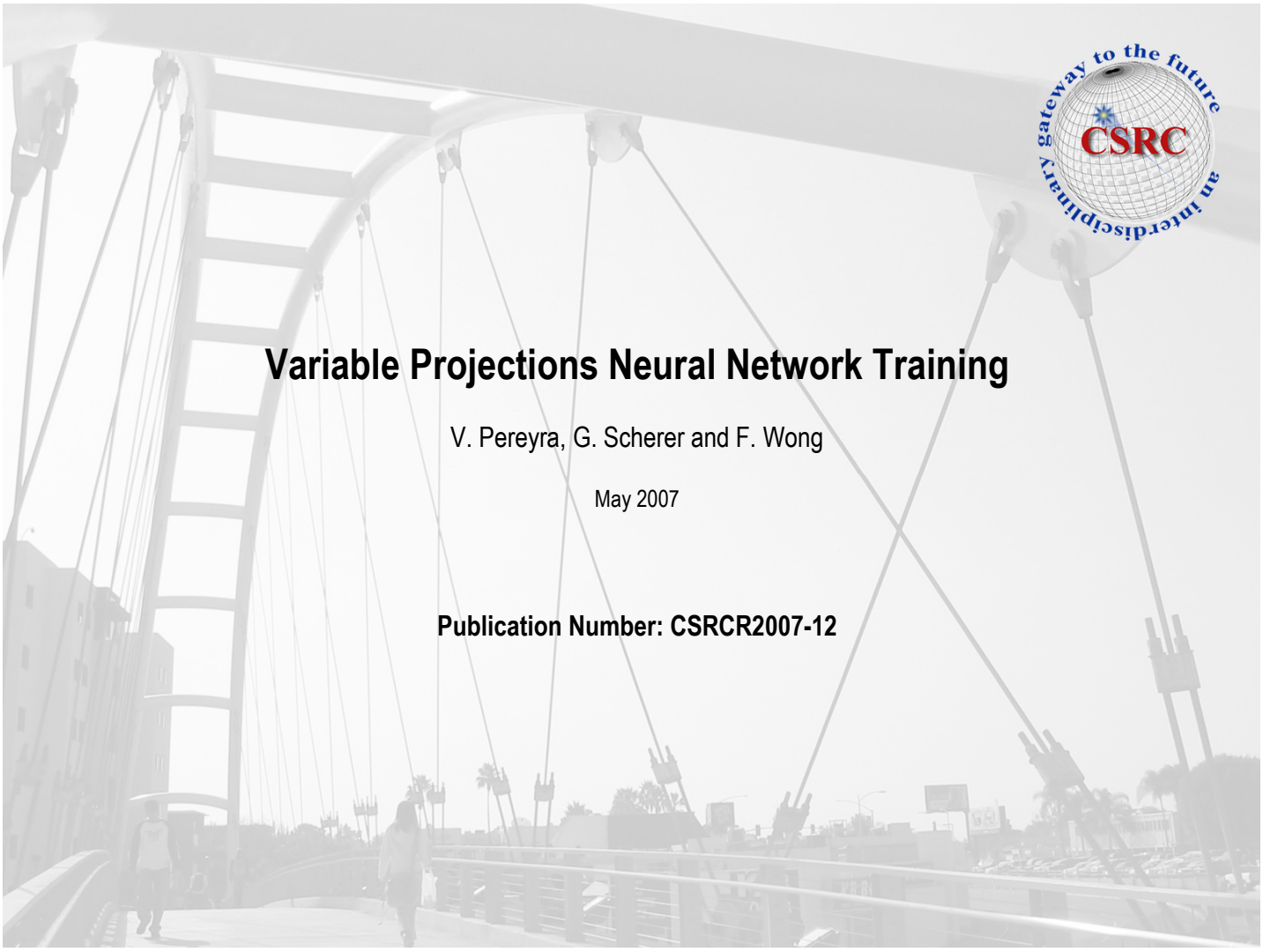

Variable Projections Neural Network Training

V. Pereyra, G. Scherer and F. Wong

May 2007

Publication Number: CSRCR2007-12

Computational Science & Engineering Faculty and Students Research Articles

Database Powered by the Computational Science Research Center Computing Group

COMPUTATIONAL SCIENCE & ENGINEERING

SAN DIEGO STATE UNIVERSITY

Computational Science Research Center
College of Sciences
5500 Campanile Drive
San Diego, CA 92182-1245
(619) 594-3430

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Mathematics and Computers in Simulation xxx (2006) xxx–xxx
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Variable projections neural network training

V. Pereyra[a,*], G. Scherer[b], F. Wong[a]

[a] *Weidlinger Associates Inc., Los [illegible], CA, United States*
[b] *University of Reading, Mathematics Department, United States*

Abstract

The training of some types of neural networks leads to separable non-linear least squares problems. These problems may be ill-conditioned and require special techniques. A robust algorithm based on the Variable Projections method of Golub and Pereyra is designed for a class of feed-forward neural networks and tested on benchmark examples and real data.

Keywords: Neural networks; Model; Variable projection algorithm

1. Introduction

Neural networks are non-linear parametric models. Training the network corresponds to fitting the parameters in these models in the least squares sense, by using pre-classified data (a training set) and an optimization algorithm. As the non-linear least squares problem (NLLSQ) that results is one whose linear and non-linear variables separate, the use of the Variable Projection (VP) algorithm is applicable and it will increase both the speed and robustness of the training process [21,25,13,14,33,34,36,35].

In this work we design and test a training algorithm based on Variable Projections. The algorithm is applicable to one-hidden layer, fully connected, neural network models using several types of activation functions. A generalization of VP developed and implemented by Golub and Leveque [11] (see also [8,10,18]) can be used for the case of multiple outputs. A version with separable equality constraints has been presented in [19].

For a small number of parameters the [illegible]ble Projection algorithm can be applied directly to the problem as a whole. When the number of parameters increases (for instance in VLSI design), the NLLSQ problem can be broken up and solved using block techniques combined with separability [28]. This will be the subject of a forthcoming paper.

In the next section the necessary neural network concepts and notation are given and the specific least squares problem for training the network is formulated. Section 3 is an overview of the Variable Projection method, including a description of the implementation used (VARPRO). It also contains comments on the literature concerning the numerical condition of the problem. The tests results for the algorithm are presented in Section 4.

Why Variable Projections? Besides the obvious advantage of reducing the number of parameters to be determined by eliminating the linear parameters, it has been shown [30,33,22,24] that the resulting reduced problem is better conditioned than the original full one, and if the same optimization algorithm is used it always converges in less

* Corresponding author.
E-mail addresses: pereyra@ca.wai.com (V. Pereyra), godela@compuserve.com (G. Scherer).

0378-4754/$32.00 © 2006 Published by Elsevier B.V. on behalf of IMACS.
doi:10.1016/j.matcom.2006.06.017

iterations. Since with a careful implementation, the cost per iteration is about the same, these results and extensive applications as reported in [14] (where a wealth of references can be found) indicate that there is a net gain in using VP over solving the unreduced problem with a conventional non-linear least squares algorithm. Since usually neural networks are trained by using very slow (hundred of thousands of iterations) algorithms, such as back propagation (gradient method with step control), then the gain as compared to conventional training methods is even larger.

On the negative side (for every algorithm), if there are no good *a priori* initial values for the non-linear parameters, then these non-linear, non-convex problems will frequently have multiple solutions and therefore some kind of global optimization technique needs to be used to escape from undesired local minimae. We show in the application section that a simple Monte Carlo method in which multiple initial values are chosen at random helps with this problem.

Valid concerns of NN practitioners are the design of the network and the so called bias-variance dilemma. Since we are considering single hidden layer perceptrons, the only design parameter in this case is the number of nodes in the hidden layer, which should be as small as possible, since parsimonius models are to be desired. The second concern is related to the fact that training data in real situations will have errors and that the resulting problems are often ill-conditioned and therefore there is a real problem in over-fitting the training data. Minimizing the bias (i.e., the sum of squares of residuals) and the variance are desirable but conflicting goals. As in any bi-objective optimization problem a compromise has to be reached. The procedure described below uses a modern implementation of the Levenberg–Marquardt method (cf. [20,5]) which has a built in approach to paliate the above problem if an approximation to the variance is available.

A very thorough discussion of ill-conditioned least squares problems can be found in [15], where many regularization techniques are considered. Observe that when used appropriately, the Levenberg–Marquardt method is akin to Tihonov regularization.

2. Neural networks

Neural networks [2] are a convenient way to represent some non-linear mappings between multidimensional spaces given in terms of superposition of non-linear functions, by using so called *activation* functions and *hidden units*. We discuss in this paper the training of one-hidden layer, feed-forward, fully connected neural networks (NN), although the techniques are applicable to more general networks, for instance, those with feedback loops. These NN consist of three types of *nodes* arranged in *layers*: *input, hidden* and *output* layers. Each node can have several inputs and outputs.

Each node acts on the input information in several possible ways, depending on the layer type:

- *Input node:* No action.
- *Hidden node:* Two possible, sequential actions,
  - Weighted sum of its inputs with a possible offset (bias) that can be incorporated as an additional input. If w_i are the weights, and x_i the inputs, $i = 0, \ldots, d$, with x_0 corresponding to the bias, i.e., $x_0 = 1$,

$$\sum_{i=0}^{d} w_i x_i \equiv \mathbf{w}^{\mathrm{T}}\mathbf{x}.$$

  - This linear output can be generalized by applying a non-linear function $\phi(.)$, called an *activation* function, so that the final output is: $\phi(\mathbf{w}^{\mathrm{T}}\mathbf{x})$. The non-linear activation function will in general depend also on some additional parameter vector β.
- *Output node:* Can weight and sum inputs only, or additionally also apply an activation function.

In a feed-forward NN there is information flowing in only one direction. A fully connected NN has every node in one layer connected to every node in the next layer and there are no connections between nodes of the same layer.

To allow for a general mapping one must consider successive transformations, corresponding to several layers of adaptive parameters, but as will be seen below, one-hidden layer is enough for the universal approximation property to hold.

2.1. Perceptrons

These are networks of threshold or sigmoid activation functions, usually applied to classification problems. The most common choice for the activation function is the logistic sigmoid function:

$$\phi(a) = \frac{1}{1 + e^{-a}}. \tag{1}$$

Important properties of these functions are that they are continuously differentiable, they are able to approximate the hard delimiter step function, and they have a simple derivative form: $\phi\prime(a) = \phi(a)(1 - \phi(a))$.

Since the sigmoid functions are dense in the space of continuous functions, a single hidden layer NN can approximate any continuous function at any precision, if enough elements are used (see [6]).

Sometimes another activation function is used:

$$\phi(a) = \tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}.$$

A neural network whose hidden units use the tanh function is equivalent to one using sigmoids but having different weights and biases. Empirically, it is often found that tanh functions give rise to faster convergence of training algorithms than sigmoid functions.

2.2. Radial basis functions networks

This is a neural network model in which the activation of a given hidden unit is determined by the distance between the input vector and a *prototype* vector. Radial basis functions (RBF) were first used as a technique for multidimensional interpolation (see [4]). They employ the following functions:

$$\phi(\|\mathbf{x} - \mu\|_2^2), \tag{2}$$

where μ is a center vector and $\|.\|_2$ is the l_2 norm. We will work with the spherical Gaussian function defined by:

$$\phi(\mathbf{x}) = \exp\left(\frac{-\|\mathbf{x} - \mu\|_2^2}{2\sigma^2}\right), \tag{3}$$

where σ_j^2 is the variance if this function is thought as a normal density function.

The radial functions NN mapping that we will consider is:

$$\sum_{j=1}^{m} w_j \phi_j(x) + w_0. \tag{4}$$

A common approach used to determine the parameters or train a RBF network is the so called *unsupervised* training, in which the linear parameters w_j and the non-linear ones μ_j and σ_j are computed in two different steps. In a first stage, the non-linear parameters μ_j and σ_j are calculated using only input data information, specifically the density distribution of the input vectors. The resulting problem in the parameters w_j is now merely a linear least squares problem that can be solved by any standard method. Notice that this parameter separation is not the same as in Variable Projections.

The choices used in unsupervised training mean that the basis function parameters ignore the information in the target set. This can have the disadvantage that the optimal choice in terms of density estimation of input need not be the optimal choice for the representation of the target or output data.

The joint determination of the whole set of parameters, both linear and non-linear is known as *supervised* training. This is the natural setup for the use of Variable Projection since the resulting problem is separable. Observe that in the original 1973 paper, Golub and Pereyra used combinations of Gaussians as one of the test cases.

Concerning the representational properties of the radial function NN, the universal approximation property is valid for networks with Gaussian basis functions using adjustable width parameters and a single hidden layer, just as in the sigmoid perceptron case.

2.3. Summary of activation functions

The activation functions described above are: the sigmoid activation function which for the multivariate case is: $\phi(\mathbf{w};\mathbf{x}) = \frac{1}{1+\exp\{\mathbf{w}^{\mathrm{T}}\mathbf{x}\}}$, and the radial basis or Gaussian spherical function: $\phi(\mu,\sigma;\mathbf{x}) = \exp(\frac{-\|\mathbf{x}-\mu\|_2^2}{2\sigma^2})$.

To complete the catalogue of activation functions we consider exponential functions $\phi(\omega;\mathbf{x}) = \exp^{\{\omega^{\mathrm{T}}\mathbf{x}\}}$, which are important to model phenomena characterized by systems of linear ordinary differential equations with constant coefficients, or in the complex case, to model spectra and also plane wave propagation of multiple signals. The real part of complex exponentials $\phi(\omega;\mathbf{x}) = a\ \exp^{\{\omega^{\mathrm{T}}\mathbf{x}\}}(\cos\mathbf{d}^{\mathrm{T}}\mathbf{x} - b/a\ \sin\mathbf{d}^{\mathrm{T}}\mathbf{x})$ is useful to approximate real oscillatory data, and we include also these functions in our tool box.

Also, in order to take into account the anisotropic case, we generalize the radial spherical functions to the ellipsoidal case: $\phi(\mu,\sigma;\mathbf{x}) = \mathrm{e}^{-\sum_{i=1}^{d}((x_i-\mu_i)/\sigma_i)^2}$.

2.4. Separable non-linear least squares formulation

In what follows we will consider a simplified three-layer NN: input, hidden layer with n nodes, and output layer with only one node to which no activation function is applied. The weight parameters involved are the weights at each hidden node j : $\mathbf{w}_{hj}$, and the weights for the output node: $\mathbf{w}_{\mathrm{o}}$. In this case the final approximating function reduces to a weighted combination of any of the aforementioned activation functions:

$$\sum_{j=1}^{n} \mathbf{w}_{\mathrm{o}j}\phi_j(\beta_j;\mathbf{w}_{hj}^{\mathrm{T}}\mathbf{x}_i).$$

The training of a neural network consists of considering known input–output pairs (the training data), in order to find the NN parameters that best reproduce this data. A frequently used measure of performance is the sum of squares of the residuals between the NN output and that of the training set, defined by $(\mathbf{x}_i, t_i)$, $i = 1,\ldots,m$. Assuming that the outputs are independent,[1] the parameters involved: weight parameters $\mathbf{w}_o$ and $\mathbf{w}_{hj}$ and activation function additional parameters β_j are to be determined so that the sum of squares error function (SSE) is minimized

$$\mathrm{SSE} = \min_{\mathbf{w}_{\mathrm{o}},\beta,\mathbf{w}_h} \frac{1}{2}\sum_{i=1}^{m}\left(\sum_{j=1}^{n}\mathbf{w}_{\mathrm{o}j}\phi_j(\beta_j;\mathbf{w}_{hj}^{\mathrm{T}}\mathbf{x}_i) - t_i\right)^2.$$

3. Numerical solution of separable non-linear least squares problems

Recasting somewhat the notation so that the connection with the formulation in the Golub–Pereyra paper [13] is clearer, the above problem reduces to the use of the approximating function:

$$\eta(\mathbf{a},\alpha;\mathbf{x}) = \sum_{j=0}^{n} a_j\phi_j(\alpha_j;\mathbf{x}),$$

and the resulting separable non-linear least squares problem is:

$$\min_{\mathbf{a},\alpha} r(\mathbf{a},\alpha;\mathbf{x}) = \min_{\mathbf{a},\alpha}\|\mathbf{t}-\eta(\mathbf{a},\alpha;\mathbf{x})\|_2^2 = \min_{\mathbf{a},\alpha}\sum_{i=1}^{m}(t_i-\eta(\mathbf{a},\alpha;\mathbf{x}_i))^2 = \min_{\mathbf{a},\alpha}\|(\mathbf{t}-\mathbf{\Phi}(\alpha)\mathbf{a})\|_2^2. \quad (5)$$

Observe that in the formula of SSE above we have replaced w_{oj} by a_j and both, the $\mathbf{w}_{hj}$ and the β_j by α_j.

To summarize, the size of the training set is m, the number of hidden nodes is n, the output-node weights are the (linear parameters) a_j, while the α_j are the non-linear parameters, among others, the weights inside the activation function

[1] In the case when there is a correlation between the outputs or the errors have different sizes [24] suggests a more appropiate formulation involving the output error correlation matrix, giving rise to a Generalized Least Squares problem [3]. See also [2] Chapter 6.

associated with the hidden node j. The bias is introduced at the first layer through $x_0 = 1$, and in the hidden layer through $\phi_0 = 1$. We assume that the vectors α_j have k elements, (i.e., there are $k-1$ inputs) and that $m \geq n \times k$. In addition, we call $\boldsymbol{\Phi}(\alpha) = [\phi_0(\alpha),\ \phi_1(\alpha), \ldots,\ \phi_n(\alpha)]$ the $(m \times n)$ matrix function whose elements are $\boldsymbol{\Phi}_{i,j}(\alpha) = \phi_j(\alpha_j; x_i)$.

This type of non-linear least squares problem is frequently ill-conditioned.

3.1. Variable Projection algorithm

The *Variable Projection algorithm* [13] takes advantage of the separability of the parameters $\boldsymbol{\alpha}$ and $\mathbf{a}$ to reduce the original minimization problem to a smaller (albeit possibly more complex) non-linear one, involving only the parameters $\boldsymbol{\alpha}$, and a linear one in the parameters $\mathbf{a}$, which is solved in a post-processing step, after the non-linear parameters have been calculated.

Basically, in a first step the linear parameters are eliminated using the explicit solution to the linear problem in terms of the pseudo-inverse of $\Phi(\alpha)$ (for any fixed α). Then the reduced non-linear problem is solved and finally the linear parameters are obtained. In more detail, the steps are:

- For any fixed α, the problem $\min_{\mathbf{a},\alpha} \|\mathbf{t} - \boldsymbol{\Phi}(\alpha)\mathbf{a}\|_2^2$ is linear with minimal solution: $\mathbf{a} = \boldsymbol{\Phi}^+(\boldsymbol{\alpha})\mathbf{t}$, where $\Phi^+(\alpha)$ is the Moore–Penrose generalized inverse of the rectangular matrix $\Phi(\alpha)$ [12].
- Replacing this value of $\mathbf{a}$ into the original problem we obtain the reduced non-linear functional:

$$r_2(\alpha) = \|\mathbf{t} - \boldsymbol{\Phi}(\alpha)\boldsymbol{\Phi}^+(\alpha)\mathbf{t}\|_2^2 = \|\mathbf{P}^{\perp}_{\boldsymbol{\Phi}(\alpha)}\mathbf{t}\|_2^2. \tag{6}$$

For each fixed α, the linear operator:

$$\mathbf{P}^{\perp}_{\boldsymbol{\Phi}(\alpha)} = (\mathbf{I} - \boldsymbol{\Phi}(\alpha)\boldsymbol{\Phi}^+(\alpha)),$$

is the projector on the orthogonal complement of the column space of $\boldsymbol{\Phi}(\alpha)$. Thus the name Variable Projection given to this reduction procedure, since the solution of the above non-linear least squares problem requires an iterative process in which α will change from an initial guess α^0 to a converged minimizer.

In Fig. 1 we show a cartoon of the VP in action. We depict $\boldsymbol{\Phi}(\alpha)$ (for fixed α) as a linear mapping from $\mathcal{R}^n$ into $\mathcal{R}^m$. That is, the range of $\boldsymbol{\Phi}(\alpha)$ is a linear sub-space of dimension n (or less, if the matrix is rank deficient). When α varies during the VP iteration this subspace pivots around the origin. For each α the residual is equal to the L_2 distance from the data vector t to the corresponding subspace. Thus we see in this cartoon that using a linear perceptron (regression with fixed α^0, i.e., non-adaptive sigmoids), cannot in general be as good as the final converged non-linear adaptive perceptron.

- Once a minimizer $\hat{\alpha}$ for the above problem is obtained, it is substituted into the linear one, which is solved for $\hat{\mathbf{a}}$ and the final minimizer for the original problem $(\hat{\mathbf{a}}, \hat{\alpha})$ is obtained.

There are several important points proved in [13,33] and [30] respectively, that make the algorithm not only viable, but also more efficient and specially much better conditioned:

- The stationary point set of the original and the reduced functionals coincide and the algorithm is valid also in the rank deficient case.
- The reduced non-linear functional, although it looks more complex and therefore costly to evaluate, gives raise to a better conditioned problem, which always takes less iterations to converge than the full problem [30]. By careful implementation of the linear algebra involved and by use of a simplification due to Kaufman [17], it turns out that the cost per iteration for the reduced functional is similar to that for the full functional, with little or no increase in the number of iterations.
- The elimination reduces the dimensionality of the problem and as a consequence fewer initial parameters need to be guessed.

3.2. VARPRO program

One of the Variable Projection implementations we use is based on the original program written by Pereyra [13] as modified by J. Bolstad (1977) [16]. The minimization method used is a modification of Osborne of the Levenberg–

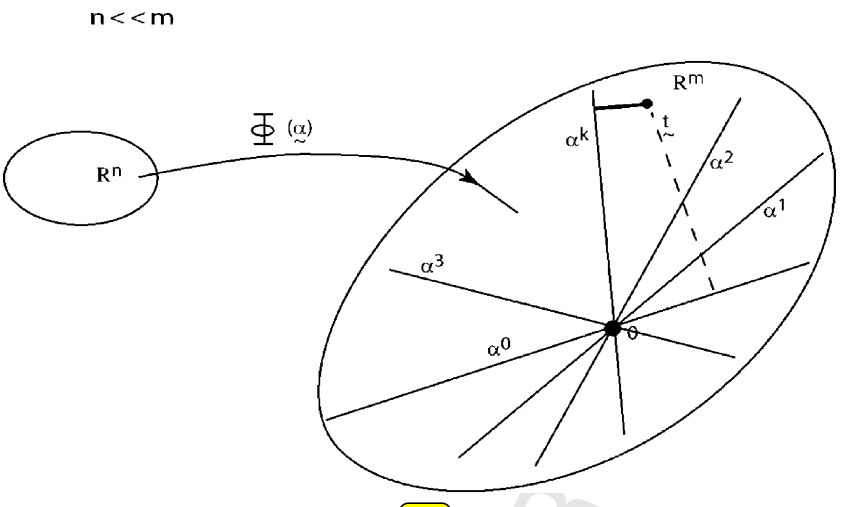

Fig. 1. Variable tion in action.

Marquardt (L–M) algorithm, and it also includes the Kaufman simplification.[2] Careful implementation of the linear algebra involved in the differentiation of the Variable Projector and the L–M algorithm produces an efficient algorithm.

The information provided by the program allows for a statistical analysis, including for example, uncertainty bounds in the parameter estimations (for a brief overview see [31]).

3.3. Numerical condition and regularization

Analysis and experimental results in [32] for fully connected NN using sigmoid activation functions (which have limited discrimination capabilities) suggest that many network training problems are ill-conditioned. One can show that columns of the Jacobian can easily be nearly linearly dependent. As proven in [33], the use of the Variable Projection method improves the condition of the problem.

The interesting analysis in [7] might be useful to design an even more robust algorithm, which would not only be applicable to the original NN problem but to non-linear separable problems in general. In fact the authors show theoretically and practically that regularizing the NN non-linear problem directly (using the Tikhonov approach to compensate for ill-conditioning), and then linearizing using Gauss–Newton, gives a problem with smaller condition number than the usual approach for a non-linear problem, namely to linearize and then to regularize (Levenberg–Marquardt).

4. Numerical tests

We start first with a set of synthetic tests to make sure that the implementation is correct and also to gain insight on the particularities of the different activation functions and implementations. For each class of activation function and program we run the same tests. Four basis functions are considered (i.e., four nodes in the hidden layer of the network) and a set of 11×11 uniformly distributed input values in $[-1, 1] \times [-1, 1]$ are taken.

Then, non-linear parameters are selected at random, while the linear parameters are set to the value j, $j = 1, \ldots, 4$. These parameters are considered the target values and they are used to generate output values (corresponding to the 121 input values) for training. Two kind of tests are performed.

Basically, after "forgetting" the target values, 100 sets of initial values are chosen for the non-linear parameters and VARPRO is run. For the first nine sets we select initial values which are increasingly farther from the target values, by

[2] Where a term in the Frechet derivative is neglected.

Table 1
Program NSF (Gay-Kaufman)

Basis	Last of 9	Iters.	Res.	Iters. best	Res. Best	Total time (s)
Exp.	9	111	1.9 (-14)	61	7.6 (-15)	23.58
Gauss	6	74	7.8 (-15)	94	6.1 (-15)	36.08
Ellipt.	2	150	5.8 (-13)	–	–	100.35
Sigmoid	0	–	–	126	2.6 (-12)	56.71

Table 2
Program NSG (Gay-Kaufman)

Basis	Last of 9	Iters.	Res.	Iters. best	Res. Best	Total time (s)
Exp.	9	75	8.9 (-15)	67	4.7 (-15)	10.8
Gauss	6	51	5.7 (-13)	165	5.8 (-15)	17.43
Ellipt.	2	136	3.8 (-15)	121	5.8 (-06)	27.05
Sigmoid	1	182	1.1 (-11)	–	–	22.48

multiplying those target values by $(1 - irun \times 0.1)$, $irun = 1, \ldots, 9$. We stop at 9 since for run 10 all the non-linear initial parameters would be set to zero, which would create problems with some of the activation functions.

For the next 91 runs we select the initial values at random, since that will be how the actual Montecarlo global optimization algorithm that we use later will operate. For each run the termination values are recorded.

Thus we test both how far one can move from the exact solution and still get convergence, and also the characteristics of a simple global optimization Monte Carlo approach combined with a local optimization iteration.

4.1. Codes tested

We test three different implementations of Variable Projections: the original VARPRO due to Pereyra and modified by Bolstad, and two codes from the PORT library implemented by Gay and Kaufman: NSF and NSG [9]. The difference between the two PORT codes is that NSF does not require derivatives since it approximates them by finite differences, while NSG uses derivatives, just as VARPRO. All the codes are implementations of the Marquardt algorithm for non-linear least squares, taking into account separability and using the Kaufman simplification. The PORT codes use more modern technology, including a trust method for step control. NSF also uses graph coloring technology to save on function evaluations when approximating the sparse Jacobian [1,5].

The codes report the total number of iterations to reach convergence (which we have chosen to declare when the residual norm is below 10^{-16}), or an error condition if there is abnormal termination. The most common errors are faillure for ill-conditioning or excess of iterations (the maximum number of iterations is set to 200 unless noted).

In Tables 1–3 we show a summary of the results. For all types of basis functions ill-conditioning is a problem. Also, starting without good *a priori* information leads to a slow search for the convergence basin and in many cases to divergence for excess of iterations.

These tests were performed on a dual processor Pentium III 800 MHz computer, under the LINUX operating system, with the programs implemented in FORTRAN 90 using the INTEL Fortran compiler ifc.

There are several considerations regarding pre-processing of the training set and initialization of the parameters that are specific to neural network problems. The most common form of pre-processing consists of a linear rescaling of the input data so that they have similar values. We have implemented this scaling following the recommendations

Table 3
Program VARPRO (Pereyra–Bolstad)

Basis	Last of 9	Iters.	Res.	Iters. best	Res. Best	Total time (s)
Exp.	2	72	2.5 (-16)	72	1.8 (-16)	19.6
Gauss	4	106	5.3 (-16)	29	2.3 (-16)	5.37
Ellipt.	2	227	3.1 (-16)	300	7.0 (-06)	11.19
Sigmoid	2	198	2.9 (-16)	200	9.5 (-09)	35.41

in [2], by re-scaling the input variables independently, using the mean and variance for each variable. So for each input variable x_i, $i = 1, \ldots, d$, the mean $\overline{x_i}$ and variance σ_i^2 are determined and the new variables are defined as: $\widetilde{x_i}^n = \frac{x_i^n - \overline{x_i}}{\sigma_i}$. This set has now mean zero and variance one. In the case of radial function networks, it is particularly important to normalize the input data so that they span similar ranges, because the activation of the basis functions are triggered by the distances to the function centers, measured in the l_2 norm.

Some of the headings in the tables are self explanatory, but we spell them out for clarity.

- *Basis:* Type of activation function;
- *Last of 9:* Last run converged of the first 9; it measures how far away from the solution we can start.
- *Iters.:* number of iterations that run took to converge;
- *Iter. Best:* Number of iterations for the best converged run (i.e., that run with the minimum residual that converges to the target or some reasonable permutation of it) for the 91 random initial values runs;
- *Res.:* Residual for that run;
- *Total time:* For the 100 runs, in seconds.

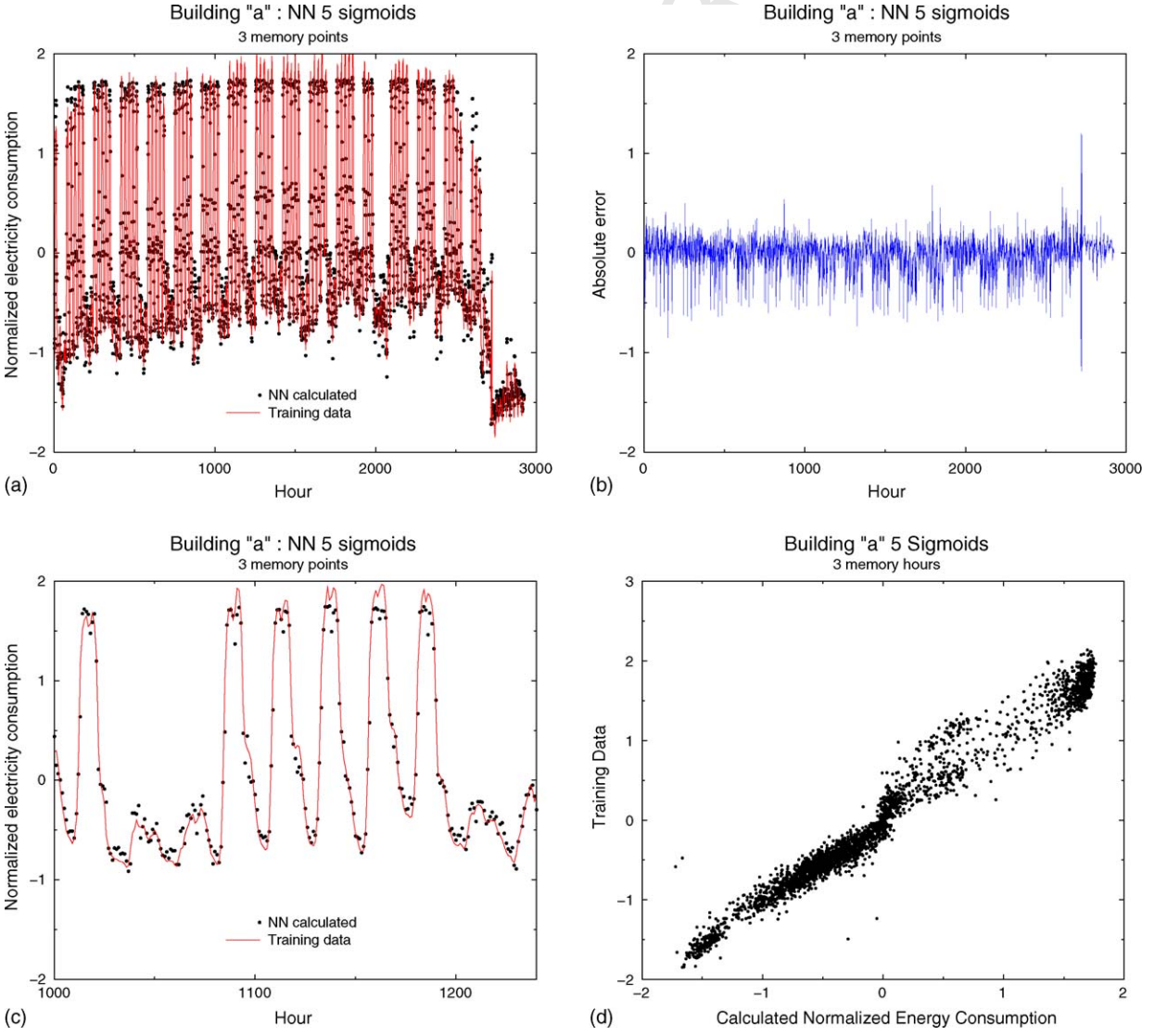

Fig. 2. Results for building 'a' perceptron training with five sigmoids.

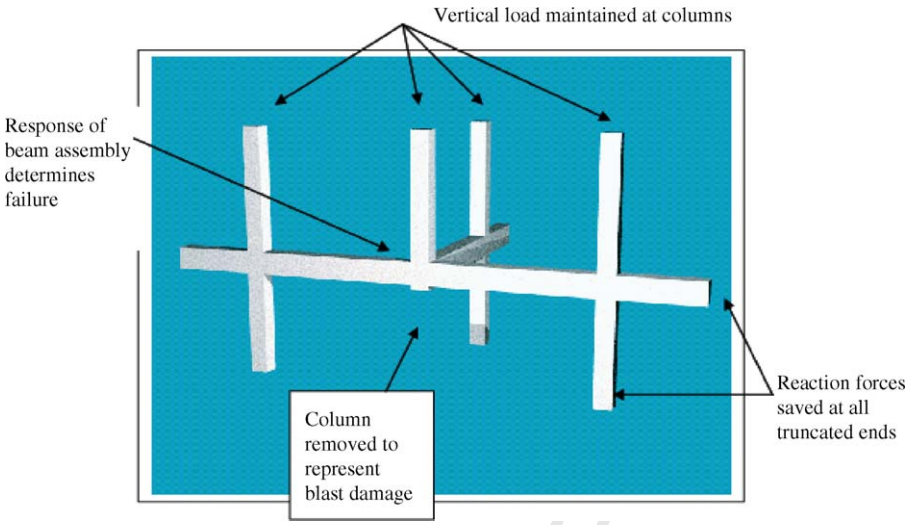

Fig. 3. A typical frame unit of the building.

4.2. Initialization of the parameters for radial basis networks

In the case of unsupervised training, prior to the solution of the (linear) least squares problem itself, the non-linear center and width parameters are determined from the input data set.

The choice of the centers μ_j usually will depend on the density distribution of the input vectors from the training set: they are in a way *prototype* input vectors. The *width* parameters σ_j may vary for each basis function. It is a good idea to have some overlap of the basis functions. One possibility is to use a multiple of the average distance between the centers. Another possibility would be to determine the average distance of each center to its nearest neighbours.

The strategy we used is the following. First, the number of basis functions and the centers were determined from a clustering of the input data. The width parameters were then chosen all equal for simplicity, defined as the average value of the distances between the centers.

In the supervised case, these same non-linear parameters can be used as initial values for the non-linear least squares approximation.

4.3. Benchmark tests

We consider a test problem from a public benchmark, about the prediction of hourly electrical energy consumption in a building, based on the date, time of day, outside temperature, outside air humidity, solar radiation, and wind speed.

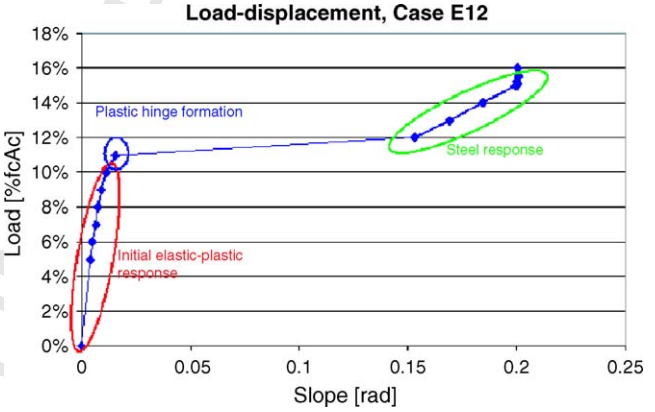

Fig. 4. Load-response characteristics.

Table 4
Dataset used in example

x_1	x_2	x_3	y
90	20	10	−0.71
308	20	80	−0.82
824	20	150	−0.58
1125	20	200	−0.52
1607	20	300	−0.57
90	25	10	−1.42
308	25	20	−0.46
824	25	100	−0.67
1125	25	150	−0.72
1607	25	250	−0.93
90	30	10	−2.67
308	30	20	−0.79
824	30	100	−1.3
1125	30	150	−1.46
1607	30	200	−1.21
90	20	20	−1.94
308	20	100	−3.36
824	20	300	−4.32
1125	20	400	−3.51
1607	20	600	−4.79
90	25	20	−5.92
306	25	40	−0.98
824	25	200	−3.38
1125	25	300	−4.57
1807	25	450	−5.56
90	30	20	−13.73
308	30	40	−1.96
824	30	*I*	−3.6
1125	30	250	−7.12
1607	30	350	−6.11
90	20	30	−6.5
306	20	140	−17.05
824	20	400	−20.57
1125	20	550	−19.73
1507	20	800	−24.52
90	25	30	−18.54
308	25	100	−16.25
824	25	300	−23.31
1125	25	400	−20.05
1607	25	600	−25.38
90	30	30	−26.61
308	30	60	−19.03
824	30	250	−30.86
1125	30	300	−17.78
1807	30	450	−22.15
90	20	50	−23.59
308	20	180	−33.97
824	20	500	−41.02
1125	20	685	−40.6
1607	20	928	−41.09
90	25	40	−26.62
308	25	160	−45.4
824	25	400	−49.3
1125	25	550	−49.85
1607	25	710	−53.26
90	30	40	−33.77
308	30	120	−44.38
824	30	350	−59.24
1125	30	500	−63.97
1607	30	685	−67.05

Table 5
Results for progressive collapse example

# Sigm.	Min. rms	Iters.	Total CPU time (100 runs)	No. of non-linear parameters
2	0.3693	52	2.39”	8
4	0.1128	167	18.71”	16
6	0.089	214	53.91”	24
8	0.0551	262	102.48”	32
10	0.0175	300	156.42	40
12	0.0053	287	205.09”	48
14	6.25 (-13)	185	251.77”	56
16	1.23 (-12)	72	240.4”	64

The data can be found in [29]. Complete hourly data for four consecutive months is provided for training, and output data for the next two months should be predicted.

The purpose of this test is to show the performance of the training algorithm and not to design the “optimal” network. The combination of a small network, a large number of data points, and the regularization provided by the combination of the Variable Projection and Marquardt algorithms hopefully guarantees that there will be no over-fitting of the training data (see [2] Chapter 9 for a more complete discussion of these issues).

We use a single hidden layer network with five sigmoids and a constant bias. Since the extrapolation that is required will not overlap in the time of the year with the training data we ignore the year and month and use only an hour count, an hour of the day count and a day of the week count as time input variables. We also include the output data of the three previous hours as memory (previous hour, first and second differences), and thus we have 10 input variables and 1 output variable. This is then a feedback network. All the variables are normalized to mean 0 and variance 1.

The resulting model when compared with the normalized training data has a residual mean error of 0.17. In Fig. 4.3 we see some detailed results of the training step, including a comparison of calculated against training output, the absolute error at the training points as a function of the hour counter, a zoomed view of the training data compared to the output of the trained network for 240 h in the middle of the training data set, and finally a scatter plot of those quantities over the whole range.

The main observation is that the global fit of this extensive and oscillatory data set with just five sigmoids (i.e., 55 non-linear and 6 linear parameters) is remarkably good and the training time per run is till only 24.2″ in average (Figs. 2 and 3).

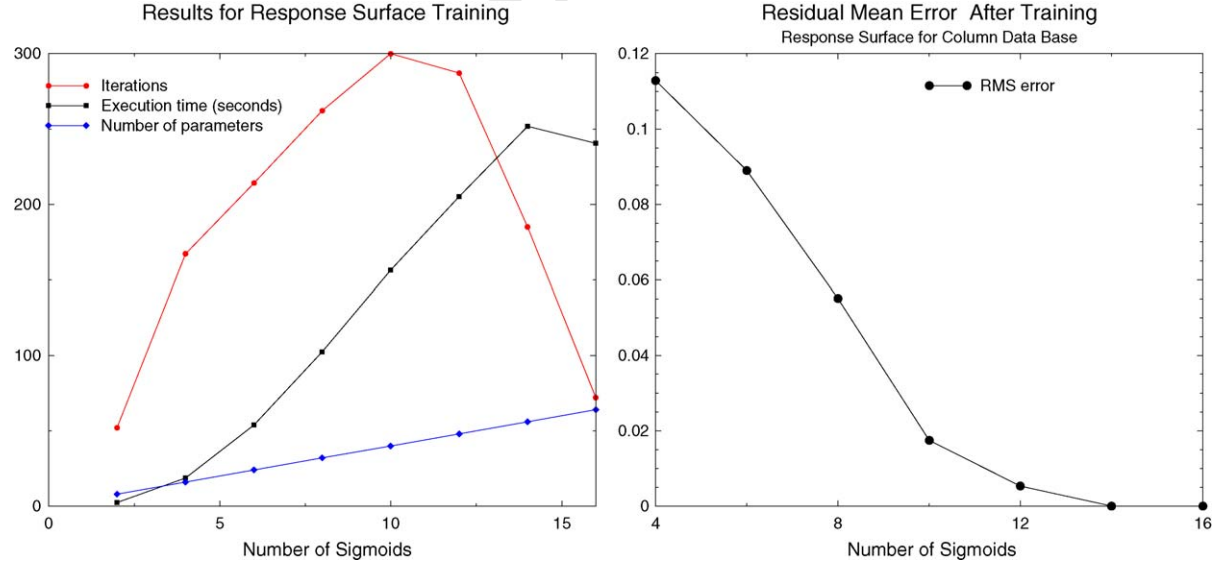

Fig. 5. Results for protective design.

4.4. Real data test

This real data test is taken from a finite-element simulation database developed by Weidlinger Associates Inc. to evaluate the progressive collapse potential of reinforced-concrete and steel frame buildings. The independent variables correspond to the configuration parameters of the structure (such as bay size, beam type, etc.) and the load applied to the structure. The dependent variable is the response of the structure (e.g., deflection) to such load. Hence, the dataset defines the load-response characteristics of the structure for particular structural configurations, as illustrated in Fig. 4.

One of the datasets taken from the database and used for illustration herein is given in Table 4. The independent variables (x_1, x_2, x_3) correspond to the beam type, the bay size and the applied load, respectively. The dependent variable, y_1, corresponds to the deflection response. Many other response quantities of interest, such as rotation, thrust and moment, are included in the database, but have been left out of the sample dataset for the sake of clarity.

We have used program NSG and different number of nodes with sigmoid activation functions, in a sigle hidden layer fully connected perceptron, with three input and one output node. In Table 5 and Fig. 5 we summarize the results on a 800 MHz Celeron PC, running under the Solaris x86 operating system.

5. Conclusions

We have presented a Variable Projection method for the fast training of certain kinds of Neural Networks. The method was implemented and tested first on synthetic examples using four different types of activation functions and three different implementations of the Variable Projection algorithm. The results indicate that all the codes perform reasonably well, with NSG having an edge on speed, although VARPRO seems more reliable, especially for sigmoid activation functions. In general NSF would not be required for this type of activation functions since derivatives are easily available.

For the benchmark test we considered a problem that has been extensively used in the literature to test and compare new Neural Net programs, *buildinga* from PROBEN [29].

Finally we have considered a real data set example, where the main purpose of the Neural Net is to build a surrogate response surface from a data base of very expensive finite element calculations. For this purpose we used NSG with different numbers of sigmoid activation functions. We saw in the results a systematic decrease in the fitting error as the number of sigmoids increased, until the number of non-linear parameters is close or above the number of data points (60), at which stage the accuracy jumps to full interpolation. We also observe, up to that point a steady increase in the computing time required, as is to be expected. Since we run 100 training iterations starting from random initial points, we can see that even for a large number of parameters the performance is very fast.

Uncited references

[23,26,27].

Acknowledgement

This research is based upon work supported by the National Science Foundation under a Phase II SBIR Grant #0321420. Submitted for publication to Applied Numerical Mathematics (July 2004).

References

[1] J. Araoz, Personal communication (1983).
[2] Ch. M. Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995.
[3] A. Björck, Numerical Methods for Least Squares, SIAM Publications, Philadelphia, 1996.
[4] M.D. Buhmann, Radial Basis Functions, Cambridge University Press, 2003.
[5] T. Coleman, B.S. Garbow, J. More, Software for estimating sparse Jacobian matrices, ACM Trans. Math. Software 10 (1984) 329–345.
[6] G. Cybenko, Approximation by superpositions of a sigmoidal function, Math. Control Signals Syst. 2 (1989) 303–314.
[7] J.E. Eriksson, M. Gulliksson, P. Linström, P-A. Wedin, Regularization tools for training feed-forward neural networks, Part I and II, J. Optim. Software 10 (1998) 49–69.
[8] D.M. Gay, Usage summary for selected optimization routines, AT&T Bell Labs. Comp. Sci. Technol. Rep. (1990).

[9] D.M. Gay, NSF and NSG; PORT Library, AT & T Bell Labs. (available from na-net), 1997.
[10] D.M. Gay, L. Kaufman, Tradeoffs in algorithms for separable nonlinear least squares, AT&T Bell Labs, Numer. Anal. Manuscript (1990) 90–111.
[11] G.H. Golub, R. LeVeque, Extensions and uses of the variable projection algorithm for solving nonlinear least squares problems, Proceedings of American Numerical Analysis and Computer Conference, 1979.
[12] G.H. Golub, C. van Loan, Matrix Computations, second ed., John Hopkins, 1989.
[13] G.H. Golub, V. Pereyra, The differentiation of pseudo-inverses and nonlinear least squares problems whose variables separate, SIAM J. Numer. Anal. 10 (1973) 413–432.
[14] G.H. Golub, V. Pereyra, Separable nonlinear least squares: the variable projection method and its applications, Inverse Problems 19 (2003) R1–R26.
[15] P. Ch. Hansen, Rank-Deficient and Discrete Ill-Posed Problems, SIAM Publications, Philadelphia, 1998.
[16] http://www.netlib.org..
[17] L. Kaufman, A variable projection method for solving separable nonlinear least squares problems, BIT 15 (1975) 49–57.
[18] L. Kaufman, Separable nonlinear least squares with multiple right hand sides, AT&T Bell Labs. Numer. Anal. Manuscript (1990) 90–103.
[19] L. Kaufman, V. Pereyra, A method for separable nonlinear least squares with separable equality constraints, SIAM J. Numer. Anal. 15 (1978) 12–20.
[20] K. Levenberg, A method for the solution of certain non-linear problems in least squares, Q. J. Appl. Math. 2 (1948) 164–168.
[21] L. Ngia, System Modeling Using Basis Functions and Applications to Echo Cancellation. Ph.D. Thesis, Chalmers Inst. of Techn., Sweden, NNSP2000, 2000.
[22] L. Ngia, Separable Nonlinear Least-Squares Methods for On-Line Estimation of Neural Nets Hammerstein Models. Manuscript, Dept. Signals and Systems, Chalmers University of Technology, Sweden, 2001.
[23] Ch. C. Paige, M.A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares, ACM Trans. Math. Software 8 (1982) 43–71.
[24] R. Parisi, D. Di Claudio, G. Orlandi, B.D. Rao, A generalized learning paradigm exploiting the structure of feedforward neural networks, IEEE Trans. Neural Networks 7 (1996).
[25] Y.C. Pati, P.S. Krishnaprasad, Analysis and synthesis of feedforward neural networks using discrete affine wavelet transformations, IEEE Trans. Neural Networks 4 (1993).
[26] V. Pereyra, Asynchronous distributed solution of large scale nonlinear inversion problems, Appl. Numer. Math. 30 (1999) 31–40.
[27] V. Pereyra, Ray tracing methods for inverse problems, Inverse Problems 16 (2000) 1–36.
[28] V. Pereyra, G. Scherer, Large least squares scattered data fitting by truncated SVD, Appl. Numer. Math. 44 (2002) 225–239.
[29] L. Prechelt, PROBEN 1—a set of benchmark neural network problems and benchmarking rules, Tech. Rep. 21, Fakultaet fuer Informatik, Universitaet Karlsruhe, 1994.
[30] A. Ruhe, P.-A. Wedin, Algorithms for separable nonlinear least squares problems, SIAM Rev. 22 (1980) 318–337.
[31] B.W. Rust, Fitting nature's basic functions part III: exponentials, sinusoids, and nonlinear least squares, Comput. Sci. Eng. (2002).
[32] S. Saarinen, R. Bramley, G. Cybenko, Ill-Conditioning in neural network training problems, SIAM J. Sci. Stat. Comput. 14 (1993) 693–714.
[33] J. Sjöberg, M. Viberg, Separable non-linear least squares minimization—possible improvements for neural net fitting, IEEE Workshop in Neural Networks for Signal Processing, Amelia Island Plantation, Fl, 1997.
[34] K. Weigl, M. Berthod, Neural Networks as Dynamical Bases in Function Space. Report #2124, INRIA, Prog. Robotique, Image et Vision. Sophia-Antipolis, France, 1993.
[35] K. Weigl, M. Berthod, Projection learning: alternative approach to the computation of the projection, Proceeding of European Symposium on Artificial Neural Networks, 19–24. Brussels,1; Belgium, 1994.
[36] K. Weigl, G. Giraudon, M. Berthod, Application of Projection Learning to the Detection of Urban Areas in SPOT Satellite Images. Report #2143, INRIA, Prog. Robotique, Image et Vision. Sophia-Antipolis, France, 1993.