PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 47, NO. 3–4, PP. 229–251 (2003)

A WEIGHTED GENERALIZED LS–SVM

József VALYON and Gábor HORVÁTH

Department of Measurement and Information Systems
Budapest University of Technology and Economics
H–1521, Budapest, Hungary
P. O. Box 91.
e-mail: valyon@mit.bme.hu, horvath@mit.bme.hu

Received: December 5, 2003

Abstract

Neural networks play an important role in system modelling. This is especially true if model building is mainly based on observed data. Among neural models the Support Vector Machine (SVM) solutions are attracting increasing attention, mostly because they automatically answer certain crucial questions involved by neural network construction. They derive an 'optimal' network structure and answer the most important question related to the 'quality' of the resulted network. The main drawback of standard Support Vector Machines (SVM) is its high computational complexity, therefore recently a new technique, the Least Squares SVM (LS–SVM) has been introduced. This is algorithmically more effective, because the solution can be obtained by solving a linear equation set instead of a computation-intensive quadratic programming problem. Although the gain in efficiency is rather significant, for really large problems the computational burden of LS-SVM is still too high. Moreover, an attractive feature of SVM, its sparseness is lost. This paper proposes a special new generalized formulation and solution technique for the standard LS-SVM. By solving the modified LS–SVM equation set in least squares (LS) sense (LS^2–SVM), a pruned solution is achieved, while the computational burden is further reduced (Generalized LS–SVM). In this generalized LS–SVM framework a further modification weighting is also proposed, to reduce the sensitivity of the network construction to outliers while maintaining sparseness.

Keywords: function estimation, least squares support vector machines, regression, support vector machines, system modelling.

1. Introduction

System modelling is an important way of investigating and understanding the world around. There are several different ways of building system models, and these ways utilize different forms of knowledge about the system. When only input-output observations are used, a behavioral or black box model can be constructed. In black box modelling neural networks play an important role.

The most important questions of neural networks are about (i) their modelling capabilities: what input–output relations can be implemented using a neural network, and (ii) their generalization capabilities: what are the answers of a trained network for inputs not used in its construction, not used during training.

The main reason of the importance of Neural Networks comes from their general modelling capabilities. Some of the neural network architectures (e.g.

multi-layer perceptrons, MLPs [1], [2], or radial basis function – RBF-networks) are universal approximators [1]–[3], this means that an MLP or an RBF of proper size can approximate any continuous function arbitrarily well [3]. A neural network is trained using a finite number of training examples and the goal is that the network give correct responses for inputs not used during training. Unfortunately, the training data is often corrupted by noise, which – if not handled properly – misleads the training.

Although there are theoretical results about the modelling capability of neural networks, some important questions are not answered yet. One of these questions is about the size of the network. What complexity network has to be used for a given modelling task? Another important question is about the generalization capability of a network. These questions are very difficult, theoretical answers that can also be used in practice cannot be found in the classic neural network field.

Recently new approaches of learning machine construction, the Support Vector Machines (SVM) [4]–[11] and their least squares modification the LS–SVM [12]–[18] have been introduced and are gaining more and more attention, because they incorporate some useful features that make them favorable in handling the above described situations.

The result of both methods can be interpreted as a neural network, as it will be shown later.

The primary advantage of the SVM method is that for a given problem it automatically derives the 'optimal' network structure (in respect of generalization error). In practice it means that several decisions that had to be made during the design of a traditional NN like the decisions about

- the number of neurons,
- the structure of the network,
- the length of the learning cycle,
- the type of the learning process.

etc. are eliminated. Another benefit of this method is that the resulting network guarantees an upper bound on the generalization errors [4], [5]. While these questions are eliminated, some knowledge is gained about the result, which assures us about the generalization performance. The SVM method was originally established by VAPNIK [1].

According to the Structural Risk Minimization [4]–[11] principle, involved by the construction of an SVM, the generalization error rate is upper bounded by a formula containing the training error and the Vapnik–Chervonenkis (VC) dimension, which describes the capacity – ability to approximate complex functions – of the network.

By minimizing this formula, an SVM produces a reasonably simple network, which assures a low upper limit of the generalization error. On the other hand, the construction of an SVM needs the solution of a convex constrained optimization problem. The solution can be obtained via quadratic programming (QP) which is a rather computation-intensive and memory-consuming method, especially if a large

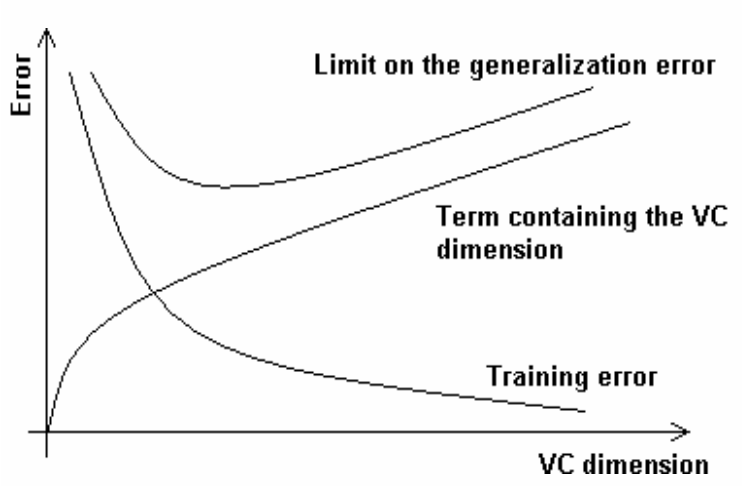

Fig. 1. Illustration of the Structural Risk Minimization Principle

training set of high-dimensional data are used. There are several iterative solutions to speed up this process [19]–[25], however, a faster method is still needed.

The least squares Support Vector Machine is attracting more and more attention, because its construction requires only the solution of a linear equation set instead of the long and computationally hard quadratic programming problem. Unfortunately, there are some drawbacks of using LS–SVM. While the least squares version incorporates all training data in the network to produce the result, the traditional SVM selects some of them (the support vectors) that are important in the regression. This sparseness of traditional SVM can also be reached with the LS-SVM by applying a *pruning* method [17, 18], but this requires the entire large problem to be solved at least once. Another possibility is the use of the *fixed LS–SVM* method, which is an iterative method for constructing an LS–SVM network of a predefined size [16].

The LS–SVM method should also be able to handle outliers (e.g. resulting from an additive non-Gaussian noise, such as a heavy-tail distribution). A modification of the method, called *weighted LS–SVM*, is aimed at reducing the effects of this type of noise. The biggest problem is that pruning and weighting – although their goals do not rule out each other – cannot be used at the same time, because the algorithms work in opposition.

This paper presents a generalized approach by allowing a more universal construction and formulation of the kernel matrix or more precisely, the LS–SVM equation set. Earlier in refs. [26, 27] we proposed the least squares modification of the LS–SVM (LS^2–SVM) which provided a sparse solution. This method is generalized further and is extended with weighting. Our objectives include noise reduction, sparseness and further reduction of algorithmic complexity while main-

taining the quality of the results. The main topic of this paper is the weighted extension of the sparse LS^2–SVM, so the described method enables us to accomplish both goals at the same time.

Both the SVM and LS–SVM methods are capable of solving both, classification and regression problems. The classification approach is easier to understand and more historic. The present study concerns regression, therefore only this is introduced in the sequel, along with the most common additional methods. Before going into the details, the main and distinguishing features of the basic procedures are summarized. Sections 2 and 3 describe the SVM and LS–SVM regression. Section 4 presents the weighted LS–SVM. The LS–SVM method is generalized in section 5, which is enhanced with weighting in section 6. Some experimental results are presented afterwards.

2. The SVM Method for Regression

The goal of regression is to approximate a $d = g(\mathbf{x})$ function, based on a training data set $\{\mathbf{x}_i, d_i\}_{i=1}^N$, where $\mathbf{x}_i \in \Re^p$ represents a p-dimensional input vector and $d_i \in \Re$ is the scalar target output. Our goal is to construct an $y = f(\mathbf{x})$ approximating function which represents the dependence of the d training targets on the $\mathbf{x}$ inputs.

To start with, a *loss function* must be defined to represent the cost of deviation from the target output d_i for each $\mathbf{x}_i$ input. In most cases the ε-insensitive loss function (L_ε) is used, but one can use other (e.g. non-linear) loss functions, too, such as given in [6]. The ε-insensitive loss function, shown in *Fig.2*, is:

$$L_\varepsilon(d,\ f(\mathbf{x})) = \begin{cases} 0 & \text{for } |f(\mathbf{x}) - d| < \varepsilon \\ |f(\mathbf{x}) - d| - \varepsilon & \text{otherwise} \end{cases} . \tag{1}$$

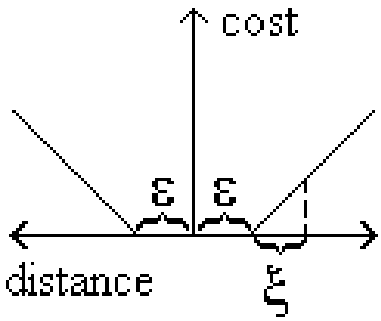

Fig. 2. The ε-insensitive loss function

In this case approximation errors smaller than ε are ignored, while the larger ones are punished in a linear way.

The estimator function f is defined as follows:

$$y = f(\mathbf{x}) = \sum_{j=1}^{h} w_j \varphi_j(\mathbf{x}) + b = \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}) + b,$$
$$\mathbf{w} = [w_1, w_2, \ldots, w_h]^T, \qquad \boldsymbol{\varphi} = [\varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), \ldots, \varphi_h(\mathbf{x})]^T. \tag{2}$$

The $\varphi(.) : \Re^p \to \Re^h$ mapping is a mostly non-linear function, which transforms the data into a higher (possibly infinite – h) dimensional feature space. Our f function should minimize the risk functional defined as

$$R[f] = \int L(d,\ f(\mathbf{x}))\, P(\mathbf{x}, y)\, \mathrm{d}\mathbf{x}\, \mathrm{d}y, \tag{3}$$

but unfortunately the $P(\mathbf{x}, y)$ probability density function is almost never known. Under certain conditions, defined in [4], [5], *Eq.* (2) may be replaced by an *empirical risk functional*:

$$R_{\text{emp}}[f] = \frac{1}{N} \sum_{i=1}^{N} L(d_i,\ f(\mathbf{x}_i)). \tag{4}$$

This should be minimized, along with the use of the above described ε-insensitive loss function $L_\varepsilon(d_i,\ f(\mathbf{x}_i))$, and also subject to the constraint of $\|\mathbf{w}\|^2 \le c_0$ to keep $\mathbf{w}$ as short as possible (c_0 is a constant). The minimization of $\|\mathbf{w}\|$ corresponds to the minimization of the Vapnik–Chervonenkis VC dimension [4, 5]. *Eq.* (5) shows the constraints defined by the training points, where $\{\xi_i\}_{i=1}^N$ and $\{\xi_i'\}_{i=1}^N$ slack variables are introduced, to represent the cost of points outside the ε insensitive boundary:

$$\begin{aligned} d_i - \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) - b &\le \varepsilon + \xi_i, \\ \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b - d_i &\le \varepsilon + \xi_i', \\ \xi_i \ge 0, \quad \xi_i' \ge 0, \quad i &= 1, \ldots, N. \end{aligned} \tag{5}$$

The measure of this cost is determined by the loss function. The complex optimization of SVM is solved by minimizing the following equation in $\mathbf{w}$.

$$F(\mathbf{w}, \xi, \xi') = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\left(\sum_{i=1}^{N}(\xi_i + \xi_i')\right)$$
$$\text{with constraints:} \tag{6}$$
$$d_i - \mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) - b \le \varepsilon + \xi_i, \qquad \xi_i \ge 0,$$
$$\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b - d_i \le \varepsilon + \xi', \qquad \xi_i' \ge 0, \qquad i = 1, \ldots, N.$$

The $\mathbf{w}^T\mathbf{w}$ term stands for minimizing the length of the weight vector, while C constant is the trade-off parameter between this and the minimization of training

data errors. This constrained optimization can be formulated as a Lagrangian:

$$J(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}', \boldsymbol{\alpha}, \boldsymbol{\alpha}', \boldsymbol{\gamma}, \boldsymbol{\gamma}') = C \sum_{i=1}^{N} (\xi_i + \xi_i') + \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N} \alpha_i \left[\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) + b - d_i + \varepsilon + \xi_i\right] - \sum_{i=1}^{N} \alpha_i' \left[-\mathbf{w}^T \boldsymbol{\varphi}(\mathbf{x}_i) - b + d_i + \varepsilon + \xi_i'\right] - \sum_{i=1}^{N} \left(\gamma_i \xi_i + \gamma_i' \xi_i'\right) \tag{7}$$

with $\alpha_i \geq 0, \alpha_i' \geq 0$ and $\gamma_i \geq 0, \gamma_i' \geq 0$ Lagrange multipliers ($i = 1, \ldots, N$). The result is given by the saddle point:

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\gamma}, \boldsymbol{\gamma}'} \min_{\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}'} J(\mathbf{w}, \boldsymbol{\xi}, \boldsymbol{\xi}', \boldsymbol{\alpha}, \boldsymbol{\alpha}', \boldsymbol{\gamma}, \boldsymbol{\gamma}'). \tag{8}$$

The primal problem deals with convex cost function and linear constraint, therefore from this constrained optimization problem a dual problem can be constructed, in which the Karush–Kuhn–Tucker (KKT) conditions [28] are used.

$$\max_{\boldsymbol{\alpha}, \boldsymbol{\alpha}'} Q(\alpha, \alpha') = \sum_{i=1}^{N} d_i(\alpha_i + \alpha_i) - \varepsilon \sum_{i=1}^{N} (\alpha_i + \alpha_i') - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} (\alpha_i - \alpha_i')(\alpha_j - \alpha_j') K(\mathbf{x}_i, \mathbf{x}_j)$$

with constraints:

$$\sum_{i=1}^{N} (\alpha_i + \alpha_i') = 0, \qquad 0 \leq \alpha_i \leq C, \qquad 0 \leq \alpha_i' \leq C, \qquad i = 1, \ldots, N. \tag{9}$$

where $K(\mathbf{x}_i, \mathbf{x}_j) = \boldsymbol{\varphi}^T(\mathbf{x}_i)\boldsymbol{\varphi}(\mathbf{x}_j)$ is the inner product kernel function.

Finally, the values of f are calculated from *Eq.* (10), where α_i and α_i' are determined by *quadratic programming* (QP) from *Eq.* (9).

$$y = \sum_{i=1}^{N} (\alpha_i - \alpha_i') K(\mathbf{x}, \mathbf{x}_i) + b. \tag{10}$$

This most frequently used kernels are shown in *Table 1* [4].

The support vectors are the input data points corresponding to the $((\alpha_i - \alpha_i'),$ $i = 1, \ldots, N)$ non-zero multipliers. The bias b can be calculated from the KKT conditions [4]–[18].

Table 1. The most typical kernel functions

Linear SVM	$@K(\mathbf{x}, \mathbf{x}_i) = \mathbf{x}_i^T \mathbf{x}$
Polynomial SVM of degree n	$@K(\mathbf{x}, \mathbf{x}_i) = \left(\mathbf{x}_i^T \mathbf{x} + 1\right)^n$
RBF SVM	$@K(\mathbf{x}, \mathbf{x}_i) = \exp\left\{-\left\|\mathbf{x} - \mathbf{x}_i\right\|^2 / \sigma^2\right\}$ where σ is a constant.
MLP SVM	$@K(\mathbf{x}, \mathbf{x}_i) = \tanh\left(i\mathbf{x}_i^T \mathbf{x} + \theta\right)$, i and θ are properly chosen constants, since not all combinations may be used.

The user-defined parameters C and ε control the smoothness of the resulting function. We must also choose the parameters of the selected kernel. In the case of an RBF structure it means the selection of a suitable σ or a $\boldsymbol{\sigma}$ vector. In practice, it's very hard to determine the optimal values for these three parameters, because no universal approach is available. This paper does not discuss these problems, but some results can be found in refs. [29]–[31].

Support vector machines can be interpreted as neural networks, although in practice the results are rarely formulated as actual networks. However, the neural interpretation is important, because it provides an easier discussion framework than the purely mathematical point of view. Training and operating a support vector machine is a series of mathematical calculations, but the equation used for determining the answer represents exactly the same calculations as a one hidden layer neural network.

The hidden layer typically consists of non-linear neurons. *Fig.3* illustrates a neural network that can be considered as a Support Vector Machine.

The input is an M-dimensional vector. The nonlinear kernel functions are used in the hidden layer neurons. The number of these nonlinear neurons equals to the number of selected support vectors (*N-Network size*). The result (y) is the weighted sum of the outputs of the middle layer neurons. The weights are the calculated $\left(\alpha_i - \alpha_i'\right)$ Lagrange multipliers (*Weighting*). Accordingly, the smaller the network, the less calculations are required for getting an answer; therefore the goal is to reach the smallest possible network size. Since the network size determines the amount of calculations needed in the recall phase, this may be referred to as the complexity of the result. This complexity differs from the algorithmic complexity of the method used to reach this result!

This paper reasons with the neural interpretation throughout the discussions, because the points and statements of this work can be more easily understood from this neural point of view.

The main problem with the traditional SVM method is its high algorithmic complexity, namely its slow construction and extensive memory requirements. To overcome these problems, several modifications of the method have been proposed.

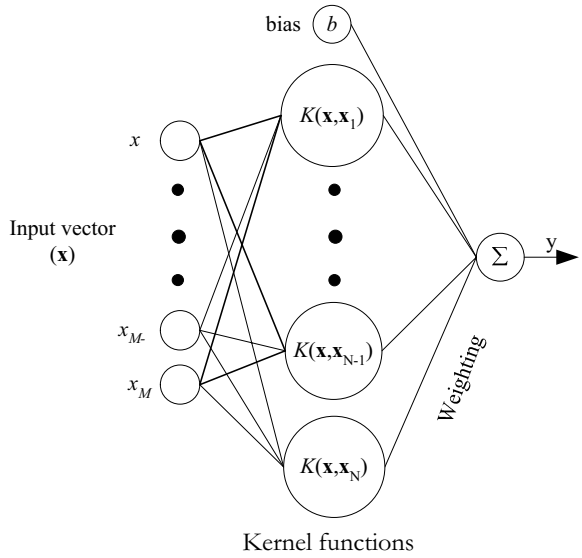

Fig. 3. The neural interpretation of a Support Vector Machine

These algorithms are mostly iterative methods that decompose the large problem into smaller optimization tasks [19]–[25]. These methods are commonly known as 'chunking' algorithms, where the methods mainly differ in the way they determine the decomposed sub-problems. The traditional 'chunking' may not reduce the problem enough, therefore different modifications are available. The two main techniques are based on Osuna's algorithm and SMO [25]. OSUNA et al. suggest maximizing the reduced QP sub-problems of a fixed size. The Sequential Minimal Optimization (SMO) brakes up the large quadratic programming task into a series of the smallest possible QP problems, which can be solved analytically [21]. These small problems consist of only two Lagrange multipliers, which are jointly optimized at every iteration. Successive overrelaxation (SOR) has also been applied to large SVM problems [25].

Another way to overcome the problem of algorithmic complexity is the use of the LS–SVM described below. The LS–SVM solves this problem by replacing the quadratic programming with a simple matrix inversion.

3. The LS–SVM Method

The basic idea is exactly the same as the one described above [12]–[18]. In the least squares support vector machine regression, the ε-insensitive loss function is replaced by a quadratic cost function. The main difference from the standard SVM

is in the constraints. The optimization problem and the inequality constraints are replaced by the following equations ($i = 1, \ldots, N$):

$$\min_{\mathbf{w},b,\mathbf{e}} J_p(\mathbf{w}, \mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\frac{1}{2}\sum_{i=1}^{N} e_i^2,$$
$$\text{with constraints: } d_i = \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i, \text{ where } i = 1, \ldots, N. \tag{11}$$

Again the first term stands for the minimization of the VC dimension, while the second one minimizes the training errors (C is the trade-off parameter between the terms).

From this, the following Lagrangian can be formed:

$$L(\mathbf{w}, b, \mathbf{e}; \boldsymbol{\alpha}) = J_p(\mathbf{w}, \mathbf{e}) - \sum_{i=1}^{N} \alpha_i \left\{\mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i\right\}, \tag{12}$$

where the α_i parameters are the Lagrange multipliers. The solution concludes in a constrained optimization, where the conditions for optimality are the following:

$$\begin{aligned}
\frac{\partial L}{\partial \mathbf{w}} = 0 \quad &\rightarrow \quad \mathbf{w} = \sum_{i=1}^{N} \alpha_i \boldsymbol{\varphi}(\mathbf{x}_i) \\
\frac{\partial L}{\partial b} = 0 \quad &\rightarrow \quad \sum_{i=1}^{N} \alpha_i = 0 \\
\frac{\partial L}{\partial e_i} = 0 \quad &\rightarrow \quad \alpha_i = C e_i \qquad i = 1, \ldots, N \\
\frac{\partial L}{\partial \alpha_i} = 0 \quad &\rightarrow \quad \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i - d_i = 0 \qquad i = 1, \ldots, N.
\end{aligned} \tag{13}$$

This leads to the following linear equation set:

$$\begin{bmatrix} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \vec{\mathbf{1}} & \boldsymbol{\Omega} + C^{-1}\mathbf{I} \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix}, \quad \mathbf{d} = [d_1, d_2, \ldots, d_N]^T,$$
$$\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]^T, \quad \vec{\mathbf{1}} = [1, \ldots, 1]^T, \quad \Omega_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j), \tag{14}$$

where $C \in \Re$ is a positive constant, b is the bias and the result is: $y = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i) + b$. This result can also be interpreted as a neural network, which contains N non-linear neurons in its single hidden layer.

The LS–SVM method – when RBF kernels are used – requires only two parameters (C and σ), while the time consumed by the learning method is reduced, by replacing the quadratic optimization problem with a simple *linear equation set*.

If N is the number of training points, then the matrix representing the linear *Eq.* (14) is of size $(N + 1) \times (N + 1)$. For large training data sets this matrix

cannot be stored in memory, therefore an iterative solution is needed. For an easier discussion, the notifications of *Eq.* (14) will be simplified as follows.

$$\mathbf{A} = \begin{bmatrix} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \vec{\mathbf{1}} & \mathbf{\Omega} + C^{-1}\mathbf{I} \end{bmatrix}, \qquad \mathbf{u} = \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix}. \tag{15}$$

The Hestens–Stiefel conjugate gradient method for solving $\mathbf{Au} = \mathbf{v}$, where $\mathbf{A} \in \Re^{N \times N}$ and $\mathbf{b} \in \Re^{N}$ can be applied. For convergence $\mathbf{A}$ should be positive definite, therefore the system must be first transformed to meet this condition [16]. The convergence of the conjugate gradient algorithm depends on the condition number of the matrix which is influenced by parameter C.

So far we have described the algorithmic complexity of the solution. In the sequel another complexity property is described, namely the complexity (size) of the resulting solution (network).

The problem with the above described solution is that the result is not sparse. The loss of sparseness is very important, especially in the light of the equivalence between SVMs and sparse approximation [16]–[18]. Practically this means that the net consists of – in its hidden layer – as many neurons as the number of training vectors used. This means an unnecessarily large network, and therefore more calculations for every result in the recall phase. To overcome this problem, the following pruning method was introduced. Pruning techniques are also well-known in the context of traditional neural networks. Their purpose is to reduce the complexity of the networks by eliminating as much hidden neurons as possible.

LS–SVM pruning: One of the main drawbacks of the least squares solution is that the solution is not sparse in the sense that it incorporates all training vectors in the resulting network. In the traditional SVM the result usually contains many zero multiplier (the weights $(\alpha_i - \alpha_i') = 0$ – see neural interpretation in Section 3) values. In LS–SVM pruning all necessary informations are obtained from the solution of the linear system [17]–[19].

The weighting of the least squares SVM reflects the importance of the inputs, therefore by eliminating some vectors, represented by the smallest values from this $|\alpha_i|$ spectrum, the number of neurons can be reduced. The support values are proportional to the errors at data points:

$$\alpha_i = Ce_i \tag{16}$$

The irrelevant points are left out, by iteratively leaving out the least significant vectors. These are the ones corresponding to the smallest $|\alpha_i|$ values. The algorithm is the following [16]:

1. Train the LS–SVM based on N points. (N is the number of all available training vectors.)
2. Remove a small amount of points (e.g. 5% of the set) with the smallest values in the sorted $|\alpha_i|$ spectrum.
3. Re-train the LS–SVM based on the reduced training set.

4. Go to 2, unless the user-defined performance index degrades. If the performance becomes worse, it should be checked whether an additional modification of C, σ might improve the performance.

In the case of the classic SVM, sparseness is achieved by the use of such loss functions where errors smaller than ε are ignored (e.g. ε-insensitive loss function). This method reduces the difference between SVM and LS–SVM, because the omission of some data points implicitly corresponds to creating an ε-insensitive zone [19].

The described method leads to a sparse model, but some questions arise: How many neurons are needed in the final model? How many iterations are necessary to reach the final model? According to our experiments the number of iterations and the results do not seem to be related. If the points are omitted in more than one step, the results are not necessarily better than in the case when the reduction is done in one step.

Another problem is that a usually large linear system must be solved in each iteration. The pruning is especially important if the number of training vectors is large. In this case, however, the iterative method is not very effective. Our proposed method the LS^2–SVM, described in section 5, leads to a sparse solution, automatically answers the questions and solves the problem described above.

4. The Weighted LS–SVM Method

In this section, the weighted extension of the original LS–SVM is presented [16], which was introduced to diminish the effects of outliers.

In the weighted LS–SVM, the importance of each constraint is modified by a v_i weight factor:

$$\min_{\mathbf{w},b,\mathbf{e}} J_p(\mathbf{w},\mathbf{e}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\frac{1}{2}\sum_{i=1}^{N} v_i e_i^2,$$
$$\text{and } d_i = \mathbf{w}^T\boldsymbol{\varphi}(\mathbf{x}_i) + b + e_i, \text{ where } i = 1,\ldots,N. \tag{17}$$

The weighted solution concludes in a constrained optimization which can be formulated as the following equation set:

$$\begin{bmatrix} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \vec{\mathbf{1}} & \boldsymbol{\Omega} + \mathbf{V}_\gamma \end{bmatrix} \begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{d} \end{bmatrix}, \quad \mathbf{d} = [d_1, d_2, \ldots, d_N]^T,$$
$$\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]^T, \quad \vec{\mathbf{1}} = [1,\ldots,1]^T, \quad \Omega_{i,j} = K(\mathbf{x}_i,\mathbf{x}_j),$$
$$\mathbf{V}_\gamma = \text{diag}\,([1/Cv_1;\ldots;Cv_N]), \tag{18}$$

where $C \in \Re$ is a positive constant, v_i weights are determined according to the $e_i = \alpha_i/C$ equation, b is the bias and the result is the well-known $y = \sum_{i=1}^{N} \alpha_i K(\mathbf{x},\mathbf{x}_i) + b$.

The effects of outliers are reduced iteratively, by using a weighting factor in the calculation based on the error variables determined from a previous – first unweighted – solution. The weighting is designed in such a way, that the results improve in view of robust statistics. A large e_i means a small weight and vice versa. A more detailed description can be found in ref. [17].

The same pruning method can be used as described earlier for the unweighted case.

5. The Proposed Generalization

A. The generalized least squares LS–SVM method

If the training set consists of N samples, then our original linear equation set will have $(N+1)$ unknowns, the α_i-s, $(N+1)$ equations and $(N+1)^2$ multiplication coefficients. These factors are mostly the values of the $K(\mathbf{x}_i, \mathbf{x}_j)$ kernel function calculated for every combination of the training inputs. The cardinality of the training set therefore determines the size of the coefficient matrix, which plays a major part in the solution as the computational complexity of both the training and recall phase depends on this. It is easy to see that, in order to reduce network size and algorithmic complexity, this matrix has to be manipulated. Let's take a closer look at the linear equation set describing the problem.

$$\left[\begin{array}{c|c} 0 & \vec{\mathbf{1}}^{\mathbf{T}} \\ \hline \vec{\mathbf{1}} & \boldsymbol{\Omega} + C^{-1}\mathbf{I} \end{array}\right] \left[\begin{array}{c} b \\ \boldsymbol{\alpha} \end{array}\right] = \left[\begin{array}{c} 0 \\ \mathbf{d} \end{array}\right]. \tag{19}$$

The first row means:

$$\sum_{i=1}^{N} \alpha_i = 0, \tag{20}$$

and the jth row stands for the

$$b + \alpha_1 K(\mathbf{x}_j, \mathbf{x}_1) + \ldots + \alpha_j [K(\mathbf{x}_j, \mathbf{x}_j) + C^{-1}] + \ldots + \alpha_N K(\mathbf{x}_j, \mathbf{x}_N) = d_j \tag{21}$$

condition.

The most important component of the main matrix is $\boldsymbol{\Omega}$ whose every element is a result of the kernel function for two training inputs:

$$\Omega_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j). \tag{22}$$

In order to reduce the number of elements, some of the training samples should usually be omitted (see *Fig. 3*). This is the case in traditional pruning of LS–SVM when by entirely deleting the insignificant samples a smaller kernel matrix is obtained.

A training vector however, can be ignored in three different ways: the corresponding column, the corresponding row, or both (column and row) may be eliminated.

Each column stands for a neuron, with a kernel centered at the corresponding input. If the ith column is left out, then the corresponding α_i is also deleted, therefore the resulting network will be smaller. The first row's condition automatically adapts, since the remaining α-s will still add up to zero.

However, the rows declare the input–output relations, represented by the training points, that the solution must satisfy. If the jth row is deleted, then the condition defined by the $(\mathbf{x}_j, d_j)$ training point is lost, because the jth equation is removed. This may be useful in the case of noisy samples, but in this case the number of columns must also be reduced, otherwise the equation set becomes underdetermined. This noise reduction technique is described in detail in [27].

It can be seen that the network size depends on the number of columns only, therefore to reach a sparse solution, the number of columns must be reduced. This means that for this purpose two possible reduction techniques may be applied to the equation set:

- *Full reduction* – a training sample $(\mathbf{x}_i, d_i)$ is fully omitted, therefore both the columns and rows corresponding to this sample are eliminated.
- *Partial reduction* – a training sample $(\mathbf{x}_i, d_i)$ is only partially omitted, by eliminating the corresponding ith column only, but keeping the ith row which defines constraints. It means that the weighted sum of that row should still meat the d_i goal (as closely as possible).

If *full reduction* is applied – meaning that only a subset of the training vectors will play part in the solution – then these vectors must be the ones most accurately representing the function. The vectors corrupted with the least amount of noise seem to be the best choice. In this case, however, reduction means that the information embedded in the omitted samples are lost. The next figure demonstrates how the equation changes if full reduction is applied. The deleted elements are coloured gray. Since rows and columns are omitted, the main matrix shrinks in both directions.

When traditional pruning is applied to the LS–SVM solution, this is exactly the case, because pruning iteratively drops some training points. The information embodied in this subset is entirely lost.

To avoid this loss of information, a *partial reduction* technique can be used. This proposition resembles to the basis of the Reduced Support Vector Machines (RSVM) introduced for standard SVM classification in ref. [32]. In the case of partial reduction, the omission of a training sample means that only the corresponding column is eliminated, while the row is kept. By selecting some (e.g. M, $M < N$) vectors as '*support vectors*', the number of variables (α) is reduced, resulting in more equations than unknowns. The effect of partial reduction is shown in the next figure where the removed elements are coloured gray.

By applying this *partial reduction* our problem becomes overdetermined, which can be solved as a *linear least squares problem*, consisting of only $(N+1)\times$

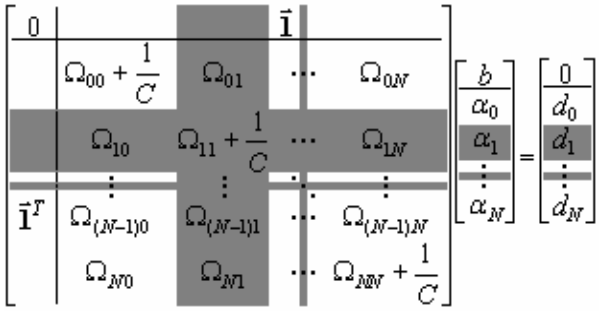

Fig. 4. The effect of full reduction

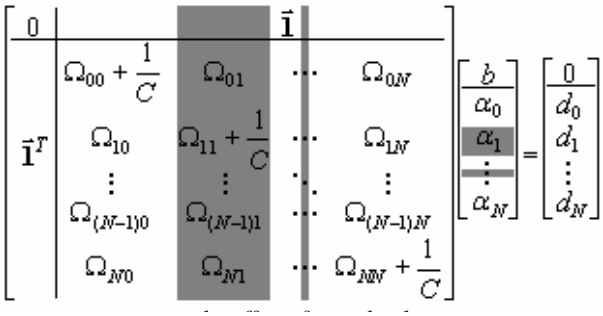

Fig. 5. The effect of partial reduction

$(M+1)$ coefficients.

In the equation set, every α variable stands for a neuron – representing its weight – and each one of the M selected training vectors will be a center of a kernel function, therefore these inputs must be chosen accordingly. This means that the following question must be answered: *How many and which vectors are needed?*

Standard SVM automatically selects a subset of the training points as support vectors. The linear equation set involved by the LS–SVM has to be reduced to an overdetermined equation set in such a way, that the solution of this reduced problem is the closest to what the original solution would be. This whole reduction method can be interpreted as follows: Let's select a linearly independent subset of the column vectors and omit all the others that can be formed as linear combinations of the selected ones. This can be easily done by finding a 'basis' (quotation marks indicate that this basis is only true under certain conditions defined later) of the coefficient matrix (**A**), which is by definition the smallest set of vectors enough to solve the problem. A slight modification of a common mathematical method – used for bringing the matrix to the reduced row echelon form – can be utilized to find this 'basis'. This is discussed in more detail in the sequel.

The basic idea of feature selection in the kernel space is not new. The nonlinear

principal component analysis technique, the Kernel PCA is based on the same idea [33]–[36]. A possible selection method for finding the basis of a kernel matrix has been shown in ref. [37].

This reduced input set (the support vectors) is (are) selected automatically by determining a 'basis' of the $\mathbf{\Omega}$ (or the $\mathbf{\Omega} + C^{-1}\mathbf{I}$) matrix. This can be easily shown as follows:

An $\mathbf{Au} = \mathbf{v}$ linear equation defines $\mathbf{v}$ as the weighted sum of the column vectors of $\mathbf{A}$. The equation set has a solution if and only if $\mathbf{v}$ is in the space spanned by columns of $\mathbf{A}$. Every solution ($\mathbf{x}$) means a possible decomposition of $\mathbf{v}$ to these vectors. The solution is unique if and only if the columns of $\mathbf{A}$ are linearly independent, therefore by determining a basis of $\mathbf{A}$ – any set of vectors that are linearly independent, and span the same space as $\mathbf{A}$ – the problem is reduced to a weighted sum of the basis vectors.

The linear dependence discussed above, does not mean exact linear dependence, because the method uses an adjustable tolerance value when determining the 'resemblance' (parallelism) of the column vectors. The use of this tolerance value is essential, because it is unlikely that the columns of $\mathbf{\Omega}$ will be exactly dependent (parallel). This tolerance (ε') can be related to the ε parameter of the standard SVM, because it has similar effects. If the chosen tolerance value is too small, a lot of vectors will form the basis and therefore a larger network will be obtained. The larger the tolerance, the fewer vectors will be selected. As it was shown earlier, the sparseness of standard SVM is due to the ε-insensitive loss function which neglects the samples falling inside the ε-insensitive zone. Keeping this in mind, it may not be very surprising to find that an additional parameter is needed to achieve sparseness in LS–SVM. This parameter corresponds to the one omitted originally when changing from the SVM to the standard least squares solution.

This selection process incorporates a parameter which indirectly controls the number of resulting basis vectors (M). Since M is the number of linearly independent columns, this number does not really depend on the training sample number (N), only on the problem. In practice it means that no matter how many training samples are presented, if the problems complexity requires M neurons, the size of the resulting network does not change.

The basis is achieved through transforming the $\mathbf{\Omega}$ matrix into reduced row echelon form [38], where the tolerance (ε') is used in the rank tests. The algorithm uses elementary row operations [39, 40]:

- Interchange of two rows.
- Multiply one row by a nonzero number.
- Add a multiple of one row to a different row.

The algorithm operates as follows [38]:

1. Loop over the entire matrix (i – row index, j – column index).
2. Determine the largest element p in column j with row index $i \geq j$.
3. If $p \leq \varepsilon'$ (where ε' is the tolerance), then zero out this part of the matrix (elements in the jth row with index $i \geq j$);

or else remember the column index because we found a base vector (support vector), and divide the row with the pivot element p and subtract the row from all other rows.

4. Step forward to $i = i + 1$ and $j = j + 1$. Go to step 1.

This method returns a list of the column vectors which are linearly independent considering tolerance ε'.

Each column (j) stands for a neuron, with a kernel centered on the corresponding input ($\mathbf{x}_j$). The formulation of this matrix can be generalized as follows:

- The kernels may be centered around any point (not just input samples), so the columns may be represented by any chosen $\mathbf{c}_j$ vector. For example, the simplest construction of a fixed LS–SVM is to define the centers (e.g. M uniformly positioned vectors), and solve the equation set formulated accordingly (see *Eq.* (13)).
- The kernel functions may be different from column to column.

The formulation of $\mathbf{\Omega}$ changes as follows:

$$\Omega_{i,j} = K_j(\mathbf{x}_i, \mathbf{c}_j) \tag{23}$$

and the result will be calculated from $y = \sum_{i=1}^{M} \alpha_i K_i(\mathbf{x}, \mathbf{c}_i) + b$, where M is the number of kernels used.

It is also important to emphasize that the number of columns will be less than the number of rows ($M < N$). This leads to an overdetermined equation set, which can be solved as a *linear least squares problem* consisting of only $(M+1) \times (N+1)$ coefficients.

$$\left[\begin{array}{c|ccc} 0 & \multicolumn{3}{c}{\vec{\mathbf{1}}^{\mathbf{T}}} \\ \hline & K_1(\mathbf{x}_1, \mathbf{c}_1) + C^{-1} & \cdots & K_M(\mathbf{x}_1, \mathbf{c}_M) \\ & \vdots & \ddots & \vdots \\ \vec{\mathbf{1}} & K_1(\mathbf{x}_M, \mathbf{c}_1) & \cdots & K_M(\mathbf{x}_M, \mathbf{c}_M) + C^{-1} \\ & \vdots & \ddots & \\ & K_1(\mathbf{x}_N, \mathbf{c}_1) & \cdots & K_M(\mathbf{x}_N, \mathbf{c}_M) \end{array}\right] \left[\begin{array}{c} b \\ \hline \alpha_1 \\ \vdots \\ \alpha_M \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline d_1 \\ \vdots \\ d_M \\ \vdots \\ d_N \end{array}\right]. \tag{24}$$

As seen earlier in *Eq.* (15), this equation set is written shortly as $\mathbf{Au} = \mathbf{v}$, where $\mathbf{A}$, $\mathbf{u}$ and $\mathbf{v}$ are the matrixes of *Eq.* (24), respectively.

There is a slight problem with the regularization parameter C, since it can only be inserted in the first M rows. This does not exactly reflect the same theoretical meaning as in the original *Eq.* (3), but it is enough to ensure us M linearly independent rows, so the equation set can be solved.

In theory, the solution can be written as

$$\mathbf{A}^T\mathbf{Au} = \mathbf{A}^T\mathbf{v}. \tag{25}$$

The modified matrix $\mathbf{A}$ has $(N+1)$ rows and $(M+1)$ columns. After the matrix multiplications the results are obtained from a reduced equation set incorporating $\mathbf{A}^T\mathbf{A}$ which is only of size $(M+1)\times(M+1)$. Reducing only the number of columns and not the rows means that the number of neurons is reduced, but all the training information are taken into consideration. This is the key concept in trying to maintain the quality, while the equation set is simplified.

This paper only describes this possible generalization, but it is only used as the framework for weighting, therefore it does not deal with other possible usages or specific questions not related to that method. These unrelated questions concern the selection of $\mathbf{c}_i$ centers, K_i kernels, and the advantages of their customisation. Even the selection of the kernel centers is a complex problem which has been studied much, mostly in respect of RBF. Briefly it can be stated, that the $\mathbf{c}_i$ centers

- may be distributed uniformly for the simplest solution (e.g. for Fixed LS–SVM),
- may be selected from the training sample set (just as in the original SVM),
- may be selected by utilizing a clustering method,

etc. A selection method along with a more detailed description on matrix reduction techniques is presented in ref. [28].

B. Algorithmic complexity of the methods

This section deals with the algorithmic issues of the described solutions. Traditional LS–SVM training requires the solution of a linear equation set. In the case of N training vectors this can be solved using the LU decomposition in $\frac{1}{3}N^3+N^2$ steps, each with one multiplication and one addition [40]. If the training set has N training samples, then the equation set consists of $N+1$ equations and the size of the matrix to be manipulated is $(N+1)\times(N+1)$. To keep the formulas simple we will consider a matrix of size $N\times N$ (since $N\gg 1$, the effect of the one additional row is neglected). The reduced row echelon form of a matrix can be reached in N^2 steps. Let's assume that the reduction leads to M 'support' vectors. In the case of partial reduction, the calculation of $\mathbf{A}^T\mathbf{A}$ (defined in *Eq.* (25) requires M^2N steps. Solving this new equation set costs $\frac{1}{3}M^3+M^2$ steps. So the total cost of the proposed algorithm adds up to $N^2+M^2N+\frac{1}{3}M^3+M^2$.

If $M\ll N$, this means a smaller complexity compared to that of traditional LS–SVM. It is important to mention that even if there is no algorithmic gain, or it is rather small, this calculation provides a sparse solution with a good performance. If the traditional LS–SVM is pruned for sparseness, then an equation set – slowly decreasing in size – is solved in each iteration, which multiplies the complexity, whilst the errors may grow.

6. The Weighted Least Squares LS–SVM Method

The above described least squares LS–SVM, aims at achieving the same goals – such as sparseness, a fixed version or noise reduction – as the standard LS–SVM. The weighted version of the standard method has been introduced to deal with noisy datasets, especially datasets containing outliers. This section shows a possible weighting method for the least squares LS–SVM.

The problem is that the considered input–output relations, namely the rows of the $\mathbf{Ax} = \mathbf{b}$ equation set should be weighted according to their significance, and the solution of the equation set must reflect the effects of this. Since the relations represented in the training samples are formulated as rows in the equation set, we have to weight the importance of the rows so that when minimizing for the mean square errors, the effect of the rows (equations) reflects their importance in the final summation. This means that the errors for the more exact equations (samples) have a larger effect in the linear least squares problem than the noisier ones.

The $\mathbf{A}^T\mathbf{A}$ matrix is the sum of the diadic products of the row vectors. By weighting this sum, the effect of each row – training sample – can be controlled. The least squares equation of the weighted equation set becomes:

$$(v_1\underline{a}_1^T\underline{a}_1 + v_2\underline{a}_2^T\underline{a}_2 + \cdots + v_i\underline{a}_i^T\underline{a}_i + \cdots + v_N\underline{a}_N^T\underline{a}_N)\begin{bmatrix} b \\ \boldsymbol{\alpha} \end{bmatrix} = \mathbf{A}^T[v_1b_1, \ldots, v_ib_i, \ldots, v_Nb_N]^T, \tag{26}$$

where the v_i-s are the weights and the $\underline{a}_i$-s are the row vectors of $\mathbf{A}$. The solution of this weighted equation set reflects the accuracy of the samples.

There are two things to determine:

- the relative importance of the points,
- a weighting strategy to calculate the actual weight factors for each points.

The relative importance means that the exact samples should have larger, while the noisy ones should have smaller weights. The weights corresponding to each row can be determined by some *a priori* knowledge (e.g. about the amount of noise for each sample) or iteratively like in the original unreduced case. In this case the weights are calculated according to the results of a previous regression, by considering how close the point is to this probably acceptable approximation.

To determine the v_i multipliers, a weighting strategy must be chosen. This strategy specifies how errors should be penalized thorough weighting. It can be as simple as a linear function of the errors, but some more sophisticated strategies (from the field of statistics) can be found in [19].

By using this method both pruning (since partial reduction is used) and weighting are achieved.

7. Experiments

This section splits the experiments into two parts. First we demonstrate the performance of partial reduction applied together with the proposed reduced row 'support vector' selection method and we compare the results with the standard LS–SVM. After this the effect of the above described weighting is demonstrated.

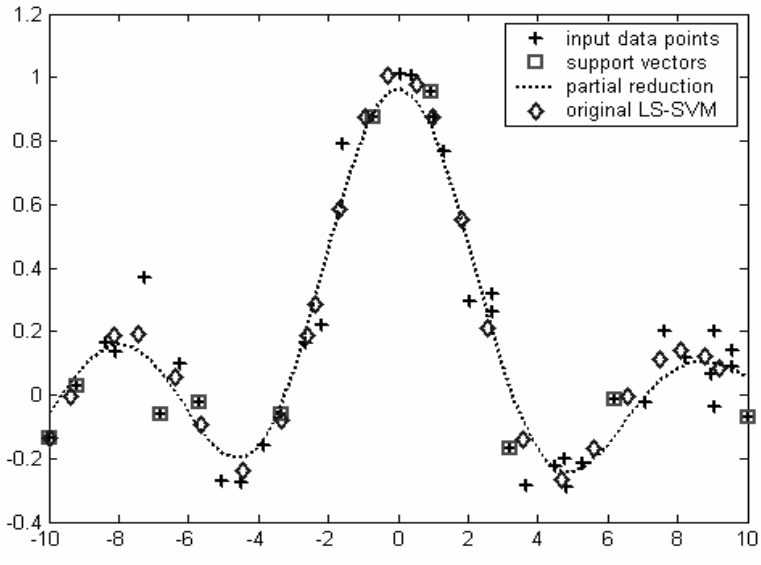

Fig. 6. The results of a least squares LS–SVM, and the original LS–SVM for a sinc (x) regression. (The training set contains 40 noisy data points).

Our results illustrated in *Fig. 6* prove that by using an overdetermined equation set, the network size can be effectively reduced without impairing from the quality of the results. It can be seen that the partial reduction gives almost the same result as the original LS–SVM, while in this case the network contains much less – 10 instead of 40 – neurons.

In *Fig. 7*, the function sinc (x) is approximated by the use of the same size support vector set for partial and full reduction. In the original LS–SVM pruning method (full reduction), the results worsen, due to the information loss, because some of the samples are left out. The fully reduced solution is only influenced by the 'support vectors', which results in a distorted estimate, as it can easily be seen in the figure. The partial reduction, however, maintains the quality.

The next figure shows the results of weighting for a sinc (x) regression. The training set contains 55 data points, about 30% of it is burdened with Gaussian noise and five points are made outliers!

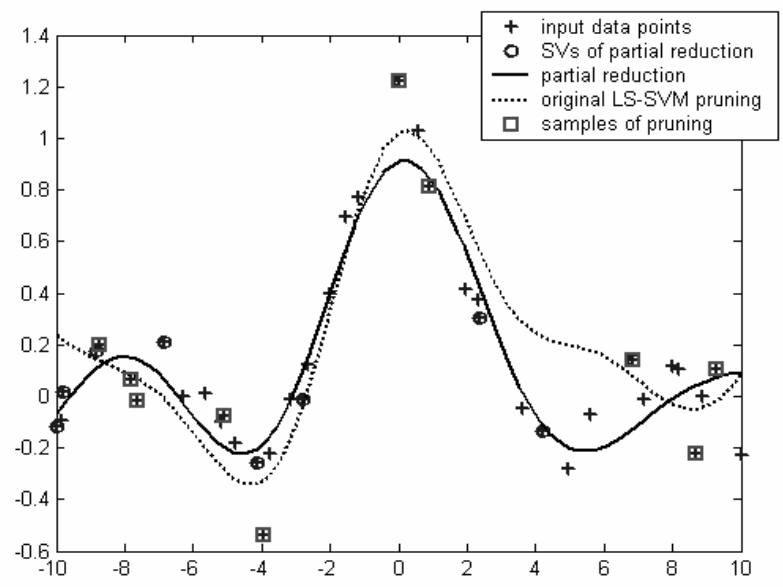

Fig. 7. The results of a full and partial reduction for a sinc (x) regression. (The training set has the same properties as earlier).

It can be seen that the outliers mislead the solution, but the proposed weighting effectively reduces the effect of these points, resulting in a much better solution. In the presented experiment the weighting is done according to an a priori knowledge about noise. The weighting strategy is extremely simple, since it punishes noise in a linear fashion.

The following table summarizes the results for several test runs on different training sets randomly generated with the same properties described above.

Table 2. Mean squares errors for several test runs with and without weighting

#	Least squares LS–SVM	Weighted least squares LS–SVM
1	0.0383	0.0117
2	0.0328	0.0240
3	0.0443	0.0316
4	0.0227	0.0043
5	0.0138	0.0216
6	0.0250	0.0318
Σ	0.1769	0.1250

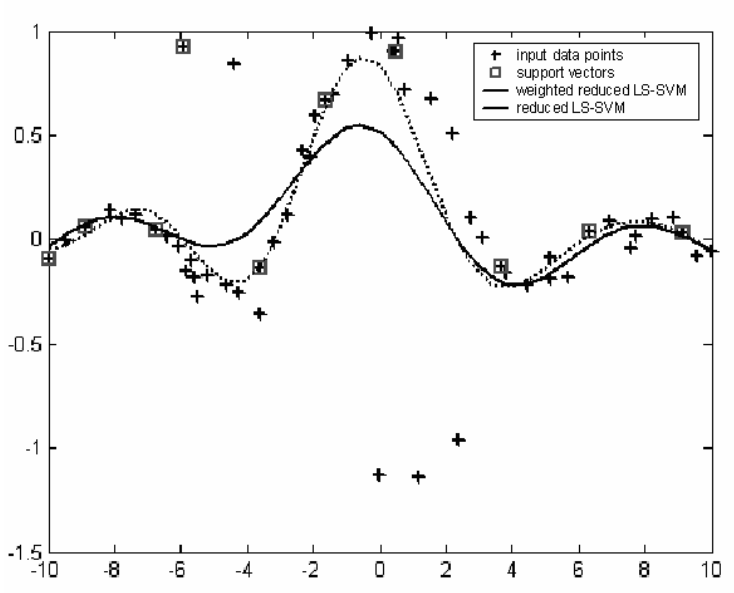

Fig. 8. The results of a least squares LS–SVM sinc (x) regression without (full line) and with weighting (dashed line), the chosen support vectors are marked.

It must be stated that this method works only well, if a large number of the points are not noisy. The method has been successfully tested for a number of problems but, of course, further experiments are needed. For example, a more sophisticated weighting strategy than the presently used linear one could be implemented.

8. Conclusion

The main idea is a generalized least squares modification of the LS–SVM, which provides us with several advantages like simplified formulation, sparse solution etc. This paper presents the weighted version of this, which is an important result, since it shows that this method could be extended just like the standard version. The described weighting technique effectively reduces the effect of outliers or, more generally, heavy tail noise distributions, resulting in a more exact solution. The weighting strategy used in the experiments is an extremely simple one, some improvements may be achieved after further enhancements. Other such extensions like Fixed LS–LS–SVM are to be published soon.

The proposed modifications provide a more effective way to achieve a sparse (or fixed) solution, than standard pruning, while other improvements are still possible.

References

[1] HORNIK, K. – STINCHCOMBE, M. – WHITE, H., Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, **2** (1989) pp. 359–366.

[2] CYBENKO, G., Approximation by Superpositions of a Sigmoidal Function, *Mathematics of Control, Signals and Systems*, **3** (1989), pp. 303–314.

[3] PARK, J. – SANDBERG, I. W., Approximation and Radial-Basis-Function Networks, *Neural Computation*, **5** No. 2. (1993), pp. 305–316.

[4] HAYKIN, S., *Neural Networks, A comprehensive Foundation*, second edition, Prentice Hall, N. J. 1999.

[5] VAPNIK, P., *The Nature of Statistical Learning Theory*, Springer-Verlag, New-York, 1995.

[6] SCHÖLKOPF, B. – SMOLA, A. J., *Learning with Kernels. Support Vector Machines, Regularisation, Optimisation and Beyond*, MIT Press. 2002.

[7] DRUCKER, H. – BURGES, C. J. C. – KAUFMAN, L. – SMOLA, A. – VAPNIK, V., Support Vector Regression Machines, In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pp. 155–161, Cambridge, MA, 1997, MIT Press.

[8] BURGES, C. J. C., A Tutorial on Support Vector Machines for Pattern Recognition, *Knowledge Discovery and Data Mining*, **2** No. 2, 1998.

[9] GUNN, S., Support Vector Machines for Classification and Regression, *ISIS Technical Report*, **14** May 1998.

[10] ANCONA, N., Properties of Support Vector Machines for Regression, *Technical Report 01-99*, Istituto Elaborazione Segnali ed Immagini, Bari, Italy, 1999.

[11] OSUNA, E. – FREUND, R. – GIROSI, F., Support Vector Machines: Training and Applications, *Technical Report AIM-1602*, MIT A.I. Lab., 1996.

[12] SUYKENS, J. A. K., Nonlinear Modelling and Support Vector Machines, *IEEE Instrumentation and Measurement Technology Conference*, Budapest, Hungary, May 21–23, 2001.

[13] SUYKENS, J. A. K. – VANDEWALLE, J., Least Squares Support Vector Machine Classifiers, *Neural Processing Letters*, **9** No. 3 (1999), pp. 293–300.

[14] SUYKENS, J. A. K. – VANDEWALLE, J., Multiclass Least Squares Support Vector Machines, *In IJCNN'99 International Joint Conference on Neural Networks*, Washington, DC, 1999.

[15] SUYKENS, J. A. K. – VAN DOOREN, P. – DE MOOR, B. – VANDEWALLE, J., Least Squares Support Vector Machine Classifiers: a Large cale Algorithm, In *European Conference on Circuit Theory and Design, ECCTD'99*, (1999), pp. 839–842.

[16] SUYKENS, J. A. K. – GESTEL, V. T. – DE BRABANTER, J. – DE MOOR, B. – VANDEWALLE, J., *Least Squares Support Vector Machines*, World Scientific, 2002.

[17] SUYKENS, J. A. K. – LUKAS, L. – VANDEWALLE, J., Sparse Approximation Using Least Squares Support Vector Machines, In *IEEE International Symposium on Circuits and Systems ISCAS'2000*, 2000.

[18] SUYKENS, J. A. K. – LUKAS, L. – VANDEWALLE, J., Sparse Least Squares Support Vector Machine Classifiers, In *ESANN'2000 European Symposium on Artificial Neural Networks*, (2000), pp. 37–42.

[19] BURGES, C. J. C. – SCHÖLKOPF, B., Improving the Accuracy and Speed of Support Vector Learning Machines, In M. Mozer, M. Jordan and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, Cambridge, MA, 1997. MIT Press, pp. 375–381.

[20] OSUNA, E. – FREUND, R. – GIROSI, F., An Improved Training Algorithm for Support Vector Machines, In J. Principe, L. Gile, N. Morgan, and E. Wilson, editors, *Neural Networks for Signal Processing VII – Proceedings of the 1997 IEEE Workshop*, New York, 1997, IEEE, pp. 276–285.

[21] PLATT, J. C., Sequential Minimal Optimization: Fast Algorithm for Training Support Vector Machines, *Microsoft Research Technical Report MSR–TR–98–14*, April 21, 1998.

[22] LASKOV, P., An Improved Decomposition Algorithm for Regression Support Vector Machines, In S.A. Solla, T.K. Leen and K.-R. Müller, editors, *Advances in Neural Information Processing Systems 12*, MIT Press, (2000), pp. 484–490.

[23] KEERTHI, S. S. – SHEVADE, S. K. – BHATTACHARYYA, C. – MURTHY, K. R. K., Improvements to Platt's SMO Algorithm for SVM Classifier Design, Technical report, Dept of CSA, IISc, Bangalore, India, 1999.
[24] JOACHIMS, TH., *Making Large–Scale SVM Learning Practical, Advances in Kernel Methods–Support Vector Learning*, MIT Press, Cambridge, USA, 1998.
[25] MOSER, M. – JORDAN, M. – PETSCHE, T., eds., Improving the Accuracy and Speed Support Vector Machines, *Neural Information Processing Systems*, **9** (1997), MIT Press, Cambridge, MA.
[26] VALYON, J. – HORVÁTH, G., Generalized Least-Squares Support Vector Machine, *13th IFAC Symposium on System Identification SYSID*, (2003), pp. 828–832.
[27] VALYON, J. – HORVÁTH, G., Reducing the Complexity and Network Size of LS–SVM Solutions, Submitted to: *IEEE Transactions on Neural Networks*, 2000.
[28] FLETCHER, R., Practical Methods of Optimization, Wiley-Interscience, New-York, NY, USA, 1987.
[29] CHAPELLE, O. – VAPNIK, V. – BENGIO, Y., Model Selection for Small Sample Regression, AT&T Research Labs, 2001.
[30] CRISTIANINI, N. – CAMPBELL, C. – SHAWE-TAYLOR, J., Dynamically Adapting Kernels in Support Vector Machines, NeuroCOLT Technical Report NC-TR-98-017, Royal Holloway College, University of London, UK, 1998.
[31] CHAPELLE, O. – VAPNIK, V., Model Selection for Support Vector Machines, In Sara A. Solla, Todd K. Leen, and Klaus-Robert Müller, editors, *Advances in Neural Information Processing Systems 12*, Cambridge, Mass: MIT Press, 2000.
[32] YUH–JYE LEE – MANGASARIAN, O. L., RSVM: Reduced Support Vector Machines, *Proceedings of the First SIAM International Conference on Data Mining*, Chicago, April 5–7. 2001.
[33] MIKA, S. – RÄTSCH, G. – SCHÖLKOPF, B. – SMOLA, A. – WESTON, J. – MÜLLER, K.-R., Invariant feature extraction and classification in kernel spaces. In *Advances in Neural Information Processing Systems 12*, Cambridge, MA, 1999. MIT Press.
[34] MIKA, S. – SCHÖLKOPF, B. – SMOLA, A. – MÜLLER, K.-R. – SCHOLZ, M. – RÄTSCH, G., Kernel PCA and De-Noising in Feature Spaces, In M. S. Kearns, S. A. Solla, and D. A. Cohn, editors, *Advances in Neural Information Processing Systems 11*, Cambridge, MA, 1999. MIT Press, pp. 536–542.
[35] SCHÖLKOPF, B. – MIKA, S. – BURGES, C. J. C. – KNIRSCH, P. – MÜLLER, K.-R. – RÄTSCH, G. – SMOLA, A., Input Space vs. Feature Space in Kernel-Based Methods, *IEEE Transactions on Neural Networks*, **10** No. 5 (1999), pp. 1000–1017.
[36] SCHÖLKOPF, B. – SMOLA, A. – MÜLLER, K. R. – BURGES, C. – VAPNIK, V., Learning and Feature Extraction with Support Vector Methods, In T. Downs, M. Frean, and M. Gallagher, editors, *Proc. of the Ninth Australian Conf. on Neural Networks*, Brisbane, Australia, 1998, University of Queensland.
[37] BAUDAT, G. – ANOUAR, F., Kernel-Based Methods and Function Approximation, In *International Joint Conference on Neural Networks*, Washington, DC July 15–19, 2001, pp. 1244–1249.
[38] Matlab 5.3, Help, MathWorks Inc.
[39] *Numerical Recipes*, Cambridge University Press, Books On-Line, Web: www.nr.com.
[40] GOLUB, H. – VAN LOAN, CH. F., *Matrix Computations*, Gene Johns Hopkins University Press, 1989.