nu_{G_{min}}$, is

$$\nu_{G_{min}} = \frac{1.3662\nu_{MAX}}{0.1874} = \frac{2\nu_{MAX}}{0.2743} = 7.2903\nu_{MAX} \Longrightarrow \triangle\mu_{max} = \frac{0.1371}{\nu_{MAX}}. \tag{40}$$

Eq. (40) points out that when GRBFs are employed to reconstruct a target function $y(x)$ according to Eq. (35), then the frequency of the Gaussian filters must be superior by factor 3.6451 to the sampling frequency of the Shannon theorem. In line with Occam's razor, which states that a sound basis for generalizing beyond a given set of examples is to prefer the simplest hypothesis that fits the data, our analysis suggests choosing $\nu_G = \nu_{G_{min}}$, i.e., $\triangle\mu = \triangle\mu_{max}$.

By analogy with the sampling theorem (34), let us consider the special case in which the weights employed in Eq. (35) are equal to the observed values of the target function in the Gaussian centers, i.e., $w(\mu_j) = y(\mu_j)$, $j = 1, ..., C$. In this case the dictionary representation (35) becomes equivalent to the kernel representation

$$\hat{y}(x) = \frac{\triangle\mu}{\sigma\sqrt{2\pi}} \sum_{j=1}^{C} y(\mu_j) e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}, \tag{41}$$

where the ideal low-pass filter employed in Eq. (34) has been implemented as a Gaussian filter. Eq. (41) means that to provide a reasonable approximation of target function $y(x)$, the uniform sampling interval (now equivalent to the Gaussian distance $\triangle\mu$) and the spread parameter σ must be chosen according to constraint (39). In other words, the sampling rate required by Eq. (41) to provide a reasonable approximation of the target function is superior by factor 3.6451 to the sampling frequency of Eq. (34) in which an ideal low-pass filter is employed. The advantage of Eq. (41) with respect to Eq. (35) is that the former can be computed easily when observed values of the target function are available for all equally-spaced Gaussian centers.

5.2 Locally distance-weighted regression for output weights

When the maximum frequency content ν_{MAX} of target function $y(x)$ is known, the Gaussian distance $\triangle\mu$ can be computed with Eq. (40). Next, the spread parameter σ can be computed according to Eq. (39). If the input space is a bounded interval $[x_{min}, x_{max}]$, the number of GRBFs required to reconstruct $y(x)$ is $C = \lfloor (x_{max} - x_{min})/\triangle\mu \rfloor = \lfloor (x_{max} - x_{min})\nu_{MAX}/0.1371 \rfloor$, where operator $\lfloor(\cdot)\rfloor$ returns the largest integer not greater than its real input value. Thus, all parameters in the hidden layer are determined. The next step consists of computing the output weights employed in Eq. (35). Since size M of the supervised training set must satisfy condition $M \geq C$ (see Section 3.2), then output weights can be computed by considering Eq. (35) as a linear model (see Section 3.1) which can be solved by pseudo-inverse techniques [6], [7], or algorithms which are numerically more stable, such

as the Singular Value Decomposition [38]. A better scheme, one that allows elimination of outliers, has recently been proposed [39]. However, these techniques are computationally demanding and may cause numerical and memory allocation problems for large networks. Thus, to compute the output weights employed in Eq. (35) a specific strategy has been developed for the HRBF network.

It has been shown that Eq. (41) can replace Eq. (35) when observed values of the target function are available for all equally-spaced Gaussian centers. In this case, computation of Eq. (41) is quite simple. However, the hypothesis of equally spaced samples is rarely satisfied in real problems. When a number $M \geq C$ of either equally- or unequally-spaced samples is available, then values of the target function at Gaussian centers can be estimated, for example, by means of a locally distance-weighted regression model [11]. In the less restrictive hypothesis of irregular but sufficiently dense sampling, the HRBF network adapts Eq. (41) as follows

$$\hat{y}(x) = \frac{\triangle\mu}{\sigma\sqrt{2\pi}} \sum_{j=1}^{C} \hat{y}^*(\mu_j) e^{-\frac{(x-\mu_j)^2}{2\sigma^2}}, \tag{42}$$

where symbol $\hat{y}^*(\mu_j)$ means that, to reduce computation time, the HRBF network, based on Eq. (42), estimates target function values at (unobserved) Gaussian centers by adapting Eq. (14) according to the following locally distance-weighted regression model

$$\hat{y}^*(\mu_j) = \frac{\sum_{x_h \in \{S_j\}} y_h k(d(x_h, \mu_j))}{\sum_{x_h \in \{S_j\}} k(d(x_h, \mu_j))}, \quad j = 1, ..., C, \quad h \in \{1, M\}, \tag{43}$$

where distance $d(x_h, \mu_j)$ is computed with Eq. (11) where $p = 2$, the kernel function $k(d(x_h, \mu_j))$ is computed according to Eq. (15), in which parameter σ_w is set equal to parameter σ already employed in Eq. (42), and set $\{S_j\}$ consists of all training patterns whose input value belongs to the domain interval centered on μ_j of amplitude $\pm 3\triangle\mu$ (such that, according to Eq. (39), interval $\pm 3\triangle\mu \approx \pm 2\sigma$, i.e., this domain interval corresponds to an area of the Gaussian probability density function approximately equal to 96%).

To summarize, on the basis of the combination of Eq. (42) with Eq. (43), the HRBF network features two important properties: i) it can be applied when the training set consists of $M \geq C$ examples regardless of whether they are equally- or unequally-spaced; and ii) it is capable of filtering out noisy points (*outliers*) since coefficients $\hat{y}^*(\mu_j)$, $j = 1, ..., C$, are estimated as a weighted average of the training examples [35]. From the computational standpoint, the HRBF learning procedure requires carrying out as many local estimates as the number of GRBFs. Therefore, the computational complexity of the model increases with complexity C of the mapping rather than with size M of the training set. This scalability allows the implementation of the algorithm on parallel hardware.

5.3 Building the hierarchy of GRBF

When Eq. (42) employs a value of spread parameter σ that satisfies Eq. (39), i.e., σ is chosen as a function of the highest frequency content of the target signal $y(x)$, then Eq. (42) guarantees reconstruction of the finest details of $y(x)$. Unfortunately, this choice may cause a waste of resources when the highest frequency content of the target function is concentrated (localized) in a narrow region of the input space (i.e., when the target

signal is *non-stationary* [1]). In this case fewer Gaussians, featuring a larger spread, can be used to reconstruct the signal in those domain intervals where the scale of $y(x)$ is larger. The solution proposed in the HRBF architecture is to start the hierarchical model with a basic layer of GRBFs, termed layer 1, equivalent to a complete grid of C_1 Gaussians at a large scale. A hierarchy of processing layers, each layer identified by an integer $l > 1$ and featuring C_l processing units, is constructively made up by the HRBF error-driven learning strategy. The scale parameter of Gaussians decreases with level l in the hierarchy. The grid of each layer $l > 1$ may not be complete, since it consists of GRBFs located at selected grid-crossing positions. These positions are identified on the basis of the domain intervals where an average local error (residual), estimated by the lower processing layer, is above a user-defined threshold ϵ. Layers are added until the average local error is below threshold ϵ over the entire input domain (uniform convergence criterion).

Unlike traditional gridding procedures where no constructive mechanism is employed, the HRBF gridding procedure can be considered error-driven and one-stage. An approach similar to HRBF has been proposed by Fritzke using Kohonen maps [40]. The main difference is that Fritzke's system requires insertion of an entire row or column into the grid, while HRBF adds units selectively in those localized regions of the input space where the average local error criterion is not satisfied.

While in the HRBF learning stage the layers are trained sequentially one after the other, in the reconstruction process the different layers operate in parallel: each layer receives the same input, namely the position of a sample in the input space, and outputs a value which is an approximation of the function at the largest scale (first layer), and an approximation of the residual at higher layers. The actual approximation $\hat{y}(x)$ of target function $y(x)$ in point x is obtained by adding up the contributions of all the layers (parallel processing).

5.4 Summary of the HRBF learning procedure

The only user-defined parameter required by HRBF to run is an average local error threshold ϵ.

For a given supervised training set (x_h, y_h), $h = 1, ..., M$, the HRBF learning procedure can be summarized as follows:

1. For layer 1 (basic layer):

   (a) The spacing between two Gaussians, $\triangle\mu_1$, is computed as $\triangle\mu_1 > \triangle\mu_{max}$, where $\triangle\mu_{max}$ is provided by Eq. (40). The choice of $\triangle\mu_1$ should allow the reconstruction of at least the grossest details of target function $y(x)$. Given the value of $\triangle\mu_1$ and the extension of the input space, the number of GRBFs, C_1, is determined.

   (b) A regular and complete grid of (equally spaced) GRBFs is created, each GRBF featuring the same spread parameter σ_1 chosen according to Eq. (39).

   (c) The interpolation coefficients $\hat{y}^*(\mu_{j,1})$, $j = 1, ..., C_1$, are computed according to Eq. (43) on the basis of the supervised training set.

   (d) Estimated target values $\hat{y_1}(x_h)$, $h = 1, ..., M$, are computed according to Eq. (42)

whose interpolation coefficients are set equal to values $\hat{y}^*(\mu_{j,1})$, $j = 1, ..., C_1$, i.e.,

$$\hat{y_1}(x_h) = \frac{\triangle\mu_1}{\sigma_1\sqrt{2\pi}} \sum_{j=1}^{C_1} \hat{y}^*(\mu_{j,1}) e^{-\frac{(x_h - \mu_{j,1})^2}{2\sigma_1^2}}. \tag{44}$$

(e) Average local errors $\hat{r}(\mu_{j,1})$, $j = 1, ..., C_1$, are computed on the basis of the supervised training set according to the L_1 metric, i.e.,

$$\hat{r}(\mu_{j,1}) = \frac{\sum_{x_h \in \{S_j\}} |y_h - \hat{y_1}(x_h)|}{|S_j|} = \frac{\sum_{x_h \in \{S_j\}} \hat{r}_1(x_h)}{|S_j|}, \quad j = 1, ..., C_1, \quad h \in \{1, M\}, \tag{45}$$

where set $\{S_j\}$ is the same set defined in Eq. (43), $|S_j|$ is the cardinality of the set and $\hat{r}_1(x_h)$ is an example-based residual.

(f) Set $\{R_1\}$, consisting of identifiers of the GRBFs that satisfy inequality $\hat{r}(\mu_{j,1}) > \epsilon$, $j = 1, ..., C_1$, is generated.

2. For the higher layers, identified by index $l > 1$:

(a) At each processing epoch $l > 1$, a new, regular and complete grid of Gaussians, consisting of C_l units, is considered. The supervised training set presented to layer l consists of (input, output) pairs where target values are the example-based residuals computed by the lower level, i.e., $(x_h, \hat{r}_{l-1}(x_h))$, $h = 1, ..., M$.

(b) The spacing between two Gaussians, $\triangle\mu_l$, is computed as $\triangle\mu_l = \triangle\mu_{l-1}/2$.

(c) The scale associated with layer l is computed as $\sigma_l = \sigma_{l-1}/2$,

(d) Among C_l Gaussian units making up a regular and complete grid at level l, only those units whose receptive field $\mu_{j,l} \pm 3\triangle\mu_l$, $j = 1, ..., C_l$, operlaps (fully or in part) regions $\mu_{p,l-1} \pm 3\triangle\mu_{l-1}$, $\forall p \in \{R_{l-1}\}$, i.e., domain intervals where the average local error is significant, are preserved; the other units are discarded. A reduced set of $C_l^* \leq C_l$ GRBFs is therefore obtained for layer l. This reduced set of GRBFs is identified as $\{L_l\}$.

(e) On the basis of the supervised training set of residuals $(x_h, \hat{r}_{l-1}(x_h))$, $h = 1, ..., M$, interpolation coefficients for surviving GRBFs of layer l, identified as $\hat{r}^*(\mu_{j,l})$, $\forall j \in \{L_l\}$, are computed according to Eq. (43) which becomes

$$\hat{r}^*(\mu_{j,l}) = \frac{\sum_{x_h \in \{S_j\}} \hat{r}_{l-1}(x_h) k(d(x_h, \mu_{j,l}))}{\sum_{x_h \in \{S_j\}} k(d(x_h, \mu_{j,l}))}, \quad \forall j \in \{L_l\}, \quad h \in \{1, M\}. \tag{46}$$

(f) Estimated target values $\widehat{yr}_l(x_h)$, $h = 1, ..., M$, are computed according to Eq. (42), whose interpolation coefficients are set equal to values $\hat{r}^*(\mu_{j,l})$, $\forall j \in \{L_l\}$, i.e.,

$$\widehat{yr}_l(x_h) = \frac{\triangle\mu_l}{\sigma_l\sqrt{2\pi}} \sum_{j \in \{L_l\}} \hat{r}^*(\mu_{j,l}) e^{-\frac{(x_h - \mu_{j,l})^2}{2\sigma_l^2}}. \tag{47}$$

(g) Average local errors $\widehat{rr}(\mu_{j,l})$, $\forall j \in \{L_l\}$, are computed on the basis of the supervised training set of residuals according to the following expression, which is derived from Eq. (45):

$$\widehat{rr}(\mu_{j,l}) = \frac{\sum_{x_h \in \{S_j\}} |\hat{r}_{l-1}(x_h) - \widehat{yr}_l(x_h)|}{|S_j|} = \frac{\sum_{x_h \in \{S_j\}} \hat{r}_l(x_h)}{|S_j|}, \quad \forall j \in \{L_l\}, \quad h \in \{1, M\}. \tag{48}$$

(h) Generate set $\{R_l\}$, consisting of GRBFs satisfying inequality $\widehat{rr}(\mu_{j,l}) > \epsilon$, $\forall j \in \{L_l\}$.

(i) If set $\{R_l\}$ is empty, then stop (because a uniform approximation criterion is satisfied), otherwise iterate the procedure (goto step 2.(a)).

Once the training phase of the HRBF network has reached convergence, a hierarchy of $l_{tot} \geq 1$ layers is found. At this stage, all structural parameters and interpolation coefficients of every layer $l = 1, ..., l_{tot}$, have been computed. In the function reconstruction phase, based on Eqs. (44) and (47), the estimate of the target value for an input point x is computed as

$$\hat{y}(x) = \hat{y}_1(x) + \sum_{l=2}^{l_{tot}} \widehat{yr}_l(x), \tag{49}$$

5.5 Reconstruction of a 3-D human face with HRBF

In this application a 3-D scan of a real face is employed as input to a learning system whose goal is the reconstruction of the continuous 3-D face surface. This is usually a two-step procedure: in the first step a very large set of data points (10,000 $\div$ 100,000) is sampled over the face, and in the second step a mathematical model is fitted to these points [35]. Owing to its computational efficiency and capability in performing smooth surface reconstruction, the HRBF network is considered suitable for use as the second stage of a 3-D human face reconstruction procedure.

Fig. 1 shows a typical input data set, consisting of $M = 12,641$ data points, whose range is: min(X) = -36.4471; max(X) = 97.8207; min(Y) = -103.3898; max(Y) = 88.8846; min(Z) = 0; max(Z) = 93.1110. Dimension Z is measured in mm. The error of the real-time image processor (the Elite system [3]) is below 1 mm. To avoid removal of useful information, the average local error threshold ϵ is set to 0.5. Since frequency ν_{MAX} of the target surface is unknown, then the Gaussian interval $\triangle\mu_{max}$ cannot be derived from Eq. (40). Let us estimate $\triangle\mu_{max}$ as follows. If samples are equally spaced (i.e., if Eq. (41) holds true), then the value of $\triangle\mu_{max}$ can be computed as

$$\triangle\mu_{max} = \sqrt{(max(X) - min(X)) * (max(Y) - min(Y))/M} = \sqrt{134.26 \cdot 192.27/12,641} = \sqrt{2.04} = 1.43. \tag{50}$$

Based on Eq. (40), ν_{MAX} becomes $\nu_{MAX} = 0.1371/\triangle\mu_{max} = 0.1371/1.43 = 0.097$. Then, according to Eq. (39), $\sigma_{min} = \sigma_{max} = 1.3662\triangle\mu_{max} = 1.3662 \cdot 1.43 = 1.95$. If we expect to reconstruct the spatial-frequency content of the target signal up to ν_{MAX} by means of a HRBF network consisting of $l_{tot} = 4$ layers then, for the basic layer, $\triangle\mu_1 = 2^{(l_{tot}-1)} \cdot \triangle\mu_{max} = 2^3 \cdot 1.43 = 11.44$, $\sigma_1 = 2^3 \cdot 1.95 = 15.60$. As a consequence, the range data set is processed by a HRBF network whose first layer consists

Table 2:

Layer	$\triangle\mu$	σ	gauss. eff./tot.	MSE	Mean Error	SE Std. Dev.
1	11.44	15.60	$126/(176 = 11 \times 16)$	49.98	4.23	5.65
2	5.72	7.80	$513/(651 = 22 \times 32)$	9.05	0.23	3.00
3	2.86	3.90	$1968/(2562 = 44 \times 64)$	2.70	0.01	1.64
4	1.43	1.95	$5606/(10164 = 88 \times 128)$	1.55	-0.009	1.24

of $(max(X) - min(X))/\triangle\mu_1 = 134.26/11.44 = 11$ Gaussian units along the x-axis, and $(max(Y) - min(Y))/\triangle\mu_1 = 192.27/11.44 = 16$ Gaussian units along the y-axis. This basic layer guarantees a rough description of the face as shown in Fig. 2, where the reconstructed surface is affected by a significant bias (mean reconstruction error = 5.65), which is typical of a low-pass filtered signal. Fig. 2 shows the sequence of estimated values of the target function when outputs of the higher layers are progressively taken into consideration. As expected, the face becomes more detailed as higher layers are involved in the reconstruction process. The final reconstruction depicted on the bottom right of Fig. 2 is considered a good geometrical model of the face surface, where noise points are filtered out while meaningful details are preserved (mean reconstruction error = 1.24). The output of each layer is depicted in Fig. 3. Parameters of the different layers are summarized in Table 2.

Acknowledgments

We are grateful to S. Ferrari for his valuable comments and for providing us with the figures depicted in this work.

References

[1] V. Cherkassky and F. Mulier, *Learning From Data: Concepts, Theory, and Methods*, Wiley: New York, 1998.

[2] R. Serra and G. Zanarini, *Complex Systems and Cognitive Processes*, Springer-Verlag: Berlin, 1990.

[3] N. A. Borghese, G. Ferrigno, G. Baroni, A. Pedotti, S. Ferrari and R. Savarè, "Autoscan: a flexible and portable 3D scanner," *IEEE Computer Graphics and Applications*, pp. 2-5, May/June 1998.

[4] T. Mitchell, *Machine Learning*, McGraw-Hill: New York, 1997.

[5] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. IJCAI-93*, Int'l Joint Conferences on Artificial Intelligence Inc., Morgan Kaufmann, 1995.

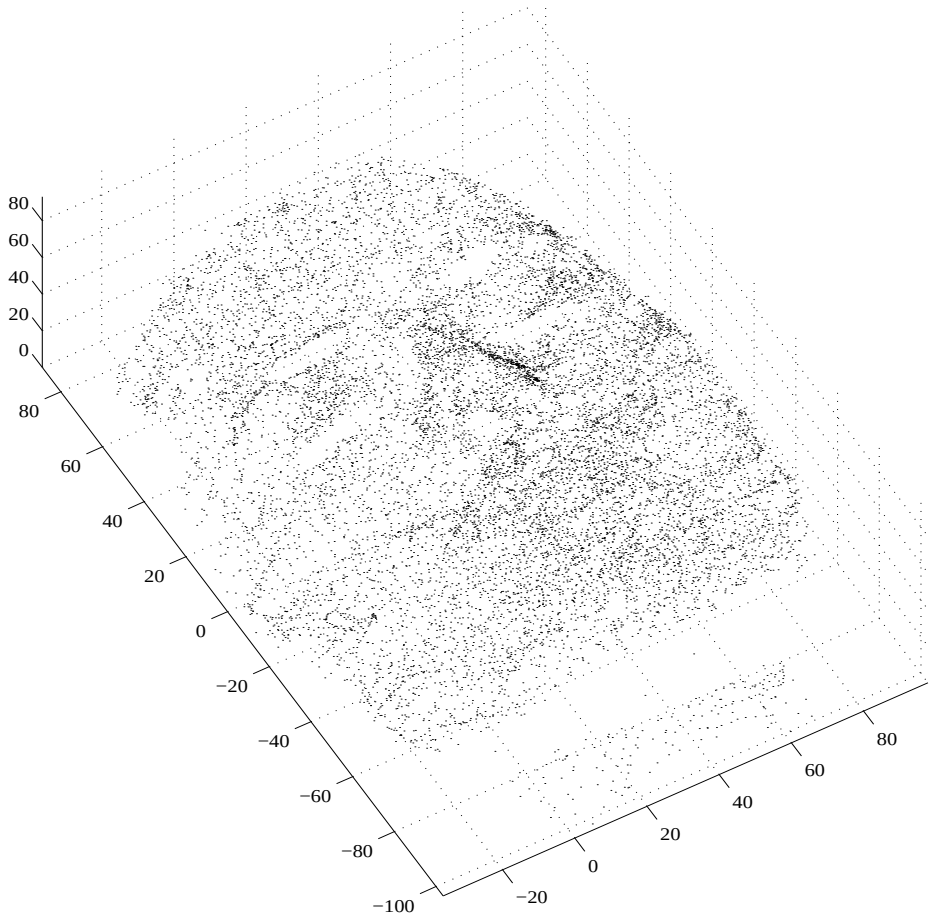

Figure 1: The ensemble of the 3-D points acquired over the face through Autoscan.

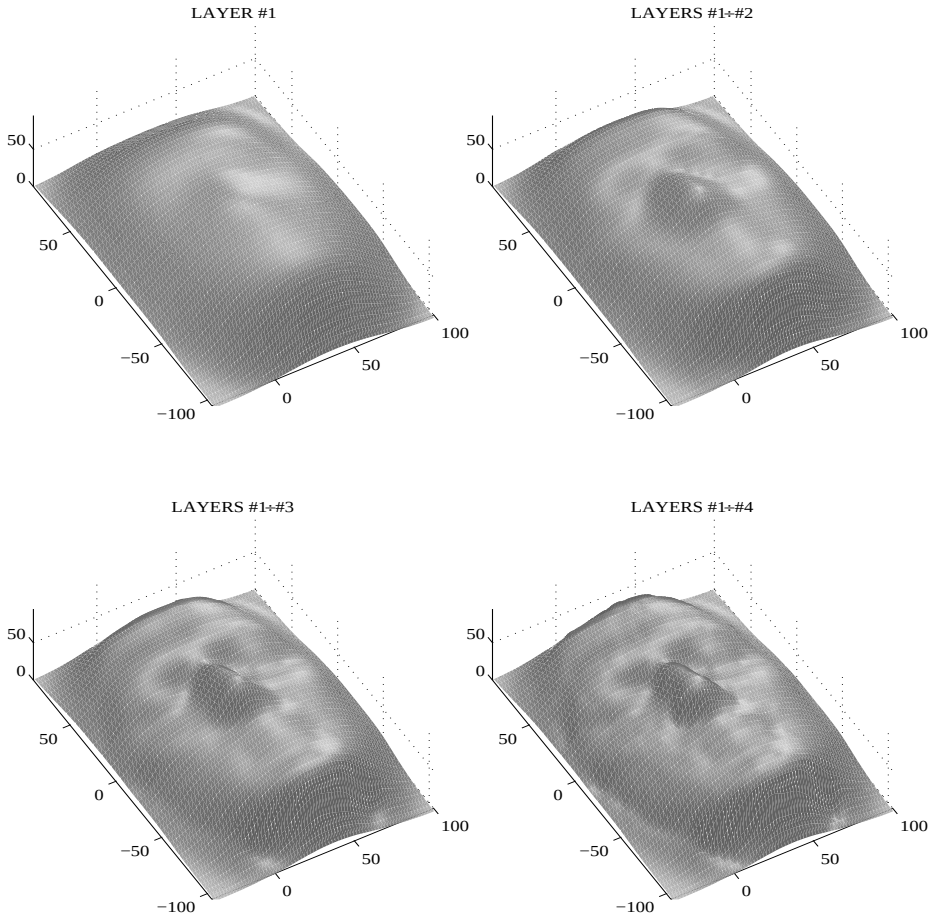

Figure 2: The reconstruction obtained using one, two, three and all four layers.

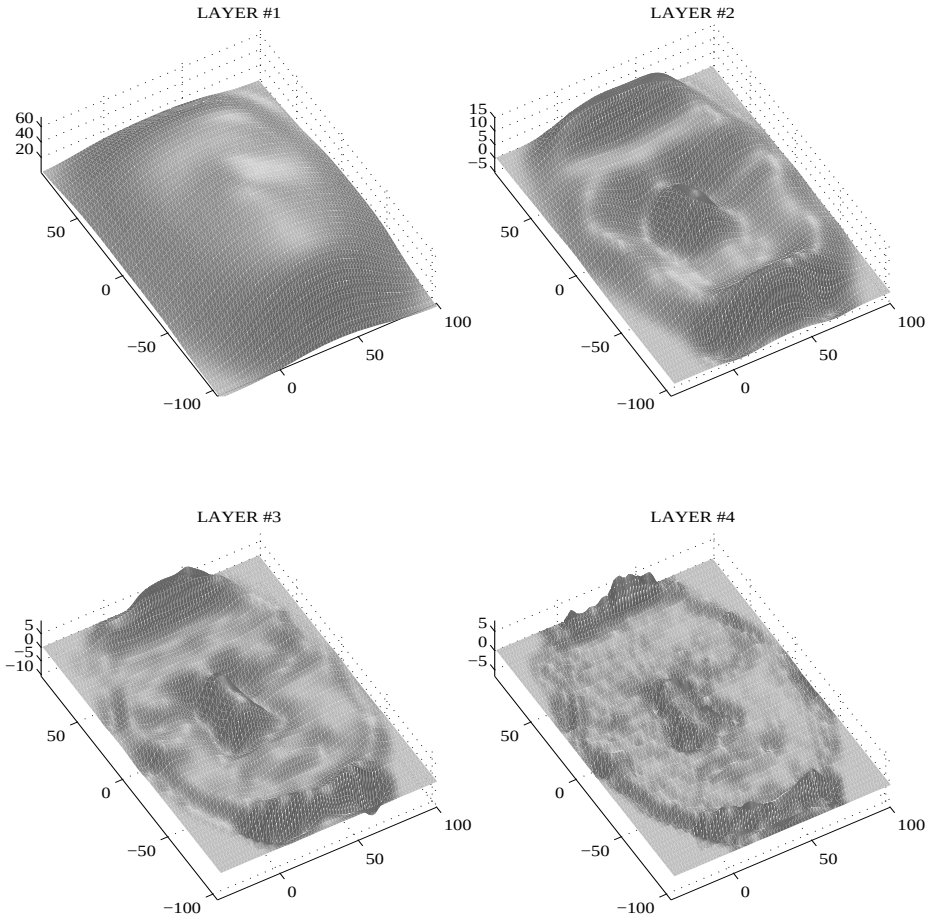

Figure 3: Output of each layer.

[6] C. M. Bishop, *Neural Networks for Pattern Recognition*, Clarendon Press: Oxford (UK), 1995.

[7] L. Xu, A. Krzyzak and A. Yuille, "On radial basis function nets and kernel regression: Statistical consistency, convergence rates, and receptive field size," *Neural Networks*, vol. 4, pp. 609-628, 1994.

[8] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distribution, and the Bayesian restoration of images," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, no. 6, pp. 721-741, Nov. 1984.

[9] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag: New York, 1995.

[10] C. Burges, "A tutorial on Support Vector Machines for pattern recognition," submitted to *Data Mining and Knowledge Discovery*, 1998.

[11] C. G. Atkeson, S. A. Schall and A. W. Moore, "Locally weighted learning," *AI Review*, vol. 11, pp. 11-73, 1997.

[12] M. Porat and Y. Zeevi, "The generalized gabor scheme of image representation in biological and machine vision," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 10, no. 4, pp. 452-467, July 1988.

[13] P. Burt and E. Adelson, "The Laplacian pyramid as a compact image code," *IEEE Trans. on Communications*, vol. COM-31, no. 4, pp. 532-540, April 1983.

[14] J. B. Martens, "The Hermite transform - Applications," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-38, pp. 1607-1618, 1990.

[15] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press: Wellesley (MA), 1997.

[16] D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart and J. L. McClelland, Eds., vol. 1, ch. 8, pp. 318-362, MIT Press: Cambridge, 1986.

[17] F. Girosi, M. Jones and T. Poggio, "Regularization theory and neural network architectures," *Neural Computation*, vol. 7, pp. 219-269, 1995.

[18] J. Moody and C. Darken, "Fast learning in networks of locally-tuned processing units," *Neural Computation*, vol. 1, pp. 281-294, 1989.

[19] T. Poggio, V. Torre and C. Koch, "Computational vision and regularization theory," *Nature*, vol. 317, pp. 314-319, 1985.

[20] T. Poggio, "A theory of how the brain might work," *Cold Spring Harbor Symposium on Quantitative Biology*, pp. 899-910, 1990.

[21] J. Park and I. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, p. 246-257, 1991.

[22] J. Jang and C. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Trans. on Neural Networks*, vol. 4, no. 1, pp. 156-159, 1993.

[23] B. Fritzke, "Incremental neuro-fuzzy systems," *Proc. SPIE's Optical Science, Engineering and Instrumentation '97: Applications of Fuzzy Logic Technology IV*, San Diego, July 1997.

[24] B. Schölkopf, K. Sung, C. Burges, F. Girosi, P. Niyogi, T. Poggio and V. Vapnik, "Comparing Support Vector Machines with Gaussian kernels to radial basis function classifiers," *IEEE Trans. on Signal Processing*, vol. 45, no. 11, pp. 2758-2765, 1997.

[25] M. Cannon and J. E. Slotine, "Space-frequency localized basis function networks for nonlinear system estimation and control," *Neurocomputing*, vol. 9, pp. 293-342, 1995.

[26] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Royal Statist. Soc. Ser. B*, vol. 39, pp. 1-38, 1977.

[27] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*, Prentice Hall: Englewood Cliffs, New Jersey, 1988.

[28] J. C. Bezdek and N. R. Pal, "Two soft relatives of learning vector quantization," *Neural Networks*, vol. 8, no. 5, pp. 729-743, 1995.

[29] B. Fritzke, "The LBG-U method for vector quantization - an improvement over LBG inspired from neural networks," *Neural Processing Letters*, vol. 5, no. 1, pp. 83-111, 1997.

[30] E. Alpaydın, "Soft vector quantization and the EM algorithm," *Neural Networks*, in press, 1998.

[31] N. Karayiannis, "Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques," *IEEE Trans. on Neural Neworks*, vol. 8, no. 6, pp. 1492-1506, 1997.

[32] S. Medasani and R. Krishnapuram, "Determination of the number of components in Gaussian mixtures using agglomerative clustering," *Proc. Int. Conference on Neural Networks*, Houston, 1997, pp. 1412-1417.

[33] C. Yiu-Ming and L. Xu, "Some further studies on detection the number of clusters," *Proc. Int. Conference on Neural Networks*, Houston, 1997, pp. 1479-1483.

[34] N. Karayiannis, "Reformulated radial basis neural networks trained by gradient descent," *IEEE Trans. on Neural Networks*, under review, 1998.

[35] N. A. Borghese and S. Ferrari, "Hierarchical RBF networks and local parameters etimate," *Neurocomputing*, vol. 19, in press, 1998.

[36] G. C. Holst, *Sampling, Aliasing, and Data Fidelity*, SPIE Press: Bellingham (WA), 1998.

[37] D. F. Specht, "Probabilistic neural networks," *Neural Networks*, vol. 3, pp. 109-118, 1990.

[38] W. Press, W. Vetterling, S. Teukolsky and B. Flannery, *Numerical Recipes in C*, Cambridge University Press: Cambridge (MT), 1992.

[39] D. V. Sanchez, "Robustization of a learning method for RBF networks," *Neurocomputing*, vol. 9, p. 85, 1995.

[40] B. Fritzke, "Growing-grid - a self-organizing network with constant neighborhood range and adaptation strength," *Neural Processing Letters*, vol. 2, no. 5, pp. 1-5, 1995.

[41] B. Fritzke, "Some competitive learning methods," Draft document, *http://www.neuroinformatik.ruhr-uni-bochum.de/ini/VDM/ research/gsn/DemoGNG*, 1998.

[42] B. Fritzke, "Growing cell structures - A self-organizing network for unsupervised and supervised learning," *Neural Networks*, vol. 7, no. 9, pp. 1441-1460, 1994.