INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Neural Networks as Dynamical Bases in Function Space

Konrad Weigl & Marc Berthod

N° 2124

December 1993

PROGRAMME 4

Robotique,
image
et vision

Rapport de recherche

1993

ISSN 0249-6399

Neural Networks as Dynamical Bases in Function Space

Konrad Weigl & Marc Berthod

Programme 4 — Robotique, image et vision
Projet Pastis

Rapport de recherche n ° 2124 — December 1993 — 39 pages

Abstract: We consider feedforward neural networks such as multi-layer perceptrons as non-orthogonal bases in a function space. The basis functions of that base are the functions computed by the neurons of the hidden layer. *Projection Learning* consists in shifting the base in such a way that the distance between a function to be approximated and its projection onto the manifold spanned by the base is minimized. That projection is the approximation by the network. The basis functions to be used are arbitrary, except that they must be differentiable in some sense with regards to their parameters/weights.
We present the paradigm and learning rule using multi-layer perceptrons and bases of multivariate Gaussians, discuss some other potential applications, and present alternatives ways to compute the projection.
The resulting networks display graceful degradation, integrate gracefully further neurons/basis functions for an improved approximation, and adapt dynamically to changes in the environment.

Key-words: Tensor theory, projection operators, metric tensors, multi-variate gaussians, multi-layer perceptrons

(Résumé : tsvp)

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Les réseaux neuromimétiques en tant que bases dynamiques dans l'espace des fonctions

Résumé : Nous considérons des réseaux neuromimétiques à propagation directe tels que les perceptrons multi-couches comme des bases non-orthogonales dans un espace de fonctions. Les fonctions qui composent cette base sont celles calculées par les neurones de la couche cachée. L'*Apprentissage par Projection* consiste à ajuster la base de telle façon que la distance entre la fonction à approximer et sa projection sur la variété engendrée par la base est minimale. Cette projection est l'approximation par le réseau. Les fonctions de base utilisables sont arbitraires, sauf qu'elles doivent être différentiables d'une manière ou d'une autre par rapport à leurs paramètres/poids.
Nous présentons l'application du paradigme et de la règle d'apprentissage à des perceptrons multi-couches et des bases de gaussiennes multi-variantes, discutons de quelques autres applications potentielles, et présentons différentes alternatives pour le calcul de la projection.
Les réseaux se montrent robustes à la dégradation, intègrent d'une manière flexible des neurones/fonctions de base supplémentaires, et s'adaptent dynamiquement à leur environnement.

Mots-clé : théorie des tenseurs, opérateurs de projection, tenseurs métriques, gaussiennes multi-variantes, perceptrons multi-couches

Contents

1 Introduction to Function Spaces 2

2 Function Approximation via Non-orthogonal Bases 2

3 Alternative Approaches to Direct Matrix Inversion 8

3.1 Computation of the Biorthogonal Base 8

3.2 Computation of the Contravariant Components by Gradient Descent . 9

3.3 Computation of the Contravariant Components by Recursive Least Squares 10

3.4 Comparison of Methods used to Compute Projection 12

3.5 The Algorithm Step by Step 13

4 Other Metrics for Functional Spaces 15

5 Some Results and Further Features of Dynamical Non-orthogonal Bases 17

6 Discussion, Conclusion and Outlook 19

1 Introduction to Function Spaces

For our needs it is not necessary to delve too deep in the intricacies of function spaces, since we are only considering finite-dimensional spaces. Consider the figure 1: We have a neural network with one input neuron, two hidden neurons, and one output neuron. Note that the functions computed by the hidden-layer neurons, called *basis functions*, are arbitrary, as long as they are differentiable *in some sense* wrt their parameters/weights: We can thus assume gaussians, sigmoids of weighted sums, etc. as basis functions. The functions do not need to be inversible, as e.g. Least-Squares-Backpropagation and other learning algorithms [5] [6] require them to be. We refer also to [29], pg. 209 ff., for an alternative description of the general problem.
Let us assume, for the sake of illustration, that we have only three samples for our approximation (cf. fig 2): We can thus represent these three values of the function, as well as the basis function values for these three input values, as vectors in a three-dimensional space (cf fig. 3), a function space. These basis functions will span a sub-manifold in that space, and the function to be approximated can be projected unto that submanifold. Our aim in learning will thus be to approach that submanifold as much as possible to the function/vector to be approximated, so that the projection A of the function F is as close to the function F itself as possible. If the network has then learned and generalized well, the approximated function will be close to the function itself also in other dimensions/for other inputs not in the training set.

2 Function Approximation via Non-orthogonal Bases

In a formal way, let F thus be a function to be approximated, from $\Re^n$ to $\Re$, a vector in a function space H^n. We want to approximate that function by another function $A(\gamma)$ which belongs to a submanifold of H^n, and which is the projection of F unto that submanifold. Let $B(\gamma)$ be the base of elementary functions g_μ of H^n spanning that submanifold, and let $B(\gamma)$ be parametrized by a parameter vector γ, $\gamma \in \Re^p, p \leq n$, $\Re^p$ thus a parameter space. $A(\gamma)$ will be the best approximation $A(\gamma_{opt})$ to F if the distance between them in H^n is minimal:

$$D(F, A(\gamma_{opt})) = min(D(F, A(\gamma))) \tag{1}$$

where D can be a distance such as the classical L^2, or another norm, as will be discussed below.
Note that that is exactly the task which a neural network such as a Radial Basis Function network [41] or Multi-layer perceptron [43] has to solve via learning: F is given by examples only, and within the class of functions which a network can represent, we look for the one closest to F. Such a multiple-layer neural network can be seen as an implementation of the algorithm above: Each hidden-layer node of the Multi-layer perceptron, is assigned an elementary function, with which to process the input; these functions span a submanifold of function space, and every function of that submanifold can be learned perfectly by the network, others only to a specific degree of approximation. *Learning* in such a network is the minimization of D; it is done by rotating and shifting the submanifold spanned by $B(\gamma)$ in such a way that it is as close as possible to F through a suitable modification of the parameters γ (Ref. fig. 3). It is sufficient to consider the submanifold of function space spanned by the dimensions of the input examples for the computation; the network will then generalize to the dimensions in between, if the examples have been chosen in an appropriate manner.
Note that the problem, seen from the viewpoint of numerical mathematics [1], can be seen as a problem belonging to the class of separable nonlinear least-squares problems [2] and it might be fruitful to compare our approch to solutions found there [22] [26] [17]. Furthermore, in contrast e.g. to Poggio et al. [41], we do not need to assume the existence of stabilizers in our cost function to ensure generalization for our solution, if the basis functions are smooth enough, as e.g. perceptrons with sigmoidal activation functions are. Expressing D as an energy functional of γ, and differentiating it w.r.t. γ, we obtain a set of differential equations for the modification of γ:

$$\frac{d\gamma}{dt} = -\frac{\delta D}{\delta\gamma} \tag{2}$$

Here we implement gradient descent directly. Obviously, conjugate gradient, or other methods, are equally suitable. By our experience, however, gradient descent implemented with Runge-Kutta, or via Euler, seems to be the most robust method. If D,

[1] Konrad Weigl is very grateful to Warren S. Sarle, SAS Institute Inc. Cary, NC 27513, USA, for pointing out the relationship to nonlinear least squares and providing us with the pointers to these articles.

[2] Except for the extension to a non-linear output neuron, which we can have optionally, see section "Other Metrics for Functional Spaces"

expressed as an energy, is not convex, other suitable methods such as Graduated Non-Convexity, Simulated resp. Mean-Field Annealing or other methods would have to be used.

In order to compute $\frac{\delta D}{\delta \gamma}$ above, we need to compute $A(\gamma)$ as a function of γ. We have:

$$A(\gamma) = \sum_{\mu} \sum_{x} A^{\mu} g_{\mu}(x, \gamma) \tag{3}$$

where the A^{μ}, scalars, are the expansion coefficients of F in the base $B(\gamma)$. We will insert the index γ from now on in the notation only where needed for comprehension. Since we already know the dependence of g_{μ} on γ, we need to compute only the A^{μ} and their dependence on γ. In order to do this, we can use for example the metric tensor of the base; further below we shall show other possibilities:
The so-called *metric tensor* of a base $B(\gamma)$ of functions g_{μ} (cf. fig. 4) is a square matrix defined by the computation of its components $g_{\mu\nu}$:

$$g_{\mu\nu} =< g_{\mu}, g_{\nu} >= \textstyle\sum_x g_{\mu}(x) g_{\nu}(x) \tag{4}$$

where the $<,>$ operator denotes the scalar product, computed over the input examples x, and obviously all of these variables depend on the parameter vector γ. Commutativity of $<,>$ implies symmetry of $g_{\mu\nu}$. Inversion results in the *contravariant metric tensor* $g^{\mu\nu}$, allowing us to compute the normalprojection $A(\gamma)$ of any function F unto the subspace spanned by $B(\gamma)$:

$$A(\gamma) \;\; = \;\; \textstyle\sum_{\mu} \sum_{\nu} A_{\mu} g^{\mu\nu} g_{\nu} \tag{5}$$

with $A_{\mu} =< g_{\mu}, F >$ the so-called *covariant components* and $A^{\mu} = g^{\mu\nu} A_{\nu}$ the *contravariant components*. The base $g^{\nu} = \sum_{\mu} g^{\mu\nu} g_{\mu}$ is called the *contravariant* or *biorthogonal* basis (See fig. 5), They are also known as *dual* [16], *adjoint* or *reciprocal* bases in literature [13]. Such bases are well-known, e.g. in the domain of Wavelet research, see e.g. Daubechies [9]. Daubechies uses the concept of *frames* to express the idea of an overcomplete, but still inversible base for a given space, i.e. too many basis vectors for the dimension of the space given. Researchers have also used these wavelet-frames to construct Neural Networks by other approaches than ours [35].
Note that, as we can see in fig. 4, these contravariant components A^{μ} are the weights on the connection between the hidden layer nodes and the outputlayer, respectively

the radial-basis-functions and the output layer. We thus do not have to compute these weights separately; rather, they are a direct result of the computation of the projection.

Figure 6 shows one iteration of the process (Refer below to subsection: The Algorithm Step by Step): Computation of covariant components and covariant metric tensor, inversion of the latter to obtain the contravariant metric tensor, matrix-vector multiplication of the contravariant metric tensor with the covariant components to obtain the contravariant components, and construction of the projection A.

Inverting a matrix generated via a scalar product is similar to an approach used in the class of so-called Least-Squares Problems in the field of model estimation [28], for the approximation of given data samples, where similar relationships are derived by analytical, non-geometrical means. The equations above are termed normal equations there and elsewhere [18], [34]. We can solve them by direct inversion, computing the contravariant metric tensor needed even using a neural-network approach such as the oscillatory approach proposed by Pellionisz as a cerebellar model [38], or by a Hopfield type network, minimizing a suitable error function: For example, let M be the matrix to be inverted; then we can write:

$$MM^{-1} = I \tag{6}$$

where M^{-1} is simply the inverse of M, and I is the identity matrix; we can thus solve problem above via minimization of the following

$$E = (MM^{-1} - I)^2 \tag{7}$$

where M is given, and the components m_{ij}^{-1} of M^{-1} are to be computed. The equation expresses the fact that for all components of the resulting matrix, the energy must be minimal: $\forall ik\imath\{1, N\}$, where N is the dimension of the matrix:

$$(\sum_j m_{ij} m_{jk}^{-1} - \delta_{ik})^2 = Min \tag{8}$$

Where Min is the energy to be minimized for each equation, and δ_{ik} is the Kronecker delta; it equals one if the indices are identical, and zero otherwise.

Differentiation of E w.r.t. $m_{\alpha k}^{-1}$ via summation over all the equation yields:

$$\frac{dE}{dm^{-1}_{\alpha k}} = \sum_i [2(\sum_j m_{ij} m^{-1}_{jk} - \delta_{ik}) m_{i\alpha}] \tag{9}$$

from which results the classical gradient descent equation to compute the optimal values for the components of the inverse M^{-1}:

$$\frac{dm^{-1}_{\alpha k}}{dt} \quad = \quad -\frac{dE}{dm^{-1}_{\alpha k}} \quad = \quad -2\sum_i [\sum_j m_{ij} m^{-1}_{jk} - \delta_{ik}) m_{i\alpha}] \tag{10}$$

Since this energy is convex for a symmetric and real-valued matrix such as the metric tensor, it converges always to a single minimum; if the matrix is ill-conditioned or singular, convergence has been observed to go to the pseudo-inverse. Using such a gradient approach can offer substantial increase in computation velocity: We need the inverse of a matrix at every iteration, knowing that the new inverted matrix will be close to the inversion computed the step before. At each step, we would thus initialize the Hopfield network with the value computed the step before, so that convergence should be extremely fast. Only at the first step do we have to initialize the network with the identity matrix, the null matrix or another suitable initialization. We could even use a dedicated Hopfield neural network, running continously, for the task.

However, instead of solving them via direct inversion as above, we can also use other neural-network approaches, as described further below.

Let now D be e.g.the distance using the L^2-norm; for the minimization, we use the distance squared, as an energy (Ref. fig. 1):

$$D(F, A(\gamma)) \quad = \quad \sum_x (F(x) - A(\gamma, x))^2 \quad = \quad \sum_x (F(x) - \sum_\nu A^\nu g_\nu(x))^2 \tag{11}$$

We then minimize D via gradient descent: Initially starting with a vector γ_0, we compute:

$$\gamma_{n+1} = \gamma_n - \left.\frac{dD(F, A(\gamma))}{d\gamma}\right|_{\gamma_n} \epsilon \tag{12}$$

where ϵ is a suitably chosen stepsize, via Runge-Kutta in the present case. Inserting for $\frac{dD(F,A(\gamma))}{d\gamma}$:

$$\frac{dD(F, A(\gamma))}{d\gamma} = \frac{d(\sum_x (F(x) - \sum_\nu A^\nu g_\nu(x))^2)}{d\gamma} \tag{13}$$

$$\frac{dD(F,A(\gamma))}{d\gamma} = \sum_x [-2(D(F,A(\gamma))\frac{d\sum_\nu A^\nu g_\nu(x)}{d\gamma}] \tag{14}$$

where:

$$\frac{d\sum_\nu A^\nu g_\nu(x)}{d\gamma} = \sum_\nu \sum_\mu g_\nu(x) g^{\nu\mu} \frac{dA_\mu}{d\gamma} + \sum_\nu \sum_\mu g_\nu(x) A_\mu \frac{dg^{\nu\mu}}{d\gamma} + \sum_\nu A^\nu \frac{dg_\nu(x)}{d\gamma} \tag{15}$$

The complexity of the algorithm above seems to increase with N^4, due to the term $\frac{dg^{\nu\mu}}{d\gamma}$, which includes a term $\frac{dg^{\nu\mu}}{dg_{\nu\mu}}$, i.e. the differentiation of each element of a N by N matrix w.r.t. each of the N x N terms of its inverse, where N is the number of basis functions; however, we can show that this term cancels out with the first term. Indeed, we have, $\forall i$, $\forall x$, since $\sum_i g^{\nu i} A_i = A^\nu$:

$$\sum_\nu g_\nu(x) g^{\nu i} \frac{dA_i}{d\gamma} + \sum_\nu g_\nu(x) A_i \frac{dg^{\nu i}}{d\gamma} = \sum_\nu g_\nu(x) \frac{dA^\nu}{d\gamma} \tag{16}$$

$$(F(x) - \sum_\sigma A^\sigma g_\sigma(x)) g_i(x) \frac{dA^i}{d\gamma} = F(x) g_i(x) \frac{dA^i}{d\gamma} - \sum_\sigma A^\sigma g_\sigma(x) g_i(x) \frac{dA^i}{d\gamma} \tag{17}$$

which implies:

$$(F(x) - \sum_\sigma A^\sigma g_\sigma(x)) g_i(x) \frac{dA^i}{d\gamma} = (F(x) g_i(x) - \sum_\sigma A^\sigma g_\sigma(x) g_i(x)) \frac{dA^i}{d\gamma} \tag{18}$$

$\sum_x F(x) g_i(x) = A_i$ and $\sum_x g_\sigma(x) g_i(x) = g_{\sigma i}$ by definition; this second equation implies, with the second term of the r.h.s. of equation above:

$$\sum_x \sum_\sigma A^\sigma g_\sigma(x) g_i(x) \quad = \quad \sum_x \sum_\sigma A^\sigma g_{\sigma i}(x) \quad = \quad A_i \tag{19}$$

We thus obtain:

$$\sum_x (F(x) - \sum_\sigma A^\sigma g_\sigma(x)) g_i(x) \tfrac{dA^i}{d\gamma} \quad = \quad \tfrac{dA^i}{d\gamma}(A_i - A_i) \quad = \quad 0 \tag{20}$$

The algorithm reduces thus to the form:

$$\begin{aligned} \tfrac{dD(F,A(\gamma))}{d\gamma} &= \tfrac{d(\sum_x (F(x) - \sum_\nu A^\nu g_\nu(x))^2)}{d\gamma} \\ \tfrac{dD(F,A(\gamma))}{d\gamma} &= -2 \sum_x (F - \sum_\nu A^\nu g_\nu(x)) \sum_\nu A^\nu \tfrac{dg_\nu(x)}{d\gamma}) \end{aligned} \tag{21}$$

The complexity of which increases with the complexity of the matrix inversion, i.e. at least N^2, where N is the number of hidden layer units resp. basis functions, resp. with another coefficient, if the inversion can be avoided.

Once the algorithm has converged, we have found the best approximation $A(\gamma_{opt})$ to F possible, for given distance metric, number and kind of elementary functions, assuming D is convex.

Fig. 3 and Fig. 6 show an example in three dimensions: Modifying γ moves g_0 and g_1, which moves the plane spanned by them, diminishing the distance between F and A_γ until it is minimal, giving us $A(\gamma_{opt})$.

At convergence, we store the set of coefficients A^μ and the parameter vector γ; given as input, for example for a two-dimensional input space, the values (x,y), the output will be $A(x,y) = \sum_\nu A^\nu g_\nu(x,y)$ or $A(x,y) = \sum_\nu A_\nu g^\nu(x,y)$.

3 Alternative Approaches to Direct Matrix Inversion

As we have seen above, one of the computations which we require, which we solved by an algorithm which is not very neural-network-like, is the computation of the $A^i = \sum_j A_j g^{ij}$, which corresponds to the solution of the normal equations of a least squares problem. There are a number of publications describing the solution of normal equations in applications related to Neural networks [12], [37], [36], [4], [16], even if the authors use other expressions for them. We can apply these directly to our problem as well. We will reformulate the approaches taken in our terminology. Note well that all of these authors operate in *input space* respectively *image space*, whereas we operate in *function space*. For a similar procedure, see [41]. The mathematical treatment is equivalent in all cases, however.

3.1 Computation of the Biorthogonal Base

The approach taken by Gaal [16] to reconstruct an image is to compute the dual/contravariant base to a base of Gabor filters [10] [11] which is an approach she postulates happens in the visual cortex; Computing the contravariant base would allow us to compute the contravariant metric tensor $g^{\mu\nu} =< g^\mu(x), g^\nu(x) >$, and thus the set of contravariant components $A^\nu = g^{\mu\nu}A_\mu$. She simply inverts the base of Gabor functions by the minimization of an energy function, similar to the following approach:

$$g_\nu(x) = \sum_\mu g_{\mu\nu} g^\mu(x) \tag{22}$$

giving us the following formulation for the energy:

$$E = \sum_x \sum_\nu (g_\nu(x) - \sum_\mu g_{\mu\nu} g^\mu(t,x))^2) \tag{23}$$

and thus the following expression for gradient descent:

$$\frac{dg^\sigma(t,x)}{dt} = -\frac{dE}{dg^\sigma(t,x)} = -\sum_x \sum_\nu [2(g_\nu(x) - \sum_\mu g_{\nu\mu} g^\mu(t,x)) g_{\nu\sigma}] \tag{24}$$

These equations are to be solved simultaneously for all σ.

Once we have the contravariant base, we can easily compute the contravariant metric tensor $g^{\mu\nu} =< g^\mu, g^\nu >$, and thus the contravariant components $A^\mu = \sum_\nu g^{\mu\nu} A_\nu$.

3.2 Computation of the Contravariant Components by Gradient Descent

However, a more efficient approach for our purposes is the one used by Daugman, Pattison and Pece [12], [37], [36]: They compute the contravariant components using the minimization of an energy expression which contains the contravariant components themselves, and implement the minimization via Neural Networks of different topologies:

The energy expression they use is in fact the normal equations themselves, which express the fact that covariant components of a function on a subspace must be identical to the contravariant components of the projection of that function unto the subspace: $\forall \mu$

$$< F, g_\mu >= \sum_x g_\mu(x) \sum_\nu A^\nu g_\nu(t,x) \tag{25}$$

implying the requirement:

$$< F, g_\mu >= \sum_\nu g_{\mu\nu} A^\nu(t) \tag{26}$$

which gives the energy:

$$E = \sum_{\mu}(< F, g_{\mu} > - \sum_{\nu} g_{\mu\nu} A^{\nu})^2 \tag{27}$$

and thus the following expression for gradient descent:

$$\frac{dA^{\sigma}}{dt} = -\frac{dE}{dA^{\sigma}} = 2\sum_{\mu}(< F, g_{\mu} > - \sum_{\nu} g_{\mu\nu} A^{\nu}(t)) g_{\sigma\mu} \tag{28}$$

The algorithm can be implemented as above, i.e. via gradient descent; since the energy is convex, a deterministic approach is sufficient.

3.3 Computation of the Contravariant Components by Recursive Least Squares

The third approach which we will present here as illustration is one which might be very suitable for large sets of examples, or even online learning of incoming data: We refer to Recursive Least Squares computation, such as is used in the Kalman filter or generally in System Identification [44]: We can compute the contravariant components iteratively, which avoids the need to invert the covariant metric tensor, except once at the start, to initialize the system: The approach is as follows (From [44]), using our terminology:

$$A^{\nu} = \sum_{\mu} g^{\mu\nu} A_{\mu} \tag{29}$$

Furthermore, we can write anytime the covariant metric tensor as a function of a variable t:

$$g_{\mu\nu}(t) = \sum_{s=0}^{t} g_{\nu}(s) g_{\mu}(s) \tag{30}$$

Once t is equal to the total number of dimensions of our system, thus in our case the number of examples we have, we obtain: $g_{\mu\nu}(t) = g_{\mu\nu}$.
We can also write:

$$g_{\mu\nu}(t) = g_{\mu\nu}(t-1) + g_{\mu}(t) g_{\nu}(t) \tag{31}$$

We define $g^{\mu\nu}(t)$ as the components of the matrix inverse of $g_{\mu\nu}(t)$, identical to our definition $g^{\mu\nu}$ as the components of the matrix inverse of $g_{\mu\nu}$, and generally:

$$A_\nu(t) = \sum_\mu g_{\mu\nu}(t) A^\mu(t) \tag{32}$$

and:

$$A^\nu(t) = \sum_\mu g^{\mu\nu}(t) A_\mu(t) \tag{33}$$

where

$$A_\mu(t) = \sum_{s=0}^{t} g_\mu(s) F(s) \tag{34}$$

Thus we can write:

$$A^\nu(t) = \sum_\mu g^{\mu\nu}(t)[A_\mu(t-1) + F(t) g_\mu(t)] \tag{35}$$

Inserting from above:

$$A^\nu(t) = \sum_\mu g^{\mu\nu}(t)[g_{\mu i}(t-1) A^i(t-1) + F(t) g_\mu(t)] \tag{36}$$

since we have:

$$g_{\mu\nu}(t-1) = g_{\mu\nu}(t) - g_\mu(t) g_\nu(t) \tag{37}$$

we can write:

$$A^\nu(t) = \sum_\mu g^{\mu\nu}(t)[g_{\nu i}(t) A^i(t-1) - g_\mu(t) g_i(t) A^i(t-1) + F(t) g\mu(t)] \tag{38}$$

Contracting $g^{\mu\nu}$ with $g_{\nu i}$ to δ_i^μ:

$$\sum_\nu g^{\mu\nu} g_{\nu i} = \delta_i^\mu \tag{39}$$

we obtain:

$$A^\nu(t) = A^\nu(t-1) - \sum_\mu \sum_i g^{\mu\nu}(t) g_\mu(t) g_i(t) A^i(t-1) + F(t) \sum_\mu g^{\mu\nu}(t) g_\mu(t)] \tag{40}$$

From which we obtain:

$$A^\nu(t) = A^\nu(t-1) + g^\nu(t)[F(t) - \sum_i g_i(t) A^i(t-1)] \tag{41}$$

which allows us to compute A^ν recursively, by letting t run through all the examples. In other words, we correct the old value $(A^\nu(t-1)$ to the new value $A^(t)$ by taking the difference between the function F to be approximated and its projection unto the submanifold $F(t)-\sum_i g_i(t)A^i(t-1)$, and projecting that difference unto the corresponding contravariant vector $g^\nu(t)$. That way, the projection of F unto the submanifold will be corrected until the error (Difference between F and projection) is orthogonal to all the contravariant base vectors.

Of course, we can thus, by including a suitable forgetting factor, also shift the base continuously while we compute the current $A^\nu(t)$; however, care must be taken not to move it too quickly, so that the influence of all the examples is felt. A random selection of the examples should help there. For the computation of the current $g^\nu(t)$ the contravariant metric tensor would have to be computed via inversion of the covariant metric tensor at every step, which puts an unnecessary computational burden on the system; in the appendix of [44] a lemma is proven which allows a recursive computation of the inverse of a matrix; we give the result here without proof:

$$g^{\mu\nu}(t) = g^{\mu\nu}(t-1) - \frac{\sum_i \sum_j g^{\mu i}(t-1) g_{ij}(t) g^{j\nu}(t-1)}{1 + \sum_\mu \sum_\nu g_\mu(t) g^{\mu\nu}(t-1) g_\nu(t)} \tag{42}$$

From that equation we obtain an equation for $g^\mu(t)$, also in [44]:

$$g^\mu(t) = \frac{\sum_\nu g^{\mu\nu}(t-1) g_\nu(t)}{1 + \sum_\mu \sum_\nu g_\mu(t) g^{\mu\nu}(t-1) g_\nu(t)} \tag{43}$$

Note that we contract the tensor to a scalar in the denominator, so that we divide here by a scalar instead of a matrix; that contraction, however, is again an operation in N^2; the multiple summation in the numerator of the iterative inversion goes obviously with N^4 per time step.

3.4 Comparison of Methods used to Compute Projection

What method we choose to compute depends obviously on the circumstances: the direct inversion is definitely the fastest if the number of nodes is not too big, since the complexity of the method increases with the complexity of the matrix inversion; that complexity should at least be N^2, where N is the number of nodes, and thus the

size of the matrix would be N x N; the Neural-network approach via the contravariant base is suitable if the nodes are linearly independent; the Neural-network approach via the computation of the contravariant components is suitable if we already have a neural-network architecture, furthermore it might be more suitable if we want the network to be fault-tolerant; the approch via Recursive least squares is interesting if we have little storage space w.r.t. the number of samples we want to train or if we simply want to train online, i.e. we process the data as it comes in, and delete it after having used it once; this approach of simply using real-world incoming data has already been used by Riemschneider [42] for the backprop-training on acoustic data. We ourselves have implemented three of the possible approaches: The direct inversion, computation of the contravariant components via gradient descent and Recursive Least Squares; the direct inversion was in our case the fastest implementation by far. A gradual computation of the contravariant components, via RLS or gradient descent, has obviously the advantage that we can have two concurrent operations: the computation of the components, on a fast timescale, and the minimization of the distance, on a slower timescale: Since the basis moves slowly, the contravariant components will only change slowly, so that a concurrent operation should pose no problems.
Note that obviously, any other classical algorithm for matrix inversion can be used, to invert the metric tensor, or directly the matrix of the function components of the base, for example Singular Value Decomposition, Gauss-Seidel, LU, et al.

3.5 The Algorithm Step by Step

In this section we will explain the simplest version of the algorithm step by step. Refer to figure 5.

Let $x_k, k \in \{1, .., M\}$ be the set of input examples, where M is the number of examples. if we have only one input line, x_k is a scalar, otherwise, it is a vector of values, one per input line. Let $F(x_k)$ be the wanted output to be associated with x_k, i.e. the output we would want the network to produce for the input given being x_k. We also have thus M such wanted output samples, $k \in \{1, .., M\}$. In the case shown in figure 5, the number of examples is 3; we have thus 3 dimensions, tagged by x_0,x_1,x_2.

The first step is initialization: Like any other network, we initialize weights and parameters; the speed of convergence depends critically on the initial weights, with differences up to factor five and even more observed. A random initialization is helpful.

Something which might be critical for small number of examples, but not for real-world problems, is the required linear independence of the basis functions in sample space; if they are not linearly independent in sample space, the metric tensor is singular and then only invertible by Singular Value Decomposition, Miller-Penrose Pseudo-inverse, or gradient descent as above. Random initial weights should take care of the problem.

The steps in the learning loop, thus per iteration, are then the following:

1) Compute the output values $g_i(x_k)$ of all the filters/nodes/hidden-layer-neurons i for all the input samples $x_k, k \in \{1, .., M\}$;

2) Multiply pairwise $g_i(x_k)$ with $g_j(x_k)$, and sum over all these products $g_i(x_k)g_j(x_k)$, $k \in \{1, .., M\}$; this gives us the scalar g_{ij}. Do this for all filters i, j; These scalars g_{ij} are the components of the covariant metric tensor, which is a symmetric and real matrix. Invert the matrix by any of the methods mentioned above, or another one of your choice. This gives us the contravariant metric tensor.

3) Multiply all the output samples given $F(x_k), k \in \{1, .., M\}$ with the corresponding filter outputs $g_i(x_k), k \in \{1, .., M\}$ computed above. Sum over all these products $F(x_k)g_i(x_k)$, $k \in \{1, .., M\}$; this gives us the covariant component A_i. Repeat for all the filters.

4) Multiply the contravariant metric tensor obtained above, which is a matrix, with the vector formed by all the covariant components A_i; this gives us the vector of contravariant components A^i. Multiplying now all the filters $g_i(x_k)$ with the corresponding A^i, and summing up over all the indices i, gives us the function approximation for the network and input sample x_k, called $A(x_k)$:

$$A(x_k) = \sum_i A^i g_i(x_k) \tag{44}$$

Thus we can now compute the distance D:

$$D = \sum_{k=1}^{M} (F(x_k) - A(x_k))^2 \tag{45}$$

5) We have to differentiate D with regards to the parameters/weights of the nodes/basisfunctions; thus we need, by the chain rule, for example, to differentiate w.r.t. the parameters of

the filter j, called $params_j$:

$$\frac{dD}{dparams_j} = \sum_x (F(x_k) - A(x_k))^2 \frac{d(F(x_k) - A(x_k))^2}{dparams_j} \tag{46}$$

implies:

$$\frac{dD}{dparams_j} = \sum_x 2(F(x_k) - \sum_i A^i g_i(x_k))(A^j \frac{dg_j(x_k)}{dparams_j} \tag{47}$$

$\frac{dg_j}{dparams_j}$ depends on the type of basis function used, obviously; for a sigmoid, it is for example: [43] $\frac{dg_j(x_k)}{dparams_j} = g_j(x_k)(1 - g_j(x_k))x_k$ for the computation of a non-bias input, and $\frac{dg_j(x_k)}{dparams_j} = g_j(x_k)(1 - g_j$ to cmpute the bias weight. 6)Once we have computed $\frac{dD}{dparams_j}$ for all parameters, we can compute:

$$\frac{dparams_j}{dt} = -\frac{dD}{dparams_j} \tag{48}$$

and compute one step via gradient descent or other; then we reiterate the loop with steps 1) - 6), until the minimal distance has been found.
Of course, if the distance, seen as an energy, is not convex, other methods, such as Simulated Annealing, must be used.
Our algorithm has intrinsically *no* learning parameters or learning rates to set; the only decision which might be critical is the initial set of parameter values. Depending on the method chosen for minimization, the learning step would have to be set accordingly; however, we have had very good results with variable stepsizes computed online, similar to the approach taken in [34] for the computation of the next stepsize for the numerical implementation of differential equations via Runge-Kutta. We have implemented an adaptive stepsize depending on the rate of descent. But such details do obviously not influence the stability of the algorithm in any way.

4 Other Metrics for Functional Spaces

Obviously, we are not restricted in our discussion to linear distance metrics in our system, nor to linear domain spaces; as an example for another metric, we could take the symmetrical version of the Kullback-information, or cross-entropy, as a metric for a space of probability functions [8], which would transform our system to a parameter estimator for given histogram distributions, for example:

$$D = \int_x F(x) log(\frac{F(x)}{\sum_\mu A^\mu g_\mu(x)})dx + \int_x \sum_\mu A^\mu g_\mu(x) log(\frac{\sum_\mu A^\mu g_\mu(x)}{F(x)})dx \qquad (49)$$

where D is the distance to be minimized, x is the domain of the function, F(x) again the function to be approximated, and $A^\mu g_\mu(x)$ the approximated function. However, we do not know if the classical scalar product $<,>$, which is the basis for our algorithm, can actually be applied unmodified in a space with such a metric, or whether other concepts from Riemannian spaces [30] would have to take its place. The fact that we would not be limited to exponential families thus would allow us to disregard partition function problems, which e.g. Kulhavy [27] has to deal with explicitly, consider non-parametric estimation models, consider probabilities $p(x)$ such that $p(x) = 0$ for some x, linear combinations of any functions, etc.

We can integrate non-linear output neurons, by changing the distance to

$$D = f(F - h(\sum_\sigma A^\sigma g_\sigma)) \qquad (50)$$

where h is the activation function of the output neuron, for example the well-known sigmoidal function, and f is one of the metrics discussed above, for example the L^2-norm. However, then, while the requirements concerning the hidden-layer functions remain unchanged, the output neuron, if it is to be nonlinear, is required to compute a function which is invertible, similar to the requirement of Biegler-Koenig and Baermann for all their neurons [6]: The covariant components A_ν will then not be the scalar products $< F, g_\nu >$ of the target function F with all the functions $g_\nu, \forall \nu$ computed by the hidden layer, but the scalar products $< h^{-1}(F), g_\nu >$. The hidden layer will then learn to approximate that function $h^{-1}(F)$ optimally, so that, once the output neuron has computed the function $h(h^{-1}(F))$, the network has again optimally learned the function F which, obviously, must be in the range of h. We did not note any significant advantage or disadvantage of using a non-linear output neuron in our studies, but haven't studied the question further.

To what extent we can integrate concepts such as non-linear submanifolds to the paradigm remains a study to be executed in the future.

A further characteristic of our system is that if the function F changes in time, e.g. is a histogram of samples from the evolution of a non-stationary random process, the system should be able to follow, assuming a change which is slow w.r.t. the intrinsic dynamics of the network.

5 Some Results and Further Features of Dynamical Non-orthogonal Bases

For the results in fig. 7 through 15, we generated random initial parameter values, from the pseudo-random number generator integrated in our system. The distribution of values output by the generator is flat from zero to one; we multiplied this output by a multiplication constant of our choice, indicated in the figure captions. Then we set the average to zero by substracting 0.5 for the parameter values for the sigmoid functions, as well as for the means of the gaussians; for the standard deviation of the latter, we used the multiplication constant without subsequent averaging, to avoid negative values for the standard deviation, and added 1.0, in order to avoid too small values for the standard deviations which might have caused very small function values at the sites of the samples. The values of the multiplication constant are indicated in the figure captions.

In the figures 7 through 12 we show the behaviour of the algorithm for different base functions and number of neurons; We have used the classical XOR-problem for illustration throughout this series. Obviously, as we can shall see in fig. 15, using a set of four samples for the XOR will provide for instantaneous learning with four and more basis functions, since any four linearly independent vectors will form a basis for a four-dimensional vector space. Where appropriate, we have therefore added four more samples to the input to generate more input samples, and thus a higher-dimensional vector space.

We show in fig. 13 and 14 that our system breaks down gracefully when neurons/elementary functions are taken away; The top left figure shows the converged network; the figure next to it shows what happens when a neuron dies: Immediately, the metric tensor gets recomputed; if there is enough overlap between the neuron that died away and the remaining ones, the latter can correct the error to a large extent after only one iteration, needed to recompute the metric tensor. If such an overlap does not exist, as seen in fig. 13, the recovery takes longer. On a longer timescale, the submanifold gets adjusted optimally again, in such a way that it is again as close as possible to the function to be approximated.

Figure 15 shows in the first two rows that such systems will as well integrate further neurons/elementary functions immediately without decreasing the quality of the approximation. It needs just one iteration, in order to be able to compute the new metric tensor. If we then insist on a smaller error, it will converge to the new optimal subspace, with the new optimal approximation. This allows for an efficient approximation to a function, adding elementary functions until the desired quality of approximation is reached, or until the distance converges to a new minimum.
The third row of figure 15 shows what we have mentioned above: The predicted behaviour for four samples, if we have four basis functions: The network converges immediately, i.e. after only one iteration.
Figures 16 and 17 show some comparisons in convergence speed for backpropagation versus the dynamic base paradigm.
The testing was done under the following conditions:
-we averaged over 20 runs each, having implemented both algorithms using the same software, i.e. standard gradient descent without any refinings;
-before each run, the weights were reinitialized to identical values for both algorithms;
-the examples were created by adding noise of uniform distribution, mean 0.0 and 0.01 max. deviation to the XOR-examples;
-Stopping criterion for convergence was a maximum tolerated error threshold of less than 0.05 for every example.
Note that, as already mentioned, when the number of examples is in the order of magnitude of the number of neurons, and the parameter values are initialized randomly and narrowly around a common mean, the functions computed by hidden-layer neurons are barely independent, so that the matrix defining the base is ill-conditioned. For the results shown in figure 16, we have therefore initialized the parameters with a mean of zero for every one out of two parameters, and a mean of 0.3 for the others, to ensure sufficient stability. Maximum deviation from these mean values was 0.005 either way for the random initialization.
For the results shown in figure 17, we have initialized all the parameters with a mean of 0.0, but with a maximum deviation of +/- 1.0. The result was that the systems converged much faster on the average to the best solution, with however the risk of

falling unto a local saddle in the energy landscape, from which exit can be tedious: As can be seen from the figure 17, one run with 3 neurons learning 12 samples pushed up the average, actually taking a whole 9.96 seconds to learn, more than one order of magnitude higher than the other runs, and four times as high as the backpropagation run with the same initial weight values.

In real world problems, of course, the number of examples is high enough to ensure stability: We never had any problem there, initializing with parameters with mean 0.0 and an even distribution with maximal values +/-0.05 to +/- 0.1.

Note that with projection learning, and so few examples, time to convergence does not increase with the number of examples; this is presumably due to the fact that, with so few examples, the major part of the computing time is spent for the inversion of the metric tensor, which is not a function of the number of examples. The time for the inversion rises steeply with the number of hidden-layer neurons in the system; this, however, does not show up in the total computation time because the submanifold spanned by more hidden-layer neurons is wider in dimensionality and thus more likely to be close to the function to be approximated. Thus we need far fewer iterations to converge if the number of hidden-layer neurons is higher. This also applies if the number of examples is much higher.

A potential drawback not to be forgotten is the question of the lacking ability to generalize of a converged network, disadvantage which might occur with too many hidden-layer neurons.

6 Discussion, Conclusion and Outlook

Crucial, again, for rapid convergence seems to be the initial choice of parameter values: Note that in figure 8, we show the behaviour of our system if we initialize with large weights; the system with only two hidden-layer neurons with sigmoidal activation function had convergence problems if the initial weights were too small, respectively it took it too long to get away from the initial set of parameter values, as we can see similarly in figure 10: The majority of the time, the parameter values were small, which can be seen in the linearity of the function approximated; only towards the end do the weights increase. Note that weights which are too small can also cause a problem in implementations, because then the weights from hidden layer to output layer have to compensate for them, resulting in large coefficients which might require high precision for correct implementation.

Note that in general, generalization is good. We have not added any noise to the samples of figures 7 through 15, to check speed of convergence, as we have done in our past papers [48], [47], [46]. Thus the behaviour between the sample points is uncontrolled, except obviously by the intrinsic smoothness of the basis functions. Here further work is needed; some initial research has been done by Kattentidt [21], who looked at the sampling intervals needed in terms of resolution, to be expressed by a tensor field, and Pati et al. [35], who studied the properties of wavelets for optimal approximation.

We believe that the view of neural networks as bases in function space, besides the algorithm we present in the present paper, will shed new light on current problems with Neural Networks, such as convergence, number of hidden layers and nodes needed, choice of examples, etc., beyond already existing work done e.g. by Mitchell [31]. It should also provide a new viewpoint from which to analyze existing and classical learning algorithms such as Backpropagation (A.k.a. Generalized Delta Rule in [43]). It might also be a paradigm that unifies the concept of Artificial Neural Networks, seeing Radial Basis functions, Gabor functions, sigmoidal functions and others as exponents of a general model of networks.
Also, searching further among the numerous other algorithms existing in the field of numerical mathematics [22] [26] [17] would certainly furnish yet further viewpoints and further approaches to the solution of the problem.
We are currently applying our network to a classification task, namely texture classification from aerial images, and the results look promising. On the theoretical side, we will look at the implementation of networks with more than one hidden layer, and more than one node in the output layer. The latter generalization should only require an extension of the distance function, while the former is less trivial. Futhermore we shall consider the synthesis of basis functions custom-made for specific applications. On another tack, we will see how far we can integrate Neural Models such as Kohonen topological maps [25] and Weightless Systems by Aleksander [32] into the paradigm. Also, while the integration of the learning of the metric tensor describing the non-orthogonality of the input space of a classical Radial Basis Function network, as Poggio has done for radial hyperbasis functions of fixed size and shape [41], should pose no problem, we have not yet implemented it.

Generally, we believe that the use of tools from differential geometry, along with other non-euclidian geometries, such as fractal geometries, [39], still has a lot to bring to field of neural networks and the study of the brain [38], [39], along with approaches from statistics by Amari [1], from Physics by Haken [19], and works built upon those, such as [20],[27], [15] et al. It should also be fruitful to interpret work already done in the field, e.g. interpretation of Multi-layer perceptrons by Kattentidt [21], the paper by Anderson et al. on motion detectors in the primate visual cortex, [4] and others. Last but not least, it is a very suitable tool for more speculative approaches to perception and cognitive processes, such as [23] [24] [49].

References

[1] Amari, S-I, Dualistic Geometry of the Manifold of Higher-Order Neurons, in: Neural Networks, Vol. 4, 1991, pg. 443-451

[2] Amari, S-I, Information Geometry of Boltzmann Machines, IEEE Trans. on Neural Networks, Vol. 3, No. 2, March 1992, pg. 260-271

[3] Amari, S-I, Differential-Geometrical Methods in Statistics, Lecture Notes in Statistics 28, Springer Verlag, 1985

[4] Anderson, J., A., Rossen, M. L., Sicuso, S., R., Sereno, M.E., Experiments with Representation in Neural Networks: Object Motion, Speech, and Arithmetic, in: Synergetics of Cognition, H. Haken, Ed., Springer, Berlin.

[5] Baermann, F., and Biegler-Koenig, F., On a Class of Efficient Learning Algorithms for Neural Networks, in: Neural Networks, Vol. 5, No. 1, pp. 139-144, 1992, Pergamon Press

[6] Biegler-Koenig, F., and Baermann, F., A Learning Algorithm for Multi-Layered Neural Networks Based on Linear Least Squares, in: Neural Networks, Vol. 6, o. 1pp. 127-131, 1993, Pergamon Press

[7] Caianiello, E., R., Systems and Uncertainty: A Geometrical Approach, in: Topics in the General Theory of Structures, Caianiello, E.R., and Aizerman, M.A., eds., Reidel Publishing Company, pg. 199-206, (1987)

[8] Caianiello, E.R., Quantum and Other Physics as Systems Theory, La Rivista del Nuovo Cimento, della Societa Italiana di Fisica, 1992, Editrice Compositon, Bologna, Italy

[9] Daubechies, I., Ten Lectures on Wavelets, SIAM, Philadelphia, Pennsylvania, 1992

[10] Daugman, J., Six Formal Properties of Two-dimensional Anisotropic Visual Filters: Structural Principles and Frequency/Orientation Selectivity, in: IEEE trans. SMC, vol. SMC-13, Sept/Oct 1983

[11] Daugman, J., Uncertainty Relation for Resolution in Space, Spatial Frequency, and Orientation Optimized by Two-dimensional Visual Cortical Filters, in: J. Opt. Soc. Am. A., Vol. 2, No. 7/July 1985

[12] Daugman, J., Complete Discrete 2-D Gabor Transform by Neural Network for Image Analysis and Compression, in: IEEE trans. ASSP, vol. 36, No. 7, July 1988, pg. 1169-1179

[13] Duschek, A., Vorlesungen ueber Hoehere Mathematik, Vol. III, pg. 233-256, Springer, Wien, (1953)

[14] Duschek, A., Vorlesungen ueber Hoehere Mathematik, Vol. II, pg. 164, Springer, Wien, (1950)

[15] Fuchs, A., and Haken, H., Pattern Recognition and Associative Memory as Dynamical Processes in a Synergetic System, in: Biol. Cybern. No 60, pg 17-22 (1988)

[16] Gaál, G., Population Coding by Simultaneous Activities of Neurons in Intrinsic Coordinate Systems Defined by their Receptive Field Weighting Functions, in: Neural Networks, Vol. 6, pp. 499-515, 1993

[17] Golub, G.H., and Pereyra, V., The Differentiation of Pseudo-inverses and Non-linear Least-Squares Problems Whose Variables Separate, SIAM, J. Numer. Analysis, Vol. 10, No. 2, April 1973, pg. 413-431

[18] Golub, G.H., and Van Loan, C.F., Matrix Computations, Oxford, North Oxford Academic, 1983

[19] Haken, H., Information and Self-Organisation, pg. 153 ff. Springer, 1988

[20] Hanzon, B., New Results on the Projection Filter, Dept. Econometrics, Free University of Amsterdam, De Boelelan 1105, 1081 HV Amsterdam, the Netherlands

[21] Kattentidt, Holger, Neural Networks and Neural Control, in: Workshop on Neural Networks, Huening H. et al. eds, Aachen 1993, Verlag der Augustinus Buchhandlung, ISBN 3-86073-140-8

[22] Kaufman, L., A Variable Projection Method for Solving Separable Nonlinear Least Squares Problems, BIT 15 (1975), pg. 49-57

[23] Kintzel, U., A Graviational Approach to Cognition, Human Thinking, Euskirchen, Germany

[24] Kintzel, U., A Relativistic Model of Human Cognition, in: Proc. WOPPLOT '92, Workshop on Parallel Processing, Logic and Technology, Springer LNCS series, to be published.

[25] Kohonen, T., Self-organisation and Associative Memory, Springer Series in Information Sciences, Springer Verlag, 1988

[26] Krogh, F.T., Efficient Implementation of a Variable Projection Algorithm for Nonlinear Least-Squares Problems, Numerical Mathematics, Communications of the ACM, March 1974, Vol. 17, Number 3.

[27] Kulhavy, R., Differential Geometry of Recursive Adaptive Estimation, Institute of Information Theory and Automation, Czechoslovak Academy of Sciences, 18208 Prague 8, Czechoslobakia

[28] Lawson, L.L., and Hanson, R.J., Solving Least Squares Problems, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974

[29] Lee, S., and Kil, R.M., A Gaussian Potential Function Network With Hierarchically Self-Organizing Learning, in: Neural Networks, Vol. 4, pp. 207-224, 1991, Pergamon Press

[30] Misner, C.W., Thorne, K.S., and Wheeler, J.A., Gravitation, Freeman, W.H. and Co., Pub., New York, (1970)

[31] Mitchell, John, A Geometric Interpretation of Hidden-Layer Neurons in Feedforward Neural Networks, in: Network, 3 (1992), pg. 19-25

[32] Mrśiĉ-Flögel, J.: Aspects of Planning with Neural Systems, PhD-thesis, Imperial College, London, 1993

[33] Mrugala, Ryszard, Information Geometry of Nonideal Gases, Information Technology Conference, Prague, 1986

[34] Press, W.H., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T., Numerical Recipes in C, Cambridge University, Cambridge, New York, Melbourne, 1988

[35] Pati, Y.C., and Krishnaprasad, P.S., Analysis and Synthesis of Feedforward Neural Networks Using Discrete Affine Wavelet Transformation, in: IEEE Trans. on Neural Networks, Vol. 4, No. 1, January 1993

[36] Pattison, Tim R., Relaxation Network for Gabor Image Decomposition, Biological Cybernetics, 67, pg. 97-102, 1992

[37] Pece, Redundancy Reduction of a Gabor Representation: A Possible Computational Role of Feedback from Primary Visual Cortex to Lateral Geniculate Nucleus, Proc. ICANN92, Brighton, North-Holland, Elsevier, Publishers, 1992

[38] Pellionisz, A., LLinás, R., Tensor Network Theory of the Metaorganisation of Functional Geometries in the Central Nervous System, in: Neurocomputing 2, Anderson, J.A., Pellionisz, A., and Rosenfeld, E., eds, the MIT-press, pg. 351-356 (1988)

[39] Pellionisz, A., Discovery of Neural Geometry by Neurobiology and its Utilization in Neurocompute Theory and Development, in Proc. ICANN Helsinki, Elsevier Pub. North Holland, pg. 485-493, (1991)

[40] Peretz, Richard, Opérateurs Analogiques Spéciaux, in 2èmes Journées Internationales de Calcul Analogique/2nd International Analogue Computation

Meetings, Strasbourg, 1958, publishers Presses Académiques Européennes Bruxelles, 1959, Masson et Cie, Paris

[41] Poggio, T., and Girosi, F., Networks for Approximation and Learning, in Proc. IEEE, Vol. 78. No. 9, Sept. 1990, pg. 1488

[42] Riemschneider, K.-R., Geraueschklassifikation mit Neuronalen Netzen, Universitaet der Bundeswehr Hamburg, Technische Informatik, Hamburg, Germany; also personal communication

[43] Rumelhart, D.E., McClelland, J.L., et al., Parallel Distributed Processing, Vol. 1, MIT-press, 1986

[44] Soederstroem, T., and Stoica, P., System Identification, Prenctice Hall, New York et al., 1989, pg. 320 ff.

[45] Weigl, K., and Berthod, M., The Restauration of Images with a System of Locally-coupled Limit-cycle Oscillators, in: Proc. of the 9th IAICV, Ramat Gan, Israel, 1992

[46] Weigl, K., and Berthod, M., Metric Tensors and Dynamical Non-Orthogonal Bases: An Application to Function Approximation, in Proc. WOPPLOT 1992, Workshop on Parallel Processing: Logic, Organization and Technology, Springer Lecture Notes in Computer Sciences, to be published.

[47] Weigl, K., and Berthod, M., Non-orthogonal Bases and Metric Tensors: An Application to Artificial Neural Networks, in New Trends in Neural Computation, Proc. IWANN'93, International Workshop on Artificial Neural Networks, Springer Lecture Notes in Computer Sciences, vol. 686.

[48] Weigl, K., and Berthod, M., Non-orthogonal Bases and Metric Tensors, in: Workshop on Neural Networks, Huening H. et al. eds, Aachen 1993, Verlag der Augustinus Buchhandlung, ISBN 3-86073-140-8

[49] Weigl, K., Activity Curvature: A new Approach to Perception, in: Proc ICANN 93, International Conference on Artificial Neural Networks, Amsterdam, 1993.

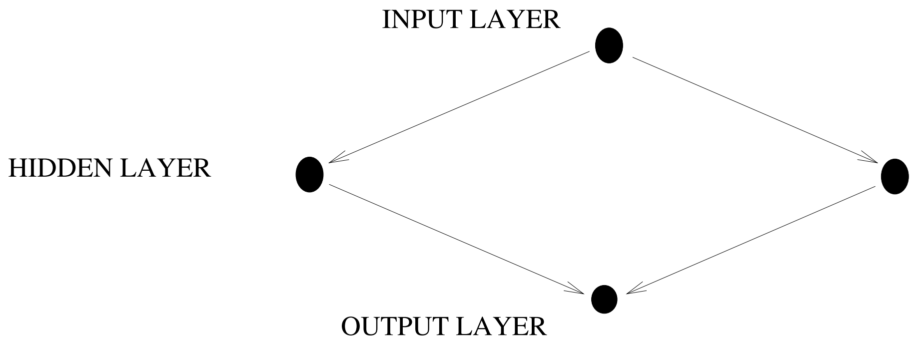

Figure 1: A simple network for illustration

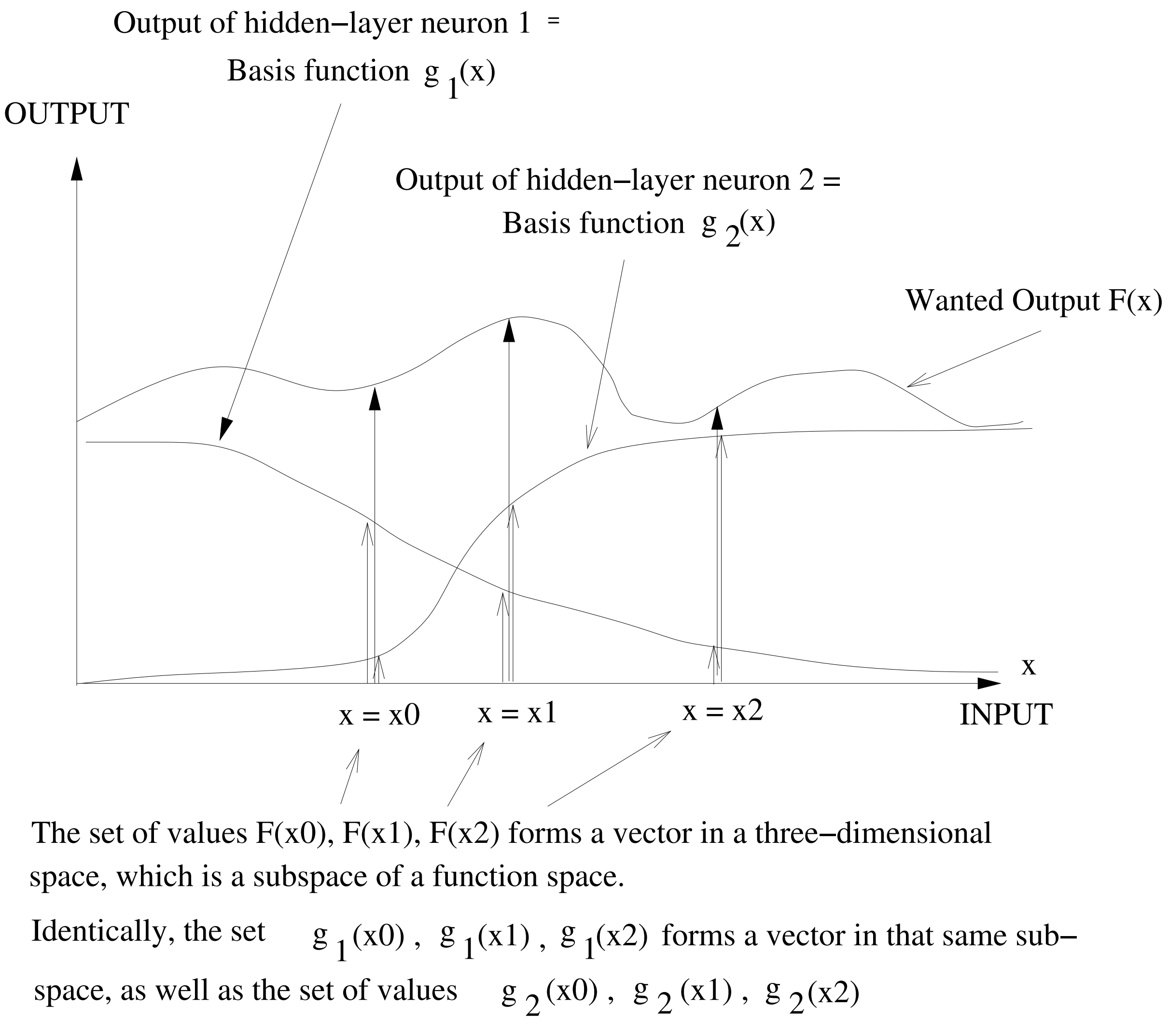

Figure 2: A scalar function of one variable

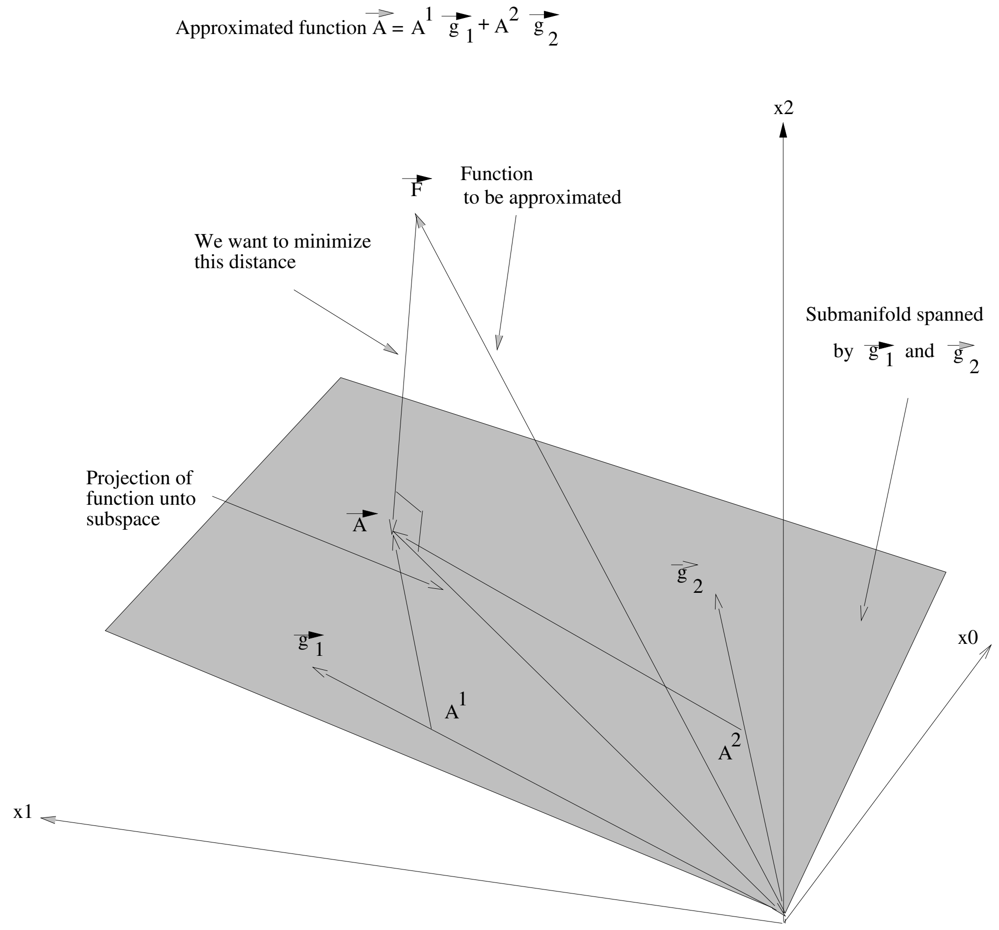

Figure 3: A representation in a three-dimensional function space

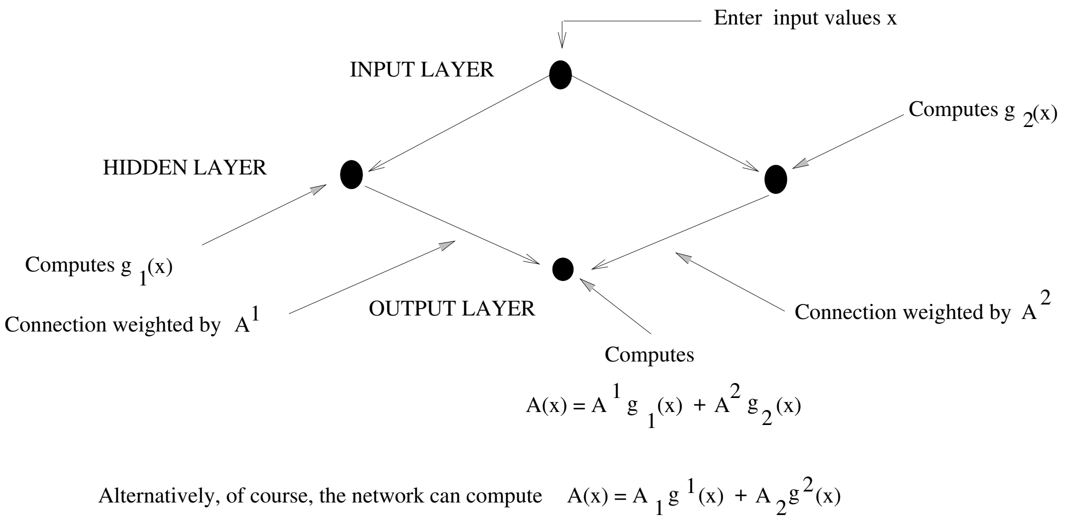

Figure 4: Mapping between the terms resp. structures used in the present paper and terms resp. structures used in the field of Neural Networks

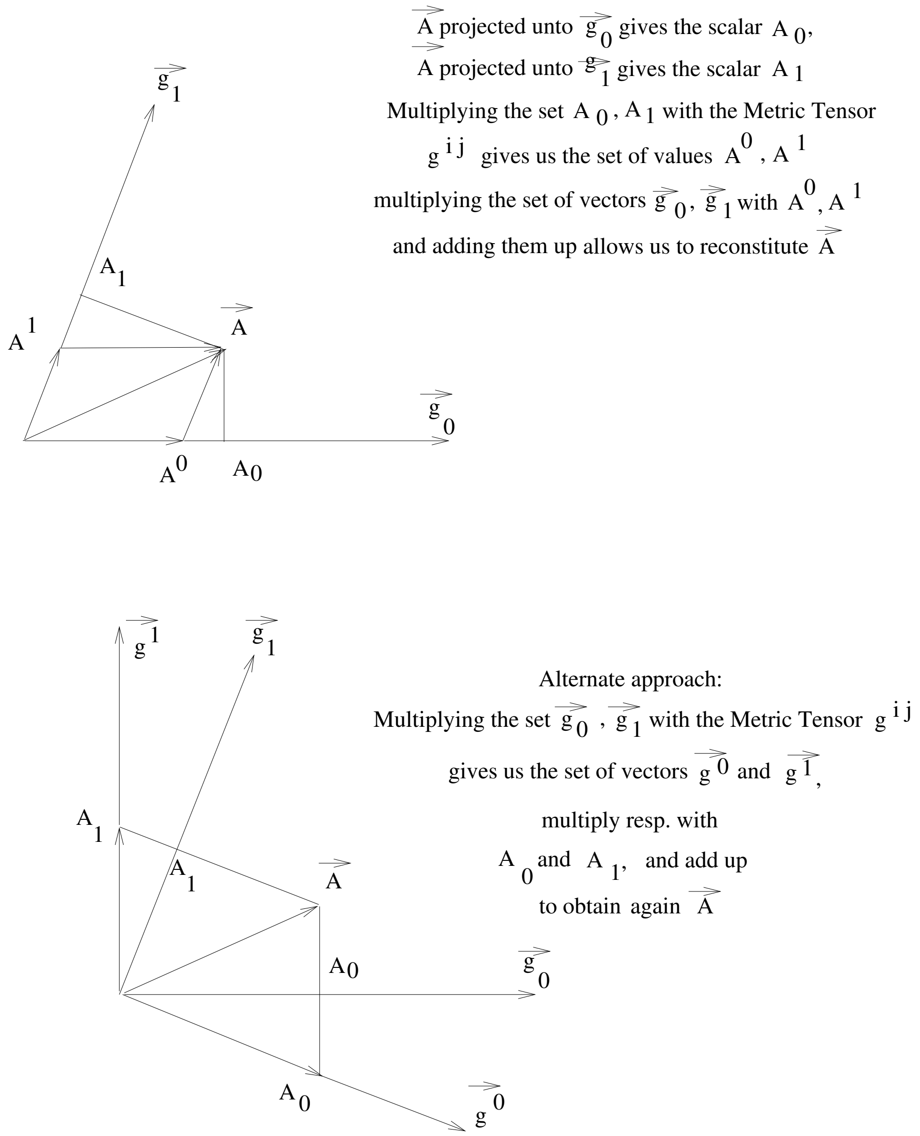

Figure 5: A covariant and a contravariant base in a two-dimensional space

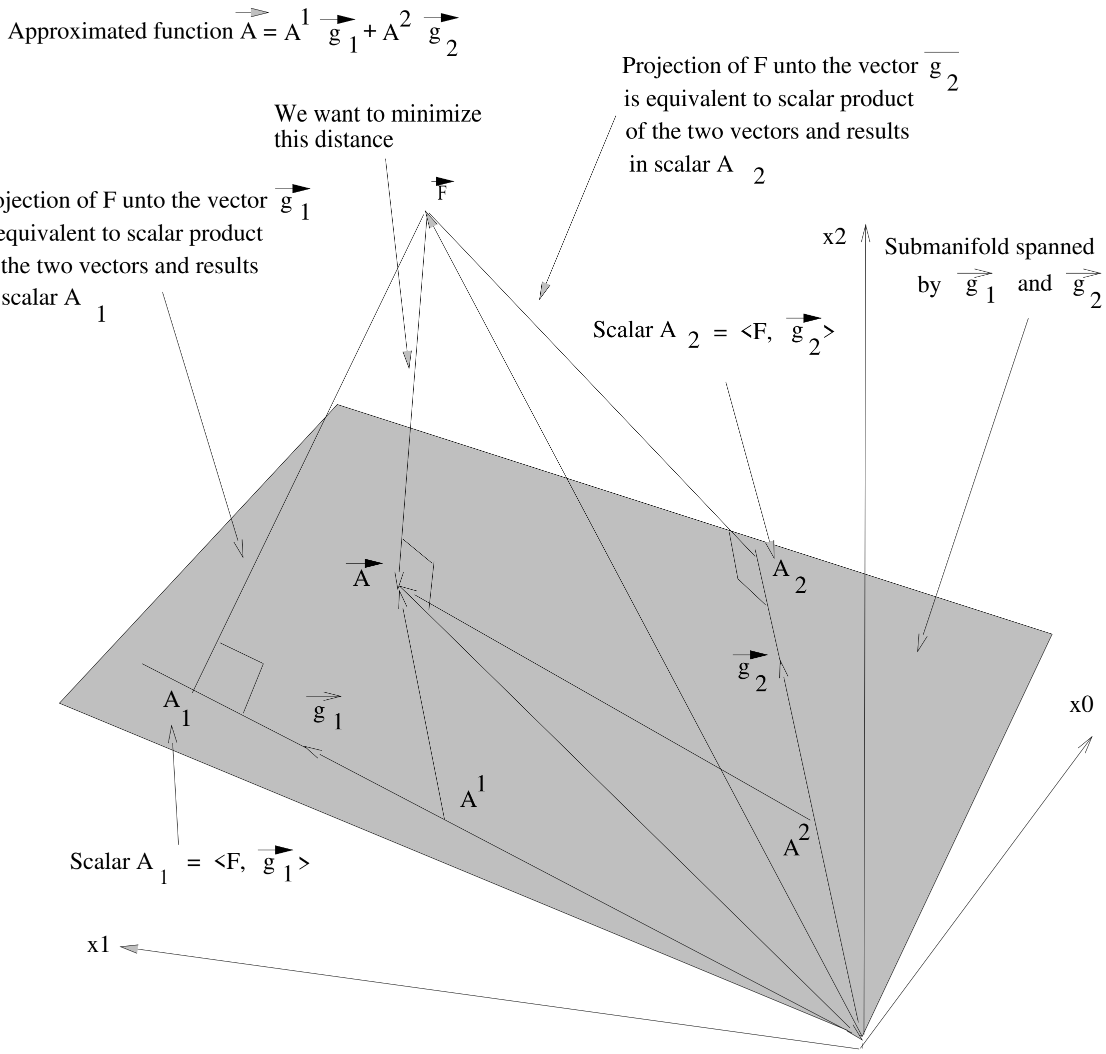

Figure 6: Projection of the function unto the two-dimensional subspace spanned by the base; computation via the scalar products $A_1 =< F, g_1 >$ and $A_2 =< F, g_2 >$

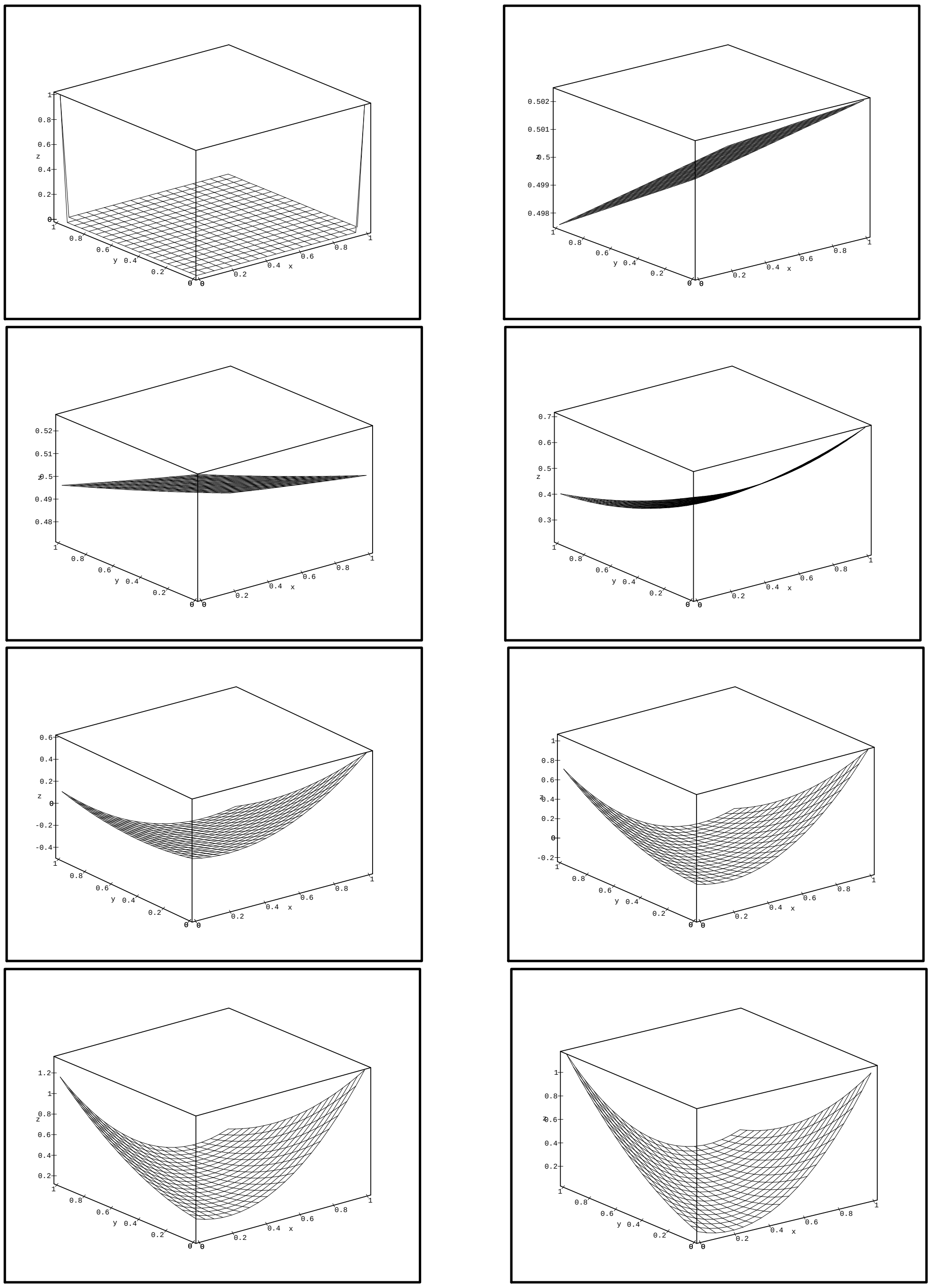

Figure 7: Evolution of the system towards an optimal approximation; two sigmoidal filters, four samples; Top left image shows the original XOR-function; the remaining images, top left linewise to bottom right, show the evolution of the system. Data: 188 iterations, total time 760 msecs on a Sparc 10, multiplier for random parameter initialization 0.01. For text concerning all results, see section “Some Results and Further Features of Dynamical Non-orthogonal Bases”

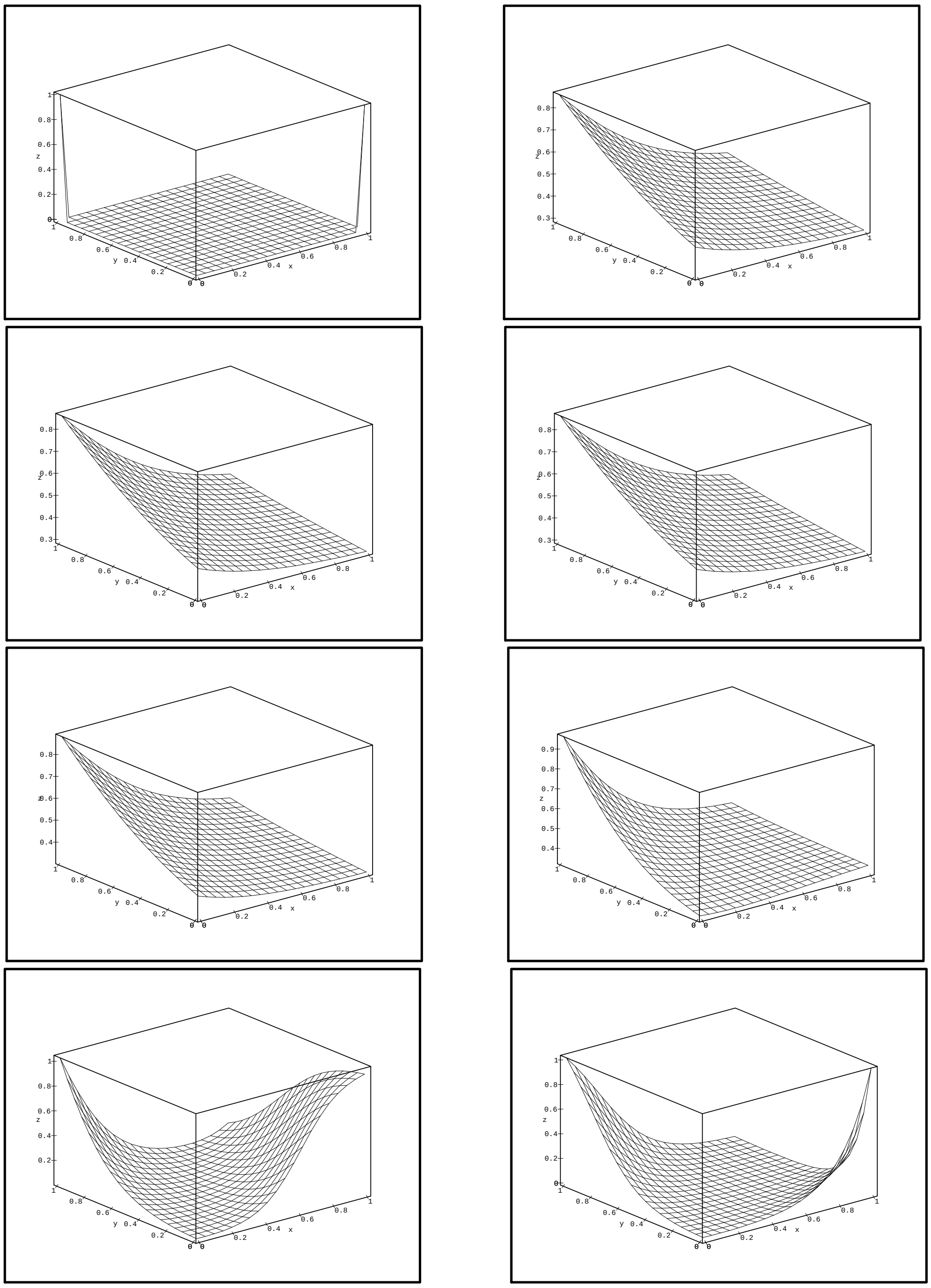

Figure 8: As before, but 8 samples (Only four shown because of lacking resolution); Top left image shows the original XOR-function; the remaining images, top left linewise to bottom right, show the evolution of the system. Data: 21 iterations, total time 170 msecs on a Sparc 10, multiplier for random parameter initialization 10.0

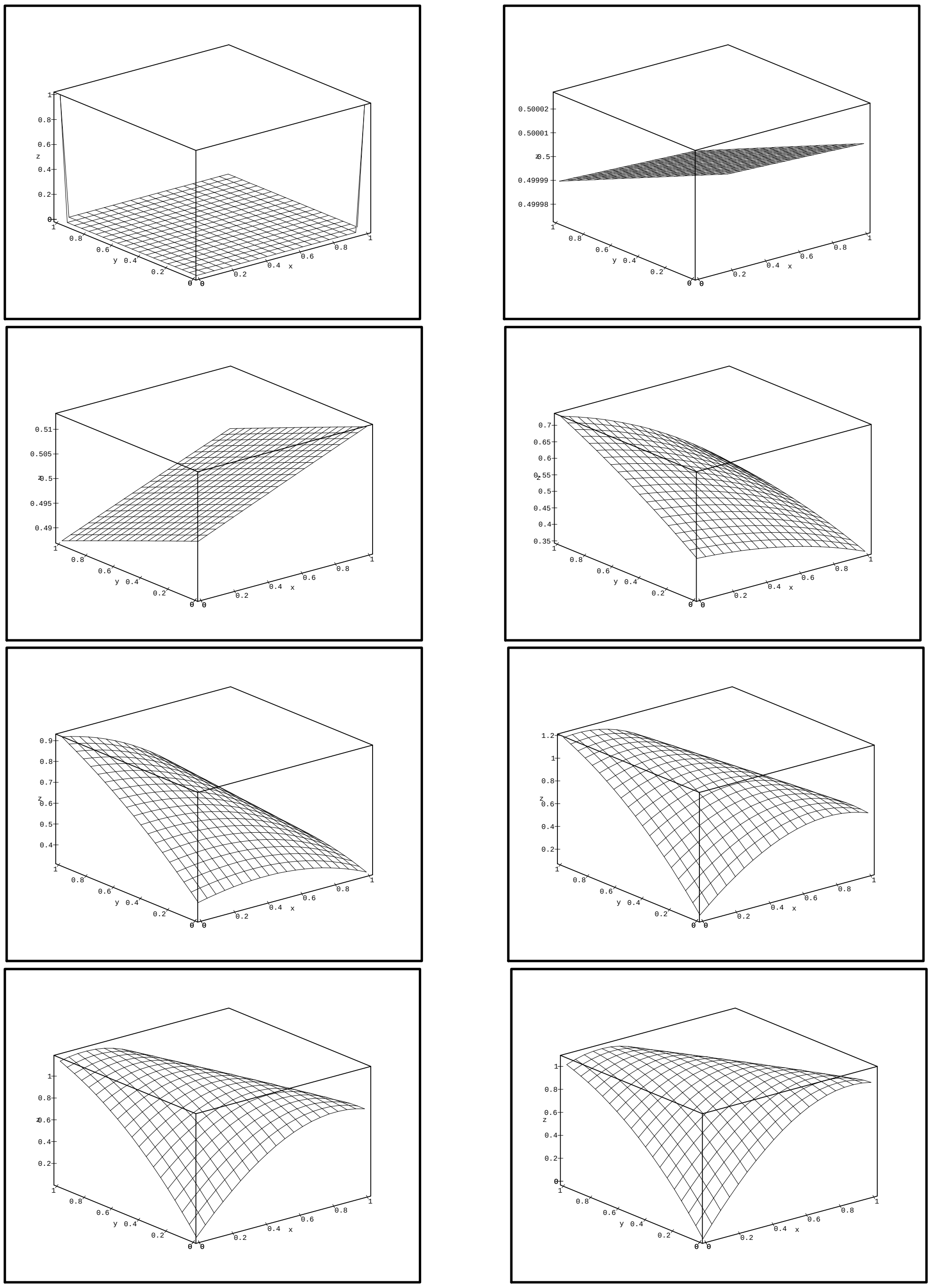

Figure 9: As above,but three filters. four samples; data: 83 iterations, total time 560 msecs on Sparc 10, multiplier for random parameter initialization 0.01

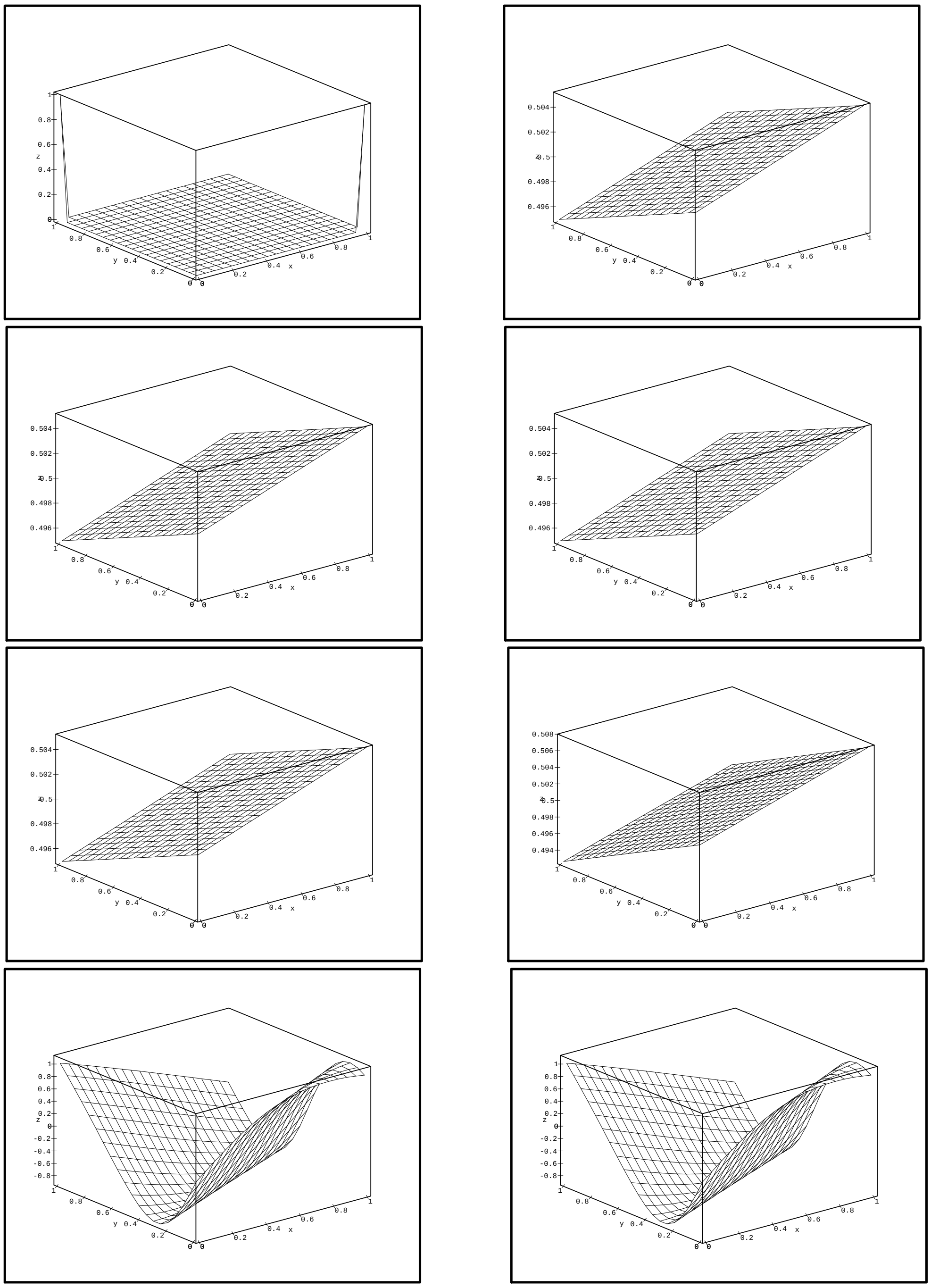

Figure 10: As above,but three filters. eight samples; Note that the majority of the time is required to get away from the initial spot in parameter space. Data: 17 iterations, total time 170 msecs on Sparc 10, multiplier for random parameter initialization 0.01

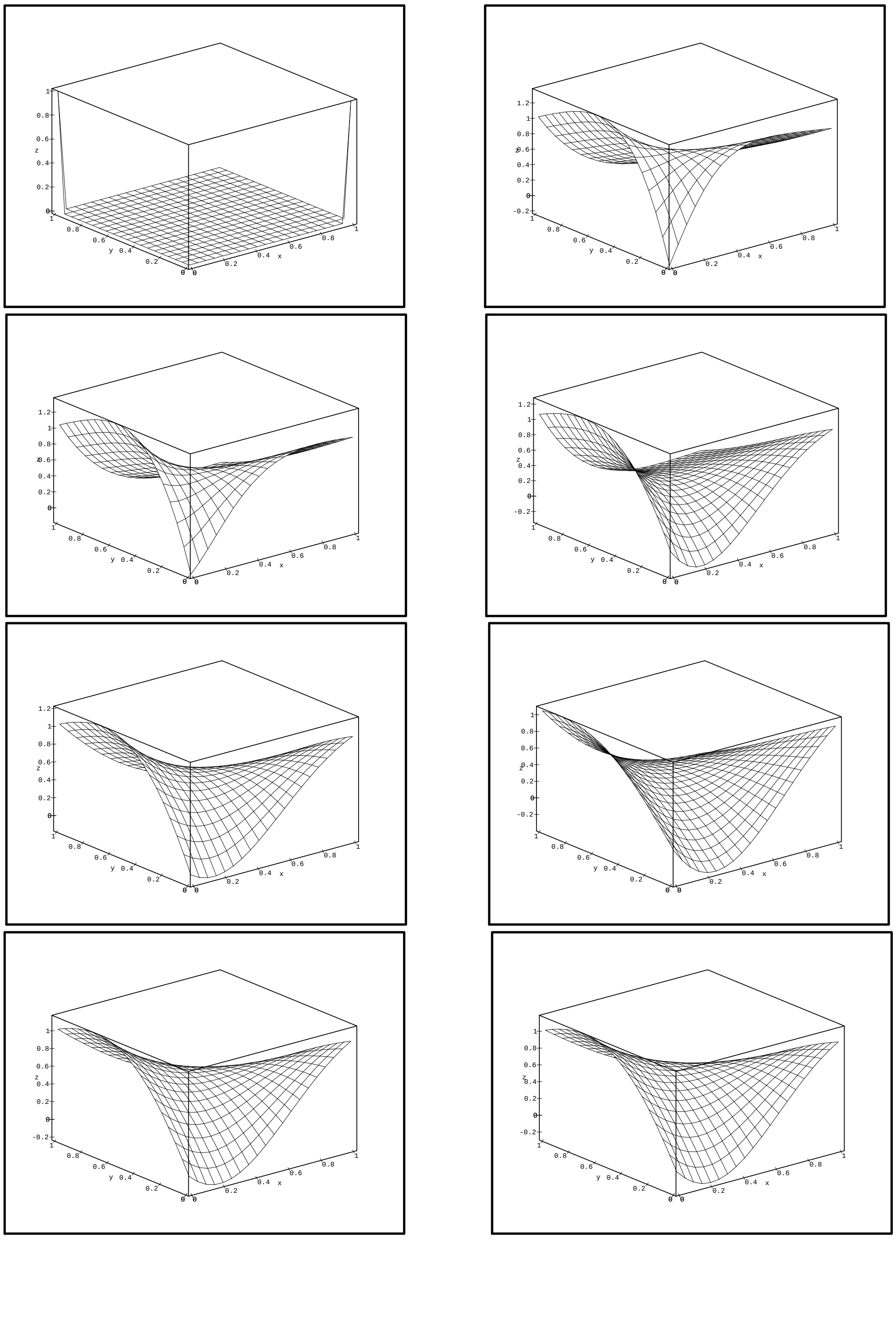

Figure 11: Four gaussian filters. eight samples; Note that in this application, each base filter has for each input dimension mean and standard deviation, thus not isotropic Gaussian. Data: 33 iterations, total time 450 msecs on Sparc 10, multiplier forrandom parameter initialization 2.0

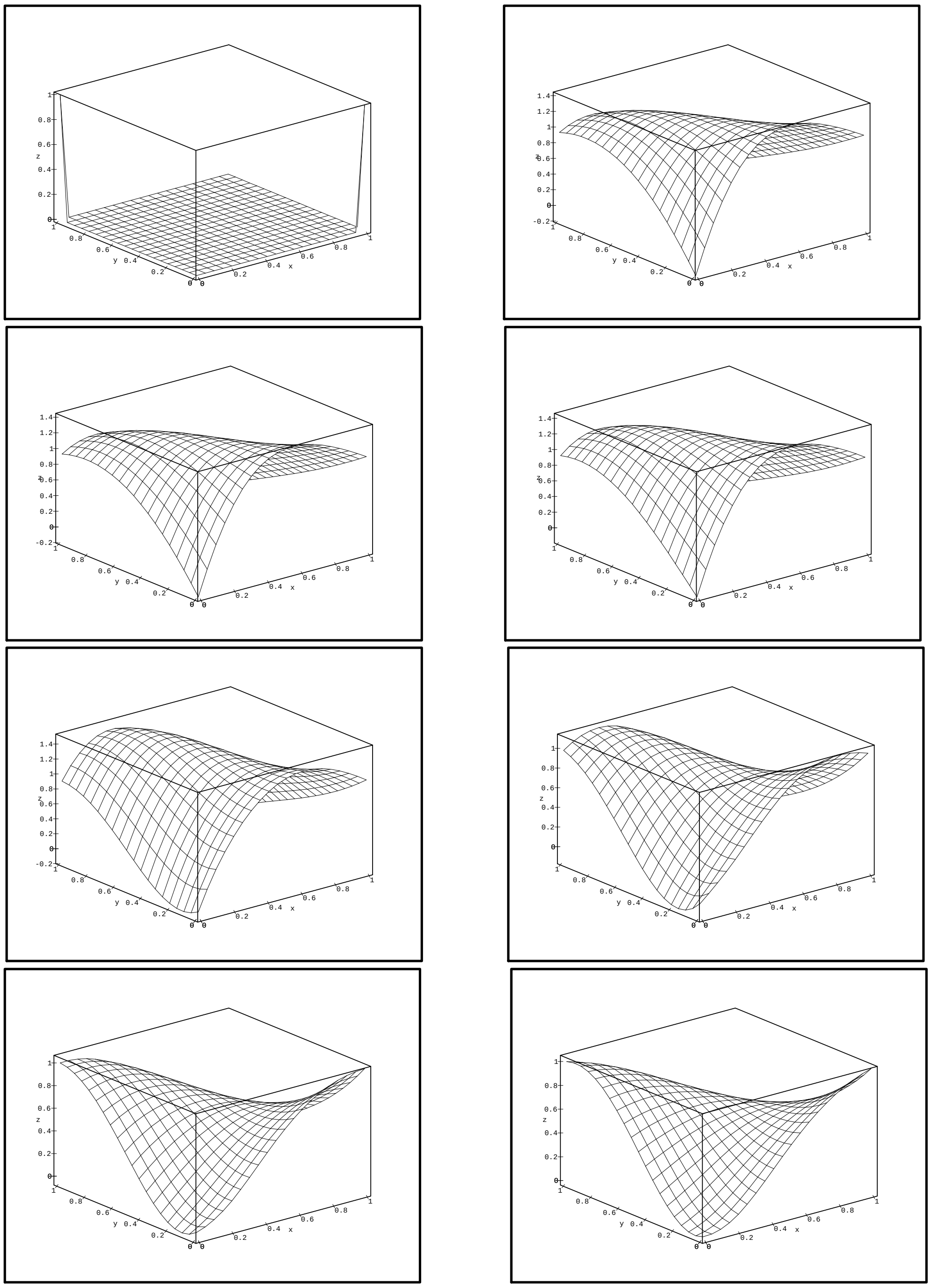

Figure 12: As above but five gaussian filters. eight samples; data: 12 iterations, total time 400 msecs on Sparc 10, multiplier for random parameter initialization 2.0

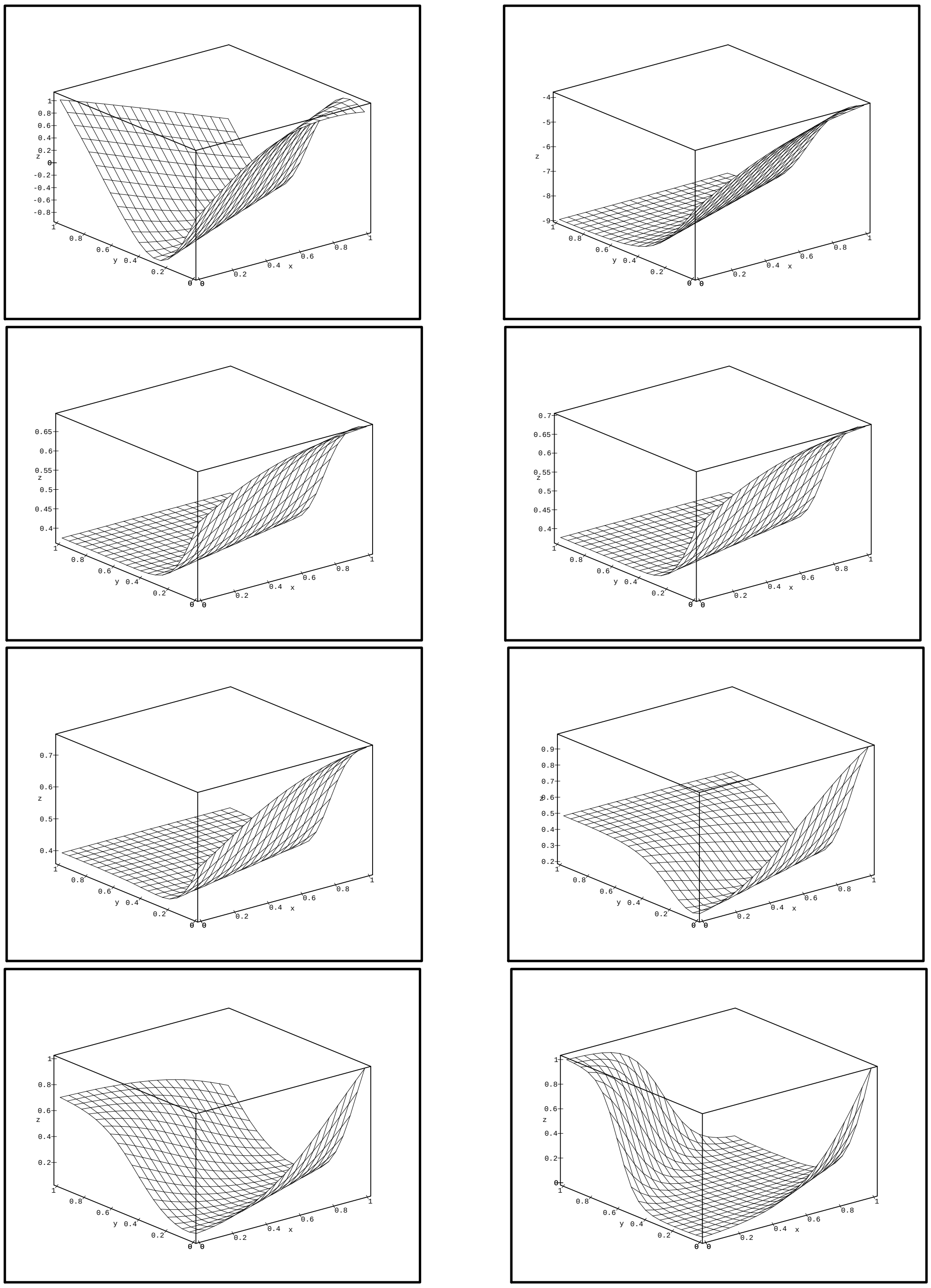

Figure 13: Evolution of the system after one filter/neuron dies; originally eight samples, three filters, shown on top left; System with one neuron dead, leaves two, shown on top right; immediate recomputation of Metric Tensor cannot correct here because of lack of overlap, as we can see in second row, left; Remaining images, from left to right and top to bottom, show the recovery; Data: 17 iterations, total time 160 msecs on a Sparc 10

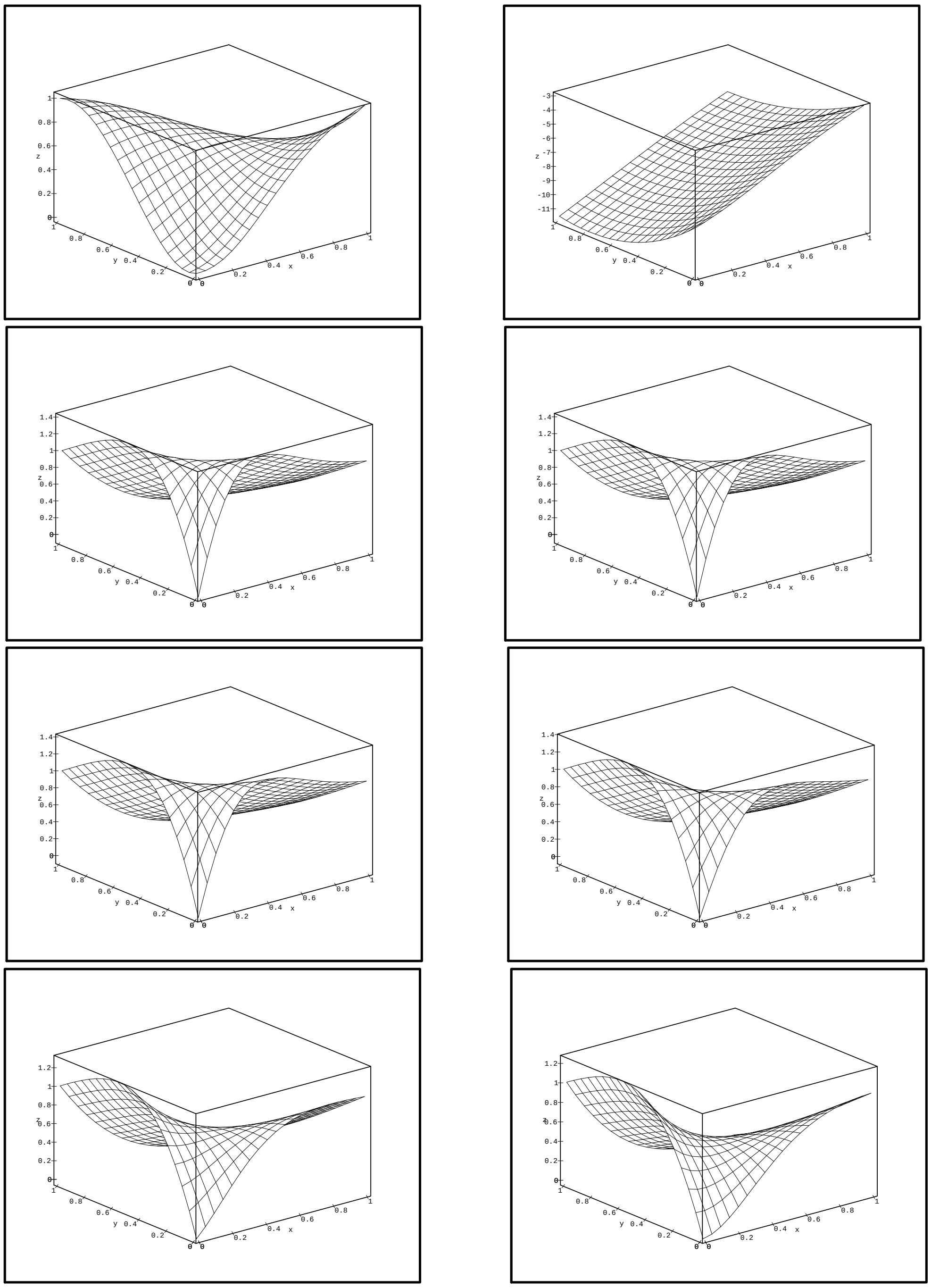

Figure 14: As above, but originally eight samples, five gaussian filters, shown on top left; System with one neuron dead, leaves four, shown on top right; immediate recomputation of Metric Tensor results in partial correction, shown second row, left; remaining images, from left to right and top to bottom, show the recovery; data: 12 iterations, total time 220 msecs on a Sparc 10

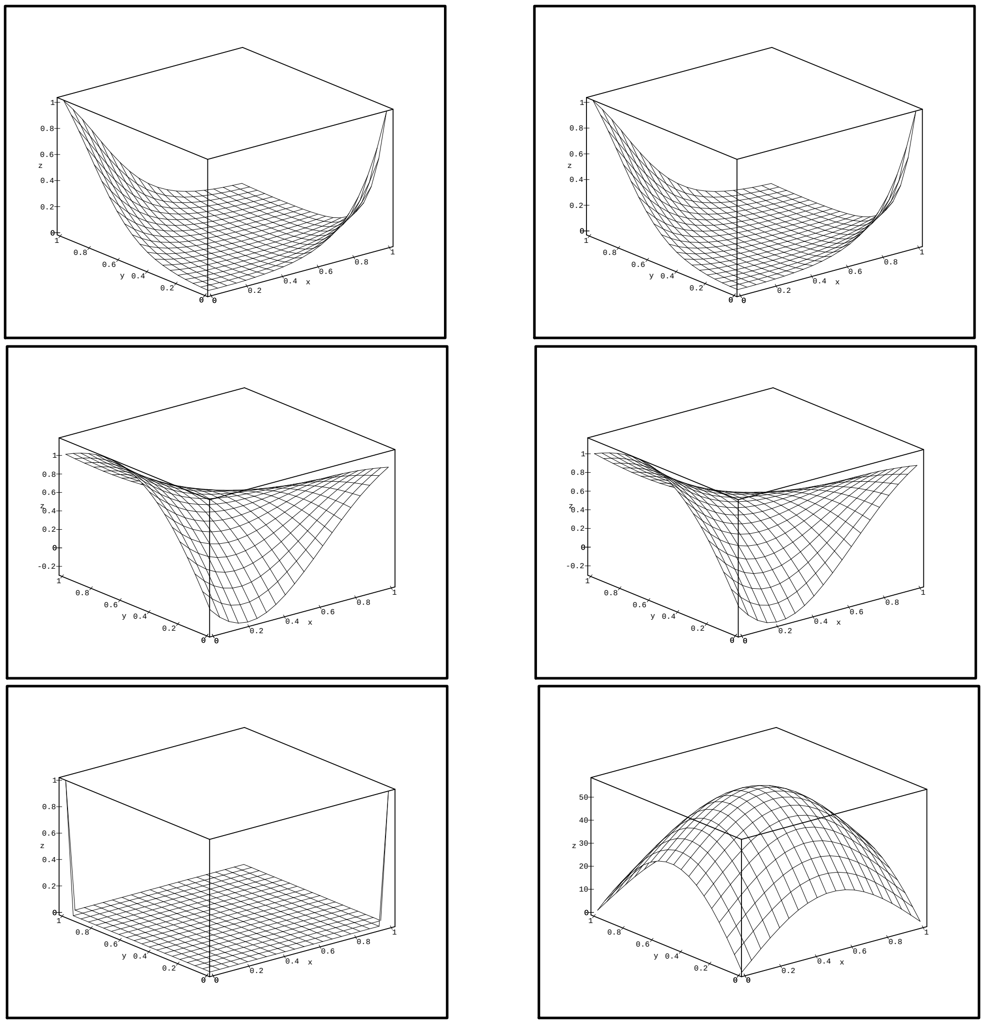

Figure 15: The top left image shows a converged system with 2 sigmoidal basis functions; top right shows the same system, once we have simply added another sigmoidal basis function, with random parameters; immediate recomputation of Metric Tensor results in instantaneous integration of new node/neuron; we could profit from the new node to decrease the error tolerated; if we choose not to do so, as shown here, integration is instantaneous. The row below shows the same with a system of originally four gaussian filters,shown on the left, to which we have added a fifth node, also gaussian, with randomly chose parameters; again, immediate recomputation of Metric Tensor results in instantaneous integration of the new node/neuron, shown in the same row, on the right. The third row shows on the left the set of four trainingsamples for XOR and on the right the instantaneous(i.e. one iteration) computation of the correct parameters with a system of four gaussian nodes: Four vectors, if linearly independent, span a four-dimensional manifold, thus in this case the whole sample space (Four samples => four dimensions). It is thus sufficient to compute the contravariant components to get the wanted approximation. As already mentioned, the contravariant components are simply the weights from the hidden layer to the output layer

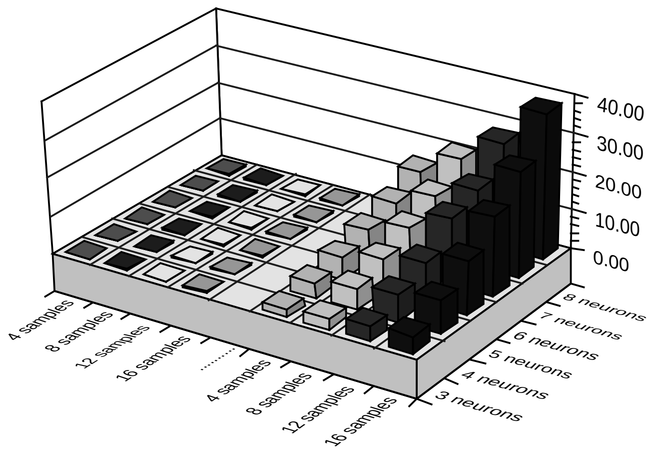

Figure 16: Time to convergence, averaged over 20 runs: Projection Learning on left, backpropagation on right: Convergence time in seconds, initializing every second weight with a mean of 0.3, the others with a mean of 0.0. Maximum amplitude of random initialization process is +/- 0.005. For text, see section "Some Results and Further Features of Dynamical Non-orthogonal Bases"

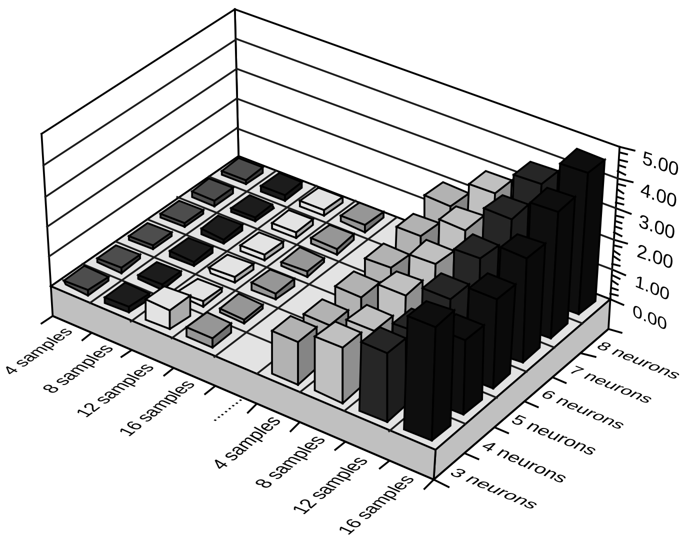

Figure 17: Time to convergence, averaged over 20 runs: Projection Learning on left, backpropagation on right: Convergence time in seconds, initializing with a mean of 0.0. Maximum amplitude of random initialization process is +/- 1.0. For text, see section "Some Results and Further Features of Dynamical Non-orthogonal Bases"

Unité de recherche INRIA Lorraine, Technôpole de Nancy-Brabois, Campus scientifique,
615 rue de Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur

INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)

ISSN 0249-6399