Sparse Online Greedy Support Vector Regression

Yaakov Engel[1], Shie Mannor[2], and Ron Meir[2] *

[1] Center for Neural Computation, Hebrew University
Jerusalem 91904, Israel
yaki@alice.nc.huji.ac.il
[2] Dept. of Electrical Engineering, Technion Institute of Technology
Haifa 32000, Israel
{shie@tx,rmeir@ee}.technion.ac.il

Abstract. We present a novel algorithm for sparse online greedy kernel-based nonlinear regression. This algorithm improves current approaches to kernel-based regression in two aspects. First, it operates online – at each time step it observes a single new input sample, performs an update and discards it. Second, the solution maintained is extremely sparse. This is achieved by an explicit greedy sparsification process that admits into the kernel representation a new input sample only if its feature space image is linearly independent of the images of previously admitted samples. We show that the algorithm implements a form of gradient ascent and demonstrate its scaling and noise tolerance properties on three benchmark regression problems.

1 Introduction

Kernel machines have become by now a standard tool in the arsenal of Machine Learning practitioners. Starting from the seventies a considerable amount of research was devoted to kernel machines and in recent years focused on Support Vector Machines (SVMs) [13]. A basic idea behind kernel machines is that under certain conditions the kernel function can be interpreted as an inner product in a high dimensional Hilbert space (feature space). This idea, commonly known as the "kernel trick", has been used extensively in generating non-linear versions of conventional linear supervised and unsupervised learning algorithms, most notably in classification and regression; see [5,8,11] for recent reviews. SVMs have the noteworthy advantage of frequently yielding sparse solutions. By sparseness we mean that the final classifier or regressor can be written as a combination of a relatively small number of input vectors, called the support vectors (SVs). Besides the practical advantage of having the final classifier or regressor depend on a small number of SVs, there are also generalization bounds that depend on the sparseness of the resulting classifier or regressor (e.g., [5]). However, the

* The research of R. M. was supported by the fund for promotion of research at the Technion and by the Ollendorff center.

T. Elomaa et al. (Eds.): ECML, LNAI 2430, pp. 84–96, 2002.

solutions provided by SVMs have been empirically shown to be not maximally sparse [1,6]. Additionally, support vector regression involves using twice as many variables as the number of samples, rendering the computational problem more difficult.

The solution of SVM problems usually involves a non-trivial quadratic optimization problem. There have been several attempts to find efficient algorithms for this problem (e.g., [9,4]), most of which are based on the special form of the dual quadratic optimization problem. These methods display a super-linear dependence of the computation time on the number of samples and require repeated access to the training samples, making them suitable only for batch learning.

Achieving sparseness by using linear dependence was suggested in [6]. The idea there is to solve the SVM problem using standard methods and then simplify the solution by eliminating linear dependencies in feature space. This simplification procedure effectively reduces the number of SVs while keeping the final classifier/regressor unchanged.

An important requirement of online algorithms is that their per-time-step complexity should be bounded by a constant independent of t (t being the time-step index), for it is assumed that samples arrive at a constant rate. Since the complexity of SV learning is super-linear in the number of samples [4], performing aggressive sparsification concurrently with learning is essential.

In this work we take advantage of an efficient greedy sparsification method in order to obtain an approximate, provably convergent, online SVR algorithm that we call SOG-SVR (sparse online greedy SVR).

The remainder of the paper is organized as follows: In Section 2 we briefly overview support vector regression along with a sequential implementation which inspired our algorithm. In Section 3 we introduce a method for detecting linear dependencies in feature space and for sparsifying the solution based on these dependencies. Section 4 presents the SOG-SVR algorithm along with a convergence proof. In Section 5 we apply SOG-SVR to two benchmark problems, and we conclude with Section 6.

2 Support Vector Regression

Consider a training sample $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$, $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, where $y_i = g(\mathbf{x}) + \zeta_i$, for some unknown function g and noise variable ζ_i. The objective is to reconstruct a good approximation to $g(\cdot)$ from the finite data set S. The derivation of the SVR equations can be found in [11] and will not be repeated here, but for completeness we recall the main results. Let $\boldsymbol{\phi}$ be a nonlinear mapping from input space to some high-dimensional feature space. For the linear regressor (in feature space) defined by $f(\cdot) \stackrel{def}{=} \langle \mathbf{w}, \boldsymbol{\phi}(\cdot) \rangle + b$, we wish to minimize

$$\mathcal{R}(\boldsymbol{\xi}, \boldsymbol{\xi}^*, \mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{\ell} (\xi_i + \xi_i^*), \tag{2.1}$$

subject to (for $\forall i \in \{1, \ldots, \ell\}$)

$$y_i - f(\mathbf{x}_i) \leq \varepsilon + \xi_i^*, \quad f(\mathbf{x}_i) - y_i \leq \varepsilon + \xi_i, \quad \xi_i, \xi_i^* \geq 0, \tag{2.2}$$

where ε defines the width of the error-insensitive zone of the cost function and ξ_i^* and ξ_i are slack variables measuring the deviation of $y_i - f(\mathbf{x}_i)$ from the boundaries of the error-insensitive zone. This primal problem is transformed to its dual quadratic optimization problem. Maximize

$$\mathcal{L}(\boldsymbol{\alpha}, \boldsymbol{\alpha}^*) = -\frac{1}{2}(\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T K(\boldsymbol{\alpha}^* - \boldsymbol{\alpha}) - \varepsilon(\boldsymbol{\alpha}^* + \boldsymbol{\alpha})^T \mathbf{e} + (\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T \mathbf{y} \tag{2.3}$$

subject to

$$(\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T \mathbf{e} = 0 \tag{2.4}$$

$$0 \quad \leq \quad \boldsymbol{\alpha}^*, \boldsymbol{\alpha} \leq C\mathbf{e} \tag{2.5}$$

where $k(\mathbf{x}, \mathbf{x}') = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}') \rangle$, $[K]_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j)$, $\boldsymbol{\alpha}^{(*)} = (\alpha_1^{(*)}, \ldots, \alpha_\ell^{(*)})^T$ and $\mathbf{e} = (1, \ldots, 1)^T$. In order to simplify notation, here and in the sequel we use the standard notation $\boldsymbol{\alpha}^{(*)}$ to refer to either $\boldsymbol{\alpha}$ or $\boldsymbol{\alpha}^*$.

The Representer Theorem [15] assures us that the solution to this optimization problem may be expressed solely in terms of the kernel function over the training set:

$$f(\cdot) = \sum_{i=1}^{\ell} \beta_i k(\mathbf{x}_i, \cdot) + b \ . \tag{2.6}$$

In SVR, once the dual problem (2.3) is solved for $\boldsymbol{\alpha}^*$ and $\boldsymbol{\alpha}$, the optimal regressor may be written as $f(\cdot) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) k(\mathbf{x}_i, \cdot) + b$.

By redefining the feature vectors and the weight vector $\bar{\boldsymbol{\phi}} = (\boldsymbol{\phi}^T, \lambda)^T$ and $\bar{\mathbf{w}} = (\mathbf{w}^T, b/\lambda)^T$ we can "homogenize" the regressor:

$$f(\mathbf{x}) = \sum_{i=1}^{\ell} (\alpha_i^* - \alpha_i) \bar{k}(\mathbf{x}_i, \mathbf{x}), \tag{2.7}$$

where $\bar{k}(\mathbf{x}', \mathbf{x}) = k(\mathbf{x}', \mathbf{x}) + \lambda^2$. The transformation to the homogeneous form is equivalent to adding a usually small, positive constant term to the kernel function. Note however, that the regularization term $\|\bar{\mathbf{w}}\|^2$ in the primal Lagrangian (2.1) now includes the free term b. Homogenizing the regressor not only simplifies the analysis but also allows us to get rid of the first constraint (2.4) in the dual problem. This is significant because the remaining constraints can be enforced locally, in the sense that they do not mix different components of $\boldsymbol{\alpha}^{(*)}$. From this point on we assume that this transformation has been performed and we drop the "bar" notation.

In [14] two sequential algorithms for learning support vector classification and regression, respectively, were introduced. While conventional SV learning algorithms operate in batch mode, requiring all training samples to be given in

advance; in these two algorithms a single training sample is observed at each time step and the update is based solely on that sample, keeping the time complexity per time step $O(1)$. The regression algorithm they propose (SVRseq) solves the kernel regression problem sequentially, and is presented in pseudo-code form in Table 1. The notation $\lfloor \mathbf{u} \rceil$ is shorthand for the truncation function defined by $\lfloor u \rceil = \max(0, \min(C, u))$; i.e., values are truncated so as to remain within the bounds specified by the constraints (2.5). η is a learning rate parameter.

Table 1. The SVRseq Algorithm

1. **Parameters:** η, ε, C.
2. **Initialize:** $\boldsymbol{\alpha}^* = 0$, $\boldsymbol{\alpha} = 0$.
3. For $i = 1, \ldots, \ell$
   $d_i = y_i - f(\mathbf{x}_i)$
   $\Delta\alpha_i^* = \eta(d_i - \varepsilon)$
   $\Delta\alpha_i = -\eta(d_i + \varepsilon)$
   $\alpha_i^* = \lfloor \alpha_i^* + \Delta\alpha_i^* \rceil$
   $\alpha_i = \lfloor \alpha_i + \Delta\alpha_i \rceil$
4. If training has converged stop, else repeat step 3.

3 Sparsity by Approximate Linear Dependence

When dealing with very large data sets we may not be able to afford, memory-wise, to maintain the parameter vectors $\boldsymbol{\alpha}^*$ and $\boldsymbol{\alpha}$ for the entire training sample. A large number of SVs would also slow the system down at its testing/operation phase. Moreover, sequential SV algorithms rely on multiple passes over the training set, since each update affects only one component of $\boldsymbol{\alpha}^*$ and $\boldsymbol{\alpha}$ and makes only a small uphill step on the dual Lagrangian. In online learning no training sample is likely to be seen more than once, so, an alternative approach is required. Hence, we must devise a way to select which input samples should be remembered in the kernel representation, and to update their corresponding coefficients.

In [6] a sparsification step is performed *after* a conventional SV solution is obtained. This is done by eliminating SVs that can be represented as linear combinations of other SVs in feature space, and appropriately updating the coefficients of the remaining ones in order to obtain exactly the same classifier/regressor[1].

Our aim is to incorporate a similar selection mechanism into an online learning scheme based on the sequential SV learning algorithm of [14]. The general idea is as follows. Let us assume that, at time step t, after having observed $t-1$ training samples we have collected a dictionary of m linearly independent basis vectors $\{\boldsymbol{\phi}(\tilde{\mathbf{x}}_j)\}_{j=1}^m$ ($\tilde{\mathbf{x}}_j$ represent those elements in the training set which were retained up to the t-th step). Now we are presented with a new sample $\mathbf{x}_t$. We

[1] Note however, that the resulting solution does not necessarily conform to the constraints of the original problem.

test whether $\boldsymbol{\phi}(\mathbf{x}_t)$ is linearly dependent on the dictionary vectors. If not, we add it to the dictionary and increment m by 1. Due to the online mode of operation we do not wish to revisit and revise past decisions, and for this reason the addition of vectors to the dictionary must be done in a greedy manner.

The requirement for exact linear dependence may lead us to numerical instabilities. Instead, for training sample $\mathbf{x}_t$, we will be content with finding coefficients $\{a_{t,j}\}_{j=1}^{m}$ with at least one non-zero element satisfying the *approximate linear dependence* condition

$$\delta_t = \left\| \sum_{j=1}^{m} a_{t,j} \boldsymbol{\phi}(\tilde{\mathbf{x}}_j) - \boldsymbol{\phi}(\mathbf{x}_t) \right\|^2 \leq \nu \ . \tag{3.8}$$

In other words, $\boldsymbol{\phi}(\mathbf{x}_t)$ can be approximated to within a squared error of ν by some linear combination of current dictionary members. By minimizing the left hand side of (3.8) we can simultaneously check whether this condition is satisfied and obtain the coefficient vector for input sample $\mathbf{x}_t$, $\mathbf{a}_t = (a_{t,1}, \ldots, a_{t,m})^T$ that best satisfies it. For reasons of numerical stability, we may sometimes prefer to sacrifice the optimality of $\mathbf{a}_t$ in return for a reduction in the size of its components. In this case we can add an ℓ_2 norm regularization term of the form $\gamma\|\mathbf{a}\|^2$ to the minimization problem defined in (3.8), ending up with the optimization problem

$$\min_{\mathbf{a}\in\mathbb{R}^m} \left\{ \mathbf{a}^T(\tilde{K} + \gamma I)\mathbf{a} - 2\mathbf{a}^T\tilde{\mathbf{k}}_t + k_{tt} \right\} \tag{3.9}$$

where $[\tilde{K}]_{i,j} = k(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$, $(\tilde{\mathbf{k}}_t)_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_t)$, $k_{tt} = k(\mathbf{x}_t, \mathbf{x}_t)$, with $i, j = 1, \ldots, m$.

Solving (3.9) yields $\mathbf{a}_t = (\tilde{K} + \gamma I)^{-1}\tilde{\mathbf{k}}_t$, and the condition for approximate linear dependence becomes

$$\delta_t = k_{tt} - (\tilde{\mathbf{k}}_t + \gamma\mathbf{a}_t)^T\mathbf{a}_t \leq \nu \ . \tag{3.10}$$

By defining $[A]_{i,j} = a_{i,j}$ and taking the linear dependence (3.8) to be exact (i.e., $\nu = 0$), we may express the full $\ell \times \ell$ training set Gram matrix K as $A\tilde{K}A^T$. For ν sufficiently small, $A\tilde{K}A^T$ is a good approximation of K, and from this point on we will assume that indeed $K = A\tilde{K}A^T$.

4 Sparse Online Greedy SVR

In this section we show how the idea of sparsification by linear dependence can be utilized in an online SVR algorithm.

4.1 A Reduced Problem

As already hinted above, we would like to end up with a SVR algorithm with time and memory requirements that are independent of the number of training samples ℓ that, for online algorithms, equals the time index t. Our aim is to

obtain an algorithm with memory and time bounds, per time step, that are dependent only on the intrinsic dimensionality m of the data in feature space. In order to do that we define a new set of $2m$ "reduced" variables

$$\tilde{\boldsymbol{\alpha}}^{(*)} = A^T \boldsymbol{\alpha}^{(*)} \ , \tag{4.11}$$

where it should be observed that $\tilde{\boldsymbol{\alpha}}^{(*)} \in \mathbb{R}^m$, while $\boldsymbol{\alpha}^{(*)} \in \mathbb{R}^\ell$, and typically $m \ll \ell$.

For clarity let us first consider the case where ℓ and m are fixed, i.e., we have a predefined set of dictionary vectors on which all other $\ell - m$ training-set feature vectors are linearly dependent. Let $\mathbf{y} = (y_1, \ldots, y_\ell)^T$, $\mathbf{f} = A\tilde{K}(\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}})$. In this case the following rule can be used repeatedly to update the reduced variables:

$$\begin{aligned}
\Delta\tilde{\boldsymbol{\alpha}}^* &= \eta^* A^T(\mathbf{y} - \mathbf{f} - \varepsilon\mathbf{e}) \\
\Delta\tilde{\boldsymbol{\alpha}} &= -\eta A^T(\mathbf{y} - \mathbf{f} + \varepsilon\mathbf{e}) \\
\tilde{\boldsymbol{\alpha}}^* &= \tilde{\boldsymbol{\alpha}}^* + \Delta\tilde{\boldsymbol{\alpha}}^* \\
\tilde{\boldsymbol{\alpha}} &= \tilde{\boldsymbol{\alpha}} + \Delta\tilde{\boldsymbol{\alpha}}
\end{aligned} \tag{4.12}$$

where η^*, η are small positive learning rate parameters. There are several points worth stressing here. First, while the update in SVRseq is scalar in nature, with only a single component of $\boldsymbol{\alpha}^*$ and $\boldsymbol{\alpha}$ updated in each step; here, in each update, all of the components are affected. Second, in the online case A becomes A_t which is a growing $t \times m$ matrix to which the row vector $\mathbf{a}_t^T$ is appended at each time step t. $\mathbf{y}_t$, $\mathbf{f}_t$ and $\mathbf{e}$ are also increasingly large t-dimensional vectors. Luckily, we need only maintain and update their m-dimensional images under the transformation A_t^T: $A_t^T\mathbf{y}_t$, $A_t^T\mathbf{f}_t = A_t^T A_t \tilde{K}(\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}})$ and $A_t^T\mathbf{e}$. Third, the evaluation of the regressor at point $\mathbf{x}_t$ can be performed using the reduced variables:

$$\begin{aligned}
f(\mathbf{x}_t) &= \sum_{i=1}^{t} (\alpha_i^* - \alpha_i) k(\mathbf{x}_i, \mathbf{x}_t) \\
&= \sum_{i=1}^{t} (\alpha_i^* - \alpha_i) \sum_{j=1}^{m} a_{i,j} k(\tilde{\mathbf{x}}_j, \mathbf{x}_t) \\
&\overset{\nu \to 0}{=} (\boldsymbol{\alpha}^* - \boldsymbol{\alpha})^T A_t \tilde{\mathbf{k}}_t = (\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}})^T \tilde{\mathbf{k}}_t
\end{aligned} \tag{4.13}$$

where the third equality above is exact only when $\nu = 0$. Fourth, due to the quadratic nature of the dual Lagrangian, optimal values for the learning rates $\eta^{(*)}$ can be analytically derived. We defer discussion of this issue to a longer version of the paper. Finally, note that here we do not truncate the updates $\Delta\tilde{\boldsymbol{\alpha}}^{(*)}$, namely the $\tilde{\boldsymbol{\alpha}}^{(*)}$ vectors are not bounded within some box, nor do they necessarily correspond to any $\boldsymbol{\alpha}^{(*)}$ obeying the constraints (2.5). We will discuss the implications of this omission later.

If, at time t, a new training sample is encountered for which the approximate linear dependence condition (3.8) is not satisfied, we append $\boldsymbol{\phi}(\mathbf{x}_t)$ to the dictionary, increasing m by 1. This has the following consequences:

- A_t is first appended with an additional column of t-1 zeros and then with the row vector $\mathbf{a}_t^T$. Consequently, A_t is lower triangular.
- $\tilde{K}$ is also appended with a column and a row, first the column $\tilde{\mathbf{k}}_t$ and then the row $(\tilde{\mathbf{k}}_t^T, k_{tt})$. Note that, contrary to K, $\tilde{K}$ has always full rank and is therefore invertible.
- $\tilde{\boldsymbol{\alpha}}^*$ and $\tilde{\boldsymbol{\alpha}}$ are each appended with an additional component, initialized at 0.

A detailed pseudo-code account of the algorithm is given in Table 2. Concerning notation, we use "," to denote horizontal concatenation, similarly, we use ";" to denote vertical concatenation. In order to make the distinction between matrices and vectors clear, we use $[\cdot]$ for a matrix and $(\cdot)$ for a vector.

Table 2. The SOG-SVR Algorithm

- **Parameters:** ε, C, ν, γ.

- **Initialize:** $\boldsymbol{\alpha}_1^* = \big(\max(0, y_1/k_{1,1})\big)$, $\boldsymbol{\alpha}_1 = \big(\min(0, -y_1/k_{1,1})\big)$, $\tilde{K}_1 = [k_{1,1}]$, $A_1^T A_1 = [1]$, $A_1^T \mathbf{y}_1 = (y_1)$, $A_1^T \mathbf{e} = (1)$, $\mathbf{0} = (0)$, $I = [1]$, $\mathbf{e} = (1)$, $m = 1$.

- **for** $t = 2, 3, \ldots$
  1. **Get new sample:** $(\mathbf{x}_t, y_t)$
  2. **Compute** $\tilde{\mathbf{k}}_t$**:** $(\tilde{\mathbf{k}}_t)_i = k(\tilde{\mathbf{x}}_i, \mathbf{x}_t)$
  3. **Approximate linear dependence test:**
     $\mathbf{a}_t = (\tilde{K}_m + \gamma I)^{-1} \tilde{\mathbf{k}}_t$
     $\delta_t = k_{tt} - (\tilde{\mathbf{k}}_t + \gamma \mathbf{a}_t)^T \mathbf{a}_t$
     if $\delta_t > \nu$ % add $\mathbf{x}_t$ to dictionary
       $\tilde{K}_{m+1} = \begin{bmatrix} \tilde{K}_m, \tilde{\mathbf{k}}_t; \tilde{\mathbf{k}}_t^T, k_{tt} \end{bmatrix}$
       $\mathbf{a}_t = (\mathbf{0}, \ldots; 1)$
       $A_t^T \mathbf{e} = (A_{t-1}^T \mathbf{e}; 1)$
       $A_t^T \mathbf{y}_t = (A_{t-1}^T \mathbf{y}_{t-1}; y_t)$
       $A_t^T A_t = [A_{t-1}^T A_{t-1}, \mathbf{0}; \mathbf{0}^T, 1]$
       $\boldsymbol{\alpha}_{t-1}^* = (\boldsymbol{\alpha}_{t-1}^*; 0)$
       $\boldsymbol{\alpha}_{t-1} = (\boldsymbol{\alpha}_{t-1}; 0)$
       $I = [I, \mathbf{0}; \mathbf{0}^T, 1]$
       $\mathbf{0} = (\mathbf{0}; 0)$
       $m = m + 1$
     else % dictionary remains unchanged
       $A_t^T \mathbf{e} = A_{t-1}^T \mathbf{e} + \mathbf{a}_t$
       $A_t^T \mathbf{y}_t = A_{t-1}^T \mathbf{y}_{t-1} + \mathbf{a}_t y_t$
       $A_t^T A_t = A_{t-1}^T A_{t-1} + \mathbf{a}_t \mathbf{a}_t^T$
  4. **Update** $\tilde{\boldsymbol{\alpha}}^*$ **and** $\tilde{\boldsymbol{\alpha}}$**:**
     $A_t^T \mathbf{f}_t = A_t^T A_t \tilde{K} (\tilde{\boldsymbol{\alpha}}_{t-1}^* - \tilde{\boldsymbol{\alpha}}_{t-1})$
     $\Delta\tilde{\boldsymbol{\alpha}}^* = \eta_t^* A_t^T (\mathbf{y}_t - \mathbf{f}_t - \varepsilon \mathbf{e})$
     $\Delta\tilde{\boldsymbol{\alpha}} = -\eta_t A_t^T (\mathbf{y}_t - \mathbf{f}_t + \varepsilon \mathbf{e})$
     $\tilde{\boldsymbol{\alpha}}_t^* = \lfloor \tilde{\boldsymbol{\alpha}}_{t-1}^* + \Delta\tilde{\boldsymbol{\alpha}}^* \rceil$
     $\tilde{\boldsymbol{\alpha}}_t = \lfloor \tilde{\boldsymbol{\alpha}}_{t-1} + \Delta\tilde{\boldsymbol{\alpha}} \rceil$

4.2 SOG-SVR Convergence

We now show that the SOG-SVR performs gradient ascent in the *original* Lagrangian, which is the one we actually aim at maximizing. Let us begin by noting that $\mathcal{L}(\boldsymbol{\alpha}^*, \boldsymbol{\alpha})$ from (2.3) may be written as the sum of two other Lagrangians, each defined over orthogonal subspaces of $\mathbb{R}^\ell$.

$$\mathcal{L}(\boldsymbol{\alpha}^*, \boldsymbol{\alpha}) = \hat{\mathcal{L}}(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\alpha}}) + \bar{\mathcal{L}}(\bar{\boldsymbol{\alpha}}^*, \bar{\boldsymbol{\alpha}}) \tag{4.14}$$

where

$$\hat{\boldsymbol{\alpha}}^{(*)} = AA^\dagger \boldsymbol{\alpha}^{(*)} \quad , \quad \bar{\boldsymbol{\alpha}}^{(*)} = (I - AA^\dagger)\boldsymbol{\alpha}^{(*)} \ , \tag{4.15}$$

$$\hat{\mathcal{L}}(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\alpha}}) = -\frac{1}{2}(\hat{\boldsymbol{\alpha}}^* - \hat{\boldsymbol{\alpha}})^T K(\hat{\boldsymbol{\alpha}}^* - \hat{\boldsymbol{\alpha}}) + (\hat{\boldsymbol{\alpha}}^* - \hat{\boldsymbol{\alpha}})^T \mathbf{y} - \varepsilon(\hat{\boldsymbol{\alpha}}^* + \hat{\boldsymbol{\alpha}})^T \mathbf{e} \tag{4.16}$$

$$\bar{\mathcal{L}}(\bar{\boldsymbol{\alpha}}^*, \bar{\boldsymbol{\alpha}}) = (\bar{\boldsymbol{\alpha}}^* - \bar{\boldsymbol{\alpha}})^T \mathbf{y} - \varepsilon(\bar{\boldsymbol{\alpha}}^* + \bar{\boldsymbol{\alpha}})^T \mathbf{e} \ , \tag{4.17}$$

and $A^\dagger = (A^T A)^{-1} A^T$ is the pseudo-inverse of A. It is easy to see that $\hat{\mathcal{L}}$ may be written entirely in terms of $\tilde{\boldsymbol{\alpha}}^{(*)}$ ($AA^\dagger$ is symmetric and therefore $\hat{\boldsymbol{\alpha}} = (A^\dagger)^T \tilde{\boldsymbol{\alpha}}^*$):

$$\hat{\mathcal{L}} = -\frac{1}{2}(\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}})^T \tilde{K}(\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}}) + (\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}})^T A^\dagger \mathbf{y} - \varepsilon(\tilde{\boldsymbol{\alpha}}^* + \tilde{\boldsymbol{\alpha}})^T A^\dagger \mathbf{e} \ , \tag{4.18}$$

Theorem 1. *For η^* and η sufficiently small, using the update rule (4.12) causes a non-negative change to the Lagrangian $\hat{\mathcal{L}}$; i.e.,*

$$\Delta\hat{\mathcal{L}} \stackrel{def}{=} \hat{\mathcal{L}}(\hat{\boldsymbol{\alpha}}^* + \Delta\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\alpha}} + \Delta\hat{\boldsymbol{\alpha}}) - \hat{\mathcal{L}}(\hat{\boldsymbol{\alpha}}^*, \hat{\boldsymbol{\alpha}}) \geq 0 \ . \tag{4.19}$$

Proof. To first order in η^*, $\Delta\hat{\mathcal{L}}$ is proportional to the inner product between the update $\Delta\tilde{\boldsymbol{\alpha}}^*$ and the gradient of $\hat{\mathcal{L}}$ w.r.t. $\tilde{\boldsymbol{\alpha}}^*$. Differentiating (4.18) yields

$$\frac{\partial\hat{\mathcal{L}}}{\partial\tilde{\boldsymbol{\alpha}}^*} = A^\dagger \mathbf{y} - \tilde{K}(\tilde{\boldsymbol{\alpha}}^* - \tilde{\boldsymbol{\alpha}}) - \varepsilon A^\dagger \mathbf{e} = A^\dagger \left(\mathbf{y} - \mathbf{f} - \varepsilon \mathbf{e}\right) \ . \tag{4.20}$$

The inner product mentioned above is:

$$\Delta\tilde{\boldsymbol{\alpha}}^{*T} \frac{\partial\hat{\mathcal{L}}}{\partial\tilde{\boldsymbol{\alpha}}^*} = \eta^* \left(\mathbf{y} - \mathbf{f} - \varepsilon \mathbf{e}\right)^T AA^\dagger \left(\mathbf{y} - \mathbf{f} - \varepsilon \mathbf{e}\right) \geq 0 \ . \tag{4.21}$$

The last inequality is based on the fact that $AA^\dagger$ is positive semi-definite[2]. The exact same treatment can be given to the update $\Delta\tilde{\boldsymbol{\alpha}}$, completing the proof. □

[2] More specifically, $AA^\dagger$ is the projection operator on the subspace of $\mathbb{R}^\ell$ spanned by the columns of A; therefore, its eigenvalues equal either 1 or 0.

A direct consequence of Theorem 1 is that, since $\bar{\boldsymbol{\alpha}}^{(*)}$ is not updated during learning, the change to the Lagrangian $\mathcal{L}$ is also non-negative. However, for a positive ε, if $\tilde{\boldsymbol{\alpha}}^{(*)}$ and $\bar{\boldsymbol{\alpha}}^{(*)}$ are not constrained appropriately, neither $\hat{\mathcal{L}}$ nor $\bar{\mathcal{L}}$ may possess a maximum. Currently, we do not see a way to maintain the feasibility of $\tilde{\boldsymbol{\alpha}}^{(*)}$ w.r.t. the original constraints (2.5) on $\boldsymbol{\alpha}^{(*)}$ with less than $O(\ell)$ work per time-step. For the purposes of convergence and regularization it is sufficient to maintain box constraints similar to (2.5), in $\mathbb{R}^m$; i.e,

$$-\tilde{C}\mathbf{e} \leq \tilde{\boldsymbol{\alpha}}^*, \tilde{\boldsymbol{\alpha}} \leq \tilde{C}\mathbf{e} \ . \tag{4.22}$$

Proving the convergence of SOG-SVR under the constraints (4.22) is a straightforward adaptation of the proof of Theorem 1, and will not be given here for lack of space. In practice, maintaining such constraints seems to be unnecessary.

5 Experiments

Here, we report the results of experiments comparing the SOG-SVR to the state-of-the-art SVR implementation SVMTorch [4]. Throughout, except for the parameters whose values are reported, all other parameters of SVMTorch are at their default values.

We first used SOG-SVR for learning the 1-dimensional *sinc* function $sin(x)/x$ defined on the interval $[-10, 10]$. The kernel is Gaussian with standard deviation $\sigma = 3$. The other SVR parameters are $C = 10^4/\ell$ and $\varepsilon = 0.01$, while the SOG-SVR specific parameters are $\lambda = 0.1$, $\nu = 0.01$ and $\gamma = 0$. Learning was performed on a random set of samples corrupted by additive i.i.d. zero-mean Gaussian noise. Testing was performed on an independent random sample of 1000 noise-free points. All tests were repeated 50 times and the results averaged. Figure 1 depicts the results of two tests. In the first we fixed the noise level (noise std. 0.1) and varied the number of training samples from 5 to 5000, with each training set drawn independently. We then plotted the generalization error (top left) and the number of support vectors as a percentage of the training set (top-right). As can be seen, SOG-SVR produces an extremely sparse solution (with a maximum of 12 SVs) with no significant degradation in generalization level, when compared to SVMTorch. In the second test we fixed the training sample size at 1000 and varied the level of noise in the range 10^{-6} to 10. We note that SVMTorch benefits greatly from a correct estimation of the noise level by its parameter ε. However, at other noise levels, especially in the presence of significant noise, the sparsity of its solution is seriously compromised. In contrast, SOG-SVR produces a sparse solution with complete disregard to the level of noise. It should be noted that SOG-SVR was allowed to make only *a single pass* over the training data, in accordance with its online nature.

We also tested SOG-SVR on two real-world data-sets - Boston housing and Comp-activ, both from Delve[3]. Boston housing is a 13-dimensional data-set with

[3] http://www.cs.toronto.edu/ delve/data/datasets.html

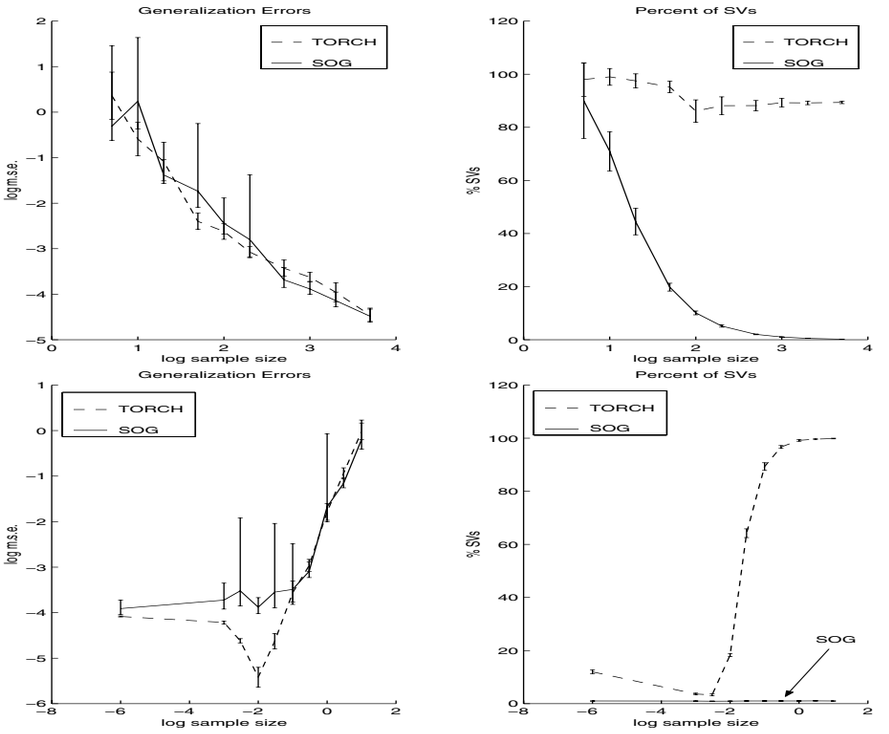

Fig. 1. SOG-SVR compared with SVMTorch on the *sinc* function. The horizontal axis is scaled logarithmically (base 10). In the generalization error graph we use a similar scale for the vertical axis, while on the SV percentage graph we use a linear scale

506 samples. Our experimental setup and parameters are based on [11]. We divided the 506 samples randomly to 481 training samples and 25 test samples. The parameters used were $C = 500$, $\varepsilon = 2$ and $\sigma = 1.4$. The SOG-SVR parameters are $\lambda = 0.1$, $\nu = 0.01$ and $\gamma = 0$. Preprocessing consisted of scaling the input variables to the unit hyper-cube, based on minimum and maximum values. since this is a relatively small data-set, we let SOG-SVR run through the training data 5 times, reporting its generalization error after each iteration. The results shown in Table 3 are averages based on 50 repeated trials. The Comp-activ data-set is a significantly larger 12-dimensional data-set with 8192 samples. Training and test sets were 6000 and 2192 samples long, respectively, and the same preprocessing was performed as for the Boston data-set. Due to the larger size of the training set, SOG-SVR was allowed to make only a single pass over the data. We made no attempt to optimize learning parameters for neither algorithm. The results are summarized in Table 4. As before, results are averaged over 50 trials.

Table 3. Results on the Boston housing data-set, showing the test-set mean-squared error, its standard deviation and the percentage of SVs. SVMTorch is compared with SOG-SVR after 1–5 iterations over the training set. SOG-SVR performs comparably using less than 1/2 of the SVs used by SVMTorch. Throughout, Percentage of SVs has a standard deviation smaller than 1%

Boston	SVMTorch	SOG-SVR 1	SOG-SVR 2	SOG-SVR 3	SOG-SVR 4	SOG-SVR 5
MSE	13.3	40.9	14.1	13.6	13.3	13.1
STD	11.8	69.2	9.3	8.9	8.6	8.4
% SV	37	17	17	17	17	17

Table 4. Results on the Comp-activ data-set, again comparing SOG-SVR with SVMTorch. Here SOG-SVR delivers a somewhat higher test-set error, but benefits from a much more sparse solution

Parameters	**Comp-activ**	SVMTorch	SOG-SVR
$\ell = 6000$, $C = 10^6/\ell = 167$,	MSE	8.8	10.9
$\epsilon = 1$, $\sigma = 0.5$, $\lambda = 0.1$,	STD	0.4	1.0
$\nu = 0.001$, $\gamma = 0$	% SV	63	9

6 Conclusions

We presented a gradient based algorithm for online kernel regression. The algorithm's per-time-step complexity is dominated by an $O(m^2)$ incremental matrix inversion operation, where m is the size of the dictionary. For this reason sparsification, resulting in a small dictionary, is an essential ingredient of SOG-SVR.

Somewhat related to our work are incremental algorithms for SV learning. [12] presented empirical evidence indicating that large data-sets can be split to smaller chunks learned one after the other, augmenting the data in each new chunk with the SVs found previously; with no significant increase in generalization error. This idea was refined in [2], where an SVM algorithm is developed which exactly updates the Lagrange multipliers, based on a single new sample. While the method of [12] lacks theoretical justification, it seems that both methods would be overwhelmed by the increasingly growing number of support vectors found in large online tasks. In [10] an incremental SVM method is suggested in which the locality of RBF-type kernels is exploited to update the Lagrange multipliers of a small subset of points located around each new sample. It is interesting to note that, for RBF kernels, proximity in input space is equivalent to approximate linear dependence in feature space. However, for other, non-local kernels (e.g., polynomial), sparsification by eliminating linear dependencies remains a possible solution. In [7] an incremental kernel classification method is presented which is capable of handling huge data-sets ($\ell = 10^9$). This method results from a quadratic unconstrained optimization problem more closely related to regularized least squares algorithms than to SVMs. The reported algorithm performs linear separation in input space - it would be interesting to see if it

could be extended to non-linear classification and regression through the use of Mercer kernels.

The current work calls for further research. First, a similar algorithm for classification can be developed. This is rather straightforward and can be deduced from the current work. Second, application of SOG-SVR to problems requiring online learning is underway. In particular, we plan to apply SOG-SVR to Reinforcement Learning problems. As indicated by the results on the Boston data-set, SOG-SVR may also be used in an iterative, offline mode simply in order to obtain a sparse solution. Additional tests are required here as well. Third, some technical improvements to the algorithm seem worthwhile. Specifically, the learning rates may be optimized resulting in faster convergence. Fourth, self tuning techniques may be implemented in the spirit of [3]. This would make the algorithm more resilient to scaling problems. Fifth, conditions on the data distribution and the kernel under which the effective rank of the Gram matrix is low, should be studied. Preliminary results suggest that for "reasonable" kernels and large sample size the effective rank of the Gram matrix is indeed much lower than the sample size, and is in fact asymptotically independent of it.

References

1. C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector machines. In *Advances in Neural Information Processing Systems*, volume 9. The MIT Press, 1997. 85
2. G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advnaces in Neural Information Systems*, pages 409–415, 2000. 94
3. O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee. Choosing multiple parameters for support vector machines. *Machine Learning*, 46:131–160, 2002. 95
4. R. Collobert and S. Bengio. SVMTorch: Support vector machines for large-scale regression problems. *Journal of Machine Learning Research*, 1:143–160, 2001. 85, 92
5. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, Cambridge, England, 2000. 84
6. T. Downs, K. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, December 2001. 85, 87
7. G. Fung and O. L. Mangasarian. Incremental support vector machine classification. In *Second SIAM Intrnational Conference on Data Mining*, 2002. 94
8. R. Herbrich. *Learning Kernel Classifiers*. The MIT Press, Cambridge, MA, 2002. 84
9. J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, pages 42–65. MIT Press, 1999. 85
10. L. Ralaivola and F. d'Alché Buc. Incremental support vector machine learning: a local approach. In *Proceedings of ICANN*. Springer, 2001. 94
11. B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002. 84, 85, 93

12. N. Syed, H. Liu, and K. Sung. Incremental learning with support vector machines. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999. 94
13. V. N. Vapnik. *Statistical Learning Theory.* Wiley Interscience, New York, 1998. 84
14. S. Vijayakumar and S. Wu. Sequential support vector classifiers and regression. In *Proceedings of the International Conference on Soft Computing (SOCO'99)*, 1999. 86, 87
15. G. Wabha. Spline models for observational data. CBMS-NSF Regional Conference Series in Applied Mathematics, Vol. 59, Philadelphia: SIAM, 1990. 86