Structural learning in artificial neural networks using sparse optimization

Article *in* Neurocomputing · July 2017

DOI: 10.1016/j.neucom.2017.07.028

3 authors:

Mikael Manngård
Novia University of Applied Sciences
17 PUBLICATIONS **123** CITATIONS
SEE PROFILE

Jan Kronqvist
KTH Royal Institute of Technology
57 PUBLICATIONS **1,117** CITATIONS
SEE PROFILE

Jari Mikael Böling
Åbo Akademi University
40 PUBLICATIONS **801** CITATIONS
SEE PROFILE

Structural learning in artificial neural networks using sparse optimization

Mikael Manngård[a,*], Jan Kronqvist[b], Jari M. Böling[a]

[a]*Process Control Laboratory, Faculty of Science and Engineering, Åbo Akademi Univeristy, Biskopsgatan 8, FIN-20500 Åbo, Finland*
[b]*Process Design and Systems Engineering, Faculty of Science and Engineering, Åbo Akademi Univeristy, Biskopsgatan 8, FIN-20500 Åbo, Finland*

Abstract

In this paper, the problem of simultaneously estimating the structure and parameters of artificial neural networks with multiple hidden layers is considered. A method based on sparse optimization is proposed. The problem is formulated as an ℓ_0-norm minimization problem, so that redundant weights are eliminated from the neural network. Such problems are in general combinatorial, and are often considered intractable. Hence, an iterative reweighting heuristic for relaxing the ℓ_0-norm is presented. Experiments have been carried out on simple benchmark problems, both for classification and regression, and on a case study for estimation of waste heat recovery in ships. All experiments demonstrate the effectiveness of the algorithm.

Keywords: Structural learning, artificial neural networks, sparse optimization, iterative reweighting

1. Introduction

Selecting the structure of models and estimating model parameters are fundamental tasks in many fields of science and engineering. Artificial neural networks (ANN) provide a non-linear way of mapping relations between inputs and outputs, and are often used as black-box models for systems where the underlying dynamics are either unknown or complex. ANN:s have been shown to approximate any continuous function to arbitrary accuracy, provided that the number of nodes is sufficiently large [1]. Thus, large networks can exactly represent an arbitrary training set, and the risk of overfitting the model to noisy data is high. This will often result in bad generalization on new data [2]. Hence, it can be beneficial to keep the network size as small as possible, while maintaining good fit to the data. By reducing the size of the networks and by allowing sparse connectivity, generalization is expected to be improved.

Various methods for reducing the number of weights in feed-forward neural networks, have been proposed in literature. Methods such as ”optimal brain damage” [3] and ”optimal brain surgeon” [4], provide training procedures where parameters are pruned according to their importance. Various regularization approaches, often referred to as ”weight decay” or ”forgetting” in the context of machine learning and neural networks, have traditionally been used to prevent networks from overfitting [5, 6]. Here weight decay and forgetting simply refers to penalizing the weights with an ℓ_2- or ℓ_1-norm respectively, where only the latter is promoting sparsity [5, 7]. From a Bayesian perspective, regularization is associated with assigning a prior distribution to the weights. The Laplacian distribution (ℓ_1-regularization) is a commonly used sparsity promoting prior [8, 9, 10].

The interest in so called ”extreme learning machines” (ELM) have increased in recent years due to the low computational cost of training such networks [11, 12]. Extreme learning machines are large single hidden layer feed-forward neural networks with a randomly generated hidden layer. Since the input layer weights are given, the task of training the network becomes a linear least squares regression problem, which can be solved efficiently. The drawback with ELM:s is that they are easily overfitted. Hence, various techniques for finding sparse representations have been applied [13, 14, 15]. Saxén and Pettersson [16] have suggested an intuitive greedy algorithm for promoting sparsity and

*Corresponding author. Tel.: +358 50 336 1461.
Email addresses: mikael.manngard@abo.fi (Mikael Manngård), jan.kronqvist@abo.fi (Jan Kronqvist), jari.boling@abo.fi (Jari M. Böling)

for determining relevant inputs for single-hidden-layer ANN:s, by iteratively setting the least significant weights to zero. However, in these methods, only the linear output layer weights are identified, while the hidden layer weights are randomly generated.

Recent advances in deep learning, such as weight dropout [17], have made it feasible to train very deep and large neural networks where the number of unknown weights are in the millions. However, having too many weights does not only increase the risk of overfitting the model, but also complicates parallel implementations for training dense neural networks and increases the computational power needed for using such models [18]. Thus, reducing the number of weights by finding sparse representations of neural networks has been of increasing interest recently. Han et al. [19] have proposed a simple method for pruning deep networks by applying ℓ_1- or ℓ_2-norm regularization. Wen et al. [20] have shown that structural sparsity learning can both improve the training accuracy and speed up evaluations for deep neural networks. Furthermore, Liu et al. [21] have used group Lasso [22, 23] for reducing the redundancy of parameters in convolutional neural networks, Scardapane et al. [24] have proposed a regularization strategy for deep neural networks based on group Lasso, Feng and Darrell [25] proposed to use the Indian Buffet Process prior for learning the structure of deep convolutional neural networks, and Lebedev and Lempitsky [26] have proposed a group-wise brain damage process for pruning convolutional layers of neural networks.

In this paper, we apply sparse optimization to simultaneously identify both the network structure and parameters of multiple-hidden-layer artificial neural networks. In sparse optimization, one wants to train a set of weights such that as many as possible are exactly zero. Such type of problems are often referred to as ℓ_0-norm minimization problems, which are combinatorial and considered intractable for large problems [7]. Thus, we relax the problem with iteratively reweighted ℓ_1- and ℓ_2-norm procedures. Similar reweighting strategies have previously been applied in the context of linear regression problems and kernel-based methods, such as least squares support vector machines (LS-SVM) [27, 7, 28, 29, 30]. The proposed reweighting strategies have so far (to the best knowledge of the authors) not been applied for promoting sparsity in multiple-hidden-layer artificial neural networks.

The rest of the paper is organized as follows. In Section 2 we present the main structure and feed-forward procedure of a neural network, and we introduce the main notations that we are going to use throughout the paper. In Section 3, we give an introduction to the sparse optimization problem and present some challenges in solving it. We also propose an approximative way of solving the problem via relaxation of the ℓ_0-norm and an iterative reweighting procedure. In Section 4 the effectiveness of the proposed method is demonstrated on some simple test problems, and to a case study for forecasting of waste heat recovery in ships.

2. Network structure and notation

Consider an artificial neural network with L hidden layers. For a single hidden layer l, the forward propagation for the j:th node is described by

$$z_j^{(l)} = h(\mathbf{w}_j^{(l)}, \mathbf{z}^{(l-1)}), \tag{1}$$

where $\mathbf{z}^{(l-1)}$ is a vector consisting of all outputs from the previous hidden layer and $\mathbf{w}_j^{(l)}$ is a vector containing all weights associated with that node. The non-linear function $h(\cdot)$ is the so called "activation function" for a node, which is typically selected as a sigmoid-, tanh- or radial-basis-function. For convenience we will also introduce the notation $w_{ij}^{(l)}$ to describe a specific weight between nodes i in layer $l-1$ and node j in layer l. We also define a merged weight vector $\mathbf{w} = [\mathbf{w}^{(1)}, ..., \mathbf{w}^{(L)}]^{\mathrm{T}} \in \mathbb{R}^K$ which contains all weights in the network. In this article we only consider networks with a single output, but all results can easily be extended for networks with multiple outputs. Then, given an M-dimensional input vector $\mathbf{x}$ and weights $\mathbf{w}$, the output of the network is described by a non-linear function $f(\mathbf{x}, \mathbf{w}) : \mathbb{R}^M \to \mathbb{R}$. Schematics of a single node at layer l in a feed-forward neural network is presented in Figure 1.

Networks typically have full connectivity between layers. However, large fully-connected-networks are easily overfitted to training data. Hence, we intend to reduce the model complexity by finding a sparse representation of the network (i.e. a representation where most of the weights are zero).

3. Sparse connectivity in neural networks

In this section, we formulate a sparse optimization problem and present some challenges of solving it. We also propose an approach for approximatively solving such problems.

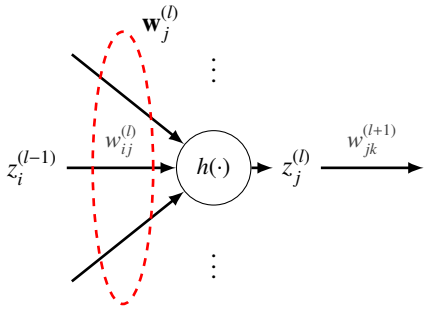

Figure 1: Schematics of a single node j in layer l in a feed-forward ANN.

3.1. Sparse optimization

Given a dataset $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, where $\mathbf{x}_n \in \mathbb{R}^M$ is an input vector and $y_n \in \mathbb{R}$ is the target response. We denote the vector of output data $\mathbf{y} = (y_n)_{n=1}^N$. For the sake of simplicity, we consider only networks with a single output. However, our presented methodology can easily be applied for networks with multiple outputs. The merged vector of model outputs is denoted $\hat{\mathbf{y}} = (f(\mathbf{x}_n, \mathbf{w}))_{n=1}^N$. We are to determine the unknown model parameters $\mathbf{w}$ of a neural network with the structure described in Section 2, so that maximum number of entries in $\mathbf{w}$ are zero while $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 \leq \varepsilon$, where $\varepsilon \geq 0$. An intuitive way of formulating such an optimization problem is

$$(P_0): \quad \begin{aligned} &\underset{\mathbf{w}}{\text{minimize}} && \|\mathbf{w}\|_0 \\ &\text{subject to} && \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 \leq \varepsilon. \end{aligned} \tag{2}$$

Here $\|\cdot\|_0$ is the so called ℓ_0-norm[1] which is defined as

$$\|\mathbf{w}\|_0 := \lim_{p \to 0} \|\mathbf{w}\|_p^p = \lim_{p \to 0} \sum_{k=1}^K |w_k|^p = \sum_{w_k \neq 0} 1, \tag{3}$$

where $\|\mathbf{w}\|_p = \left(\sum_{k=1}^K |w_k|^p\right)^{1/p}$ is the p-norm. As seen from Equation (3), we are using the $\|\cdot\|_0$ as a convenient measure of sparsity of a vector, counting the number of nonzero entries in $\mathbf{w}$. Problem (P_0) can be reformulated by introducing an appropriate Lagrange multiplier $\alpha > 0$, which results in

$$(P_0^\star): \quad \underset{\mathbf{w}}{\text{minimize}} \quad \|\mathbf{w}\|_0 + \alpha \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2. \tag{4}$$

The problem $(P_0^\star)$ is a combinatorial problem, which will become intractable when the number of parameters grows large [7].

For artificial neural neural networks, the number of parameters in the hidden layers will quickly grow large when the size of the network is increased, and hence solving $(P_0^\star)$ through an exhaustive combinatorial search is often not an option. An additional difficulty is the non-convexity of the hidden layers, which will make it difficult to guarantee global optimality of a solution. It is hence well motivated to look for approximative solutions to $(P_0^\star)$, by relaxing the ℓ_0-norm with a continuous function, such as the ℓ_1- or ℓ_2-norm. Furthermore, we introduce an iterative reweighing procedure so that the desirable sparsity promoting properties of the ℓ_0-norm are preserved.

3.2. Approximative solutions to $(P_0^\star)$

In order to approximate the ℓ_0-norm, we first relax it with the ℓ_2-norm, and introduce a diagonal weight matrix $\mathbf{\Lambda}$, which allows us to penalize individual elements in $\mathbf{w}$ differently. The relaxed formulation of $(P_0^\star)$ can be expressed as

$$(P_2): \quad \underset{\mathbf{w}}{\text{minimize}} \quad \|\mathbf{\Lambda}\mathbf{w}\|_2^2 + \alpha \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2. \tag{5}$$

[1] Note that the term "ℓ_0-norm" is a misnomer, since it does not satisfy the homogeneity axiom $\|\beta\mathbf{w}\| = |\beta| \, \|\mathbf{w}\|$ of a norm, for all arbitrary scalars β.

For a given solution $\hat{\mathbf{w}}$ to optimization problem (P_2), we update $\mathbf{\Lambda}$ according to

$$\Lambda_{k,k} = \begin{cases} 1/|\hat{w}_k| & \text{if } |\hat{w}_k| > \epsilon \\ 1/\epsilon & \text{otherwise,} \end{cases}, k = 1, 2, ..., K, \tag{6}$$

where $\epsilon > 0$, cf. [31, 7]. Similar reweighing procedure have previously been proposed by Huang et al. [28], Lopez et al. [29] to promote sparsity in LS-SVM:s. By solving problem (P_2) and updating $\mathbf{\Lambda}$ according to Equation (6) iteratively, until the parameters $\hat{\mathbf{w}}$ converge, we get

$$\begin{aligned} \|\mathbf{\Lambda}\hat{\mathbf{w}}(t)\|_2^2 &= \sum_{k=1}^{K} \Lambda_{k,k}^2 \hat{w}_k(t)^2 \\ &= \sum_{\hat{w}_k(t)=0} \frac{0}{\epsilon^2} + \sum_{\hat{w}_k(t)\neq 0} \frac{\hat{w}_k(t)^2}{|\hat{w}_k(t-1)|^2} = \sum_{\hat{w}_k(t)\neq 0} 1 =: \|\hat{\mathbf{w}}(t)\|_0, \end{aligned} \tag{7}$$

where t is the iteration index. Thus we may view the ℓ_0-norm as an adequately weighted version of the squared ℓ_2-norm.

To make sure that some weights will become exactly zero, we need to apply a threshold for $\hat{w}_k$ so that

$$\hat{w}_k = \begin{cases} 0 & \text{if } |\hat{w}_k| \leq \epsilon \\ \hat{w}_k & \text{otherwise} \end{cases}, k = 1, 2, ..., K. \tag{8}$$

Finally, we define the stop criteria for the algorithm

$$\|\hat{\mathbf{w}}(t) - \hat{\mathbf{w}}(t-1)\|_2 < \delta, \tag{9}$$

where $\delta > 0$.

The complete algorithm is presented in Algorithm 1.

Algorithm 1 Iterative procedure for approximative ℓ_0-norm minimization by ℓ_2-relaxation.

given initial parameters $\mathbf{w}$, $\mathbf{\Lambda} := \mathbf{I}_K$, $t := 1$.
repeat
Solve optimization problem (P_2) and get solution $\hat{\mathbf{w}}(t)$.
Set $\hat{w}_k(t) = \begin{cases} 0 & \text{if } |\hat{w}_k(t)| \leq \epsilon \\ \hat{w}_k(t) & \text{otherwise} \end{cases}, k = 1, 2, ..., K.$
Update $\mathbf{\Lambda}$ by $\Lambda_{k,k} = \begin{cases} 1/|\hat{w}_k(t)| & \text{if } |\hat{w}_k(t)| > \epsilon \\ 1/\epsilon & \text{otherwise} \end{cases}, k = 1, 2, ..., K$
Set $t := t + 1$
until stop criteria (9) is satisfied.

An alternative way to relax the ℓ_0-norm is by replacing the ℓ_2-relaxation with an ℓ_1-relaxation. We define the optimization problem (P_1) as follows

$$\begin{aligned} (P_1): \quad \underset{\mathbf{w}}{\text{minimize}} \quad & \|\mathbf{\Lambda}\mathbf{w}\|_1 + \alpha\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2 \\ & = \sum_{k=1}^{K} \Lambda_{k,k}|w_k| + \alpha\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2. \end{aligned} \tag{10}$$

The benefit of using the ℓ_1-norm is that no threshold is needed in order to make weights go exactly to zero [5, 7]. However, the gradient of the ℓ_1-norm is undefined for $w_k = 0$, and a subgradient needs to be specified. In this case,

the intuitive choice is to define it as to be zero in $w_k = 0$, which results in a valid subgradient. Equation (6) can still be used for updating $\mathbf{\Lambda}$, since now we also get

$$\begin{aligned} \|\mathbf{\Lambda}\hat{\mathbf{w}}(t)\|_1 &= \sum_{k=1}^{K} \Lambda_{k,k}|\hat{w}_k(t)| \\ &= \sum_{\hat{w}_k(t)=0} \frac{0}{\epsilon} + \sum_{\hat{w}_k(t)\neq 0} \frac{|\hat{w}_k(t)|}{|\hat{w}_k(t-1)|} = \sum_{\hat{w}_k(t)\neq 0} 1 =: \|\hat{\mathbf{w}}(t)\|_0 \end{aligned} \tag{11}$$

after convergence. The complete algorithm using the ℓ_1-relaxation is presented in Algorithm 2.

Algorithm 2 Iterative procedure for approximative ℓ_0-norm minimization by ℓ_1-relaxation.

given initial parameters $\mathbf{w}$, $\mathbf{\Lambda} := \mathbf{I}_K$, $t := 1$.
repeat
  Solve optimization problem (P_1) and get solution $\hat{\mathbf{w}}(t)$.
  Update $\mathbf{\Lambda}$ by $\Lambda_{k,k} = \begin{cases} 1/|\hat{w}_k(t)| & \text{if } |\hat{w}_k(t)| > \epsilon \\ 1/\epsilon & \text{otherwise} \end{cases}$, $k = 1, 2, ..., K$
  Set $t := t + 1$
until stop criteria (9) is satisfied.

To ensure that Algorithms 1 and 2 are useful, one would need to show that $\hat{w}(t)$ converges as $t \to \infty$. Some convergence claims have previously been made by Huang et al. [28] in the context of kernel-based regression and LS-SVM and by Figueiredo [10] in a Bayesian framework. However, a rigorous proof for convergence using artificial neural network models is left as an open problem. We will instead demonstrate the effectiveness of the algorithms by applying it on benchmark problems. In the next section we propose a simple backtracking gradient descent algorithm, which guarantees convergence (to a local optima) within each iteration.

3.3. Backtracking line search

The simplest way of training ANN:s is by using gradient descent methods with fixed step length s [5]. However, one cannot guarantee that an improvement in the objective is obtained in each iteration when using a fixed step length. There are several ways of performing a line search with variable step lengths. Here we are considering a simple and quite effective algorithm referred to as "backtracking line search" [32]. Given a descent direction $\Delta\mathbf{w}$, backtracking line search only requires two additional parameters $\eta \in (0, 0.5)$ and $\zeta \in (0, 1)$ and a few extra function evaluations per iteration, compared to standard gradient descent with a fixed step length. The backtracking line search starts with a unit step length $s = 1$ and reduces it with a factor ζ until the stopping condition

$$f(\mathbf{w} + s\Delta\mathbf{w}) \leq f(\mathbf{w}) + \eta s \nabla f(\mathbf{w})^{\mathrm{T}} \Delta\mathbf{w} \tag{12}$$

holds. Figure 2 illustrates the main idea of the backtracking algorithm. Inequality (12) ensures that the backtracking algorithm always results in an improvement in objective function in each step [32]. However, unless the function $f(\mathbf{w})$ is convex, we cannot guarantee finding the global optimum, and ANN:s are generally non-convex. The complete gradient descent algorithm using backtracking line search is presented in Algorithm 3.

4. Experiments

The proposed algorithm has been evaluated on two simple test problems and on a case study. In the first example we use radial-basis-function type (RBF) networks to perform regression. The second problem is a two-spiral classification problem. Finally we apply the method to a case study for estimating waste heat recovery in ships. In all experiments, backtracking parameters $\eta = 0.2$ and $\zeta = 0.5$ have been used.

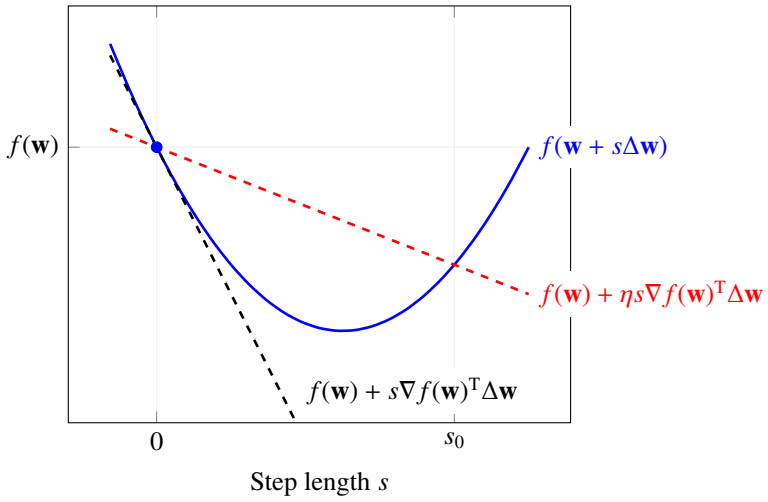

Figure 2: Illustration of the backtracking line search method, where f is the objective function. The black dashed line is a linear extrapolation in the point $f(\mathbf{w})$, the red dashed line has a slope factor $0 < \eta < 0.5$ and is an upper approximation of the optimum solution. The backtracking condition is that f lies below the upper dashed line, which gives $0 < s \leq s_0$ [32].

Algorithm 3 Gradient descent with backtracking line search [32].

given a starting point $\mathbf{w}$, $\eta \in (0, 0.5)$, $\zeta \in (0, 1)$.
while stopping criterion is not satisfied **do**
  $\Delta\mathbf{w} := -\nabla f(\mathbf{w})$.
  Line search, set $s := 1$
    while $f(\mathbf{w} + s\Delta\mathbf{w}) > f(\mathbf{w}) + \eta s \nabla f(\mathbf{w})^{\mathrm{T}} \Delta\mathbf{w}$ **do**
      $s := \zeta s$
    end while
  Update $\mathbf{w} := \mathbf{w} + s\Delta\mathbf{w}$.
end while

4.1. Illustrative example: Regression

We consider the regression problem of estimating the *sinc* function

$$g(x) = \frac{\sin(x)}{x} \tag{13}$$

from noisy data, where the noise is zero mean Gaussian with variance $\sigma^2 = 0.005$. Given a training data set of $N = 300$ samples, we train a single layer RBF-network with the basis function

$$h(\mu_i, x_n) = \exp\left(-\gamma \|x_n - \mu_i\|_2^2\right), \tag{14}$$

where μ_i is the center of the i:th radial basis function and γ is a pre-specified width parameter. Using $M = N = 300$ number of basis functions, and selecting the centres as $\mu_i = x_i$, we construct the hidden-layer-matrix

$$\mathbf{H} = \begin{bmatrix} h(x_1, x_1) & \cdots & h(x_N, x_1) \\ \vdots & \ddots & \vdots \\ h(x_1, x_N) & \cdots & h(x_N, x_N) \end{bmatrix}. \tag{15}$$

The matrix $\mathbf{H}$ is an N-by-N, symmetric, positive-semi-definite Mercer kernel [33]. The output of the network is then given by the linear equation

$$\mathbf{y} = \mathbf{H}\mathbf{w}. \tag{16}$$

Optimization problem (P_2) is formulated as

$$(P_2): \quad \underset{\mathbf{w}}{\text{minimize}} \quad \|\mathbf{\Lambda w}\|_2^2 + \alpha \|\mathbf{y} - \mathbf{H w}\|_2^2 =: L(\mathbf{w}), \tag{17}$$

which is a quadratic problem. Taking the derivative of the loss function $L(\mathbf{w})$ and setting it equal to zero, gives the optimality condition

$$\frac{\partial L}{\partial \mathbf{w}} = 2\mathbf{\Lambda}^{\mathrm{T}}\mathbf{\Lambda w} - 2\alpha \mathbf{H}^{\mathrm{T}}\mathbf{y} + 2\alpha \mathbf{H}^{\mathrm{T}}\mathbf{H w} = \mathbf{0}, \tag{18}$$

which gives us the explicit solution

$$\hat{\mathbf{w}} = \left(\mathbf{H}^{\mathrm{T}}\mathbf{H} + \alpha^{-1}\mathbf{\Lambda}^{\mathrm{T}}\mathbf{\Lambda}\right)^{-1} \mathbf{H}^{\mathrm{T}}\mathbf{y}. \tag{19}$$

This problem is often referred to as "ridge regression". Since an explicit solution exists, Algorithm 1 can be used to find sparse network structures, for given widths and centres of the radial basis functions. This problem now shows great resemblance the LS-SVM problem [34, 35, 30] (with iterative reweighting). A more detailed discussion on the single-hidden layer ANN interpretation of LS-SVM has been presented in [36].

Similarly, the optimization problem using an ℓ_1-relaxation becomes

$$(P_1): \quad \underset{\mathbf{w}}{\text{minimize}} \quad \|\mathbf{\Lambda w}\|_1 + \alpha \|\mathbf{y} - \mathbf{H w}\|_2^2 =: L(\mathbf{w}). \tag{20}$$

Although this optimization problem is convex, one cannot solve it explicitly, and we need to rely on some convex optimization framework such as CVX [37, 38].

Parameters α and γ are typically determined through cross-validation. In Table 1, results from cross-validation are presented, and some selected results obtained by running both Algorithms 1 and 2 are visualized in Figure 3. From Table 1 it becomes clear that for the unregularized case (where $\alpha = 0$) the model is overfitted to the noisy data, especially if we choose γ to be large. We were able to decrease the number of non-zero elements in $\mathbf{w}$ significantly for both cases. Surprisingly, for this test problem, the ℓ_2-norm regularization seem to result in more sparse solutions that the ℓ_1-norm regularization.

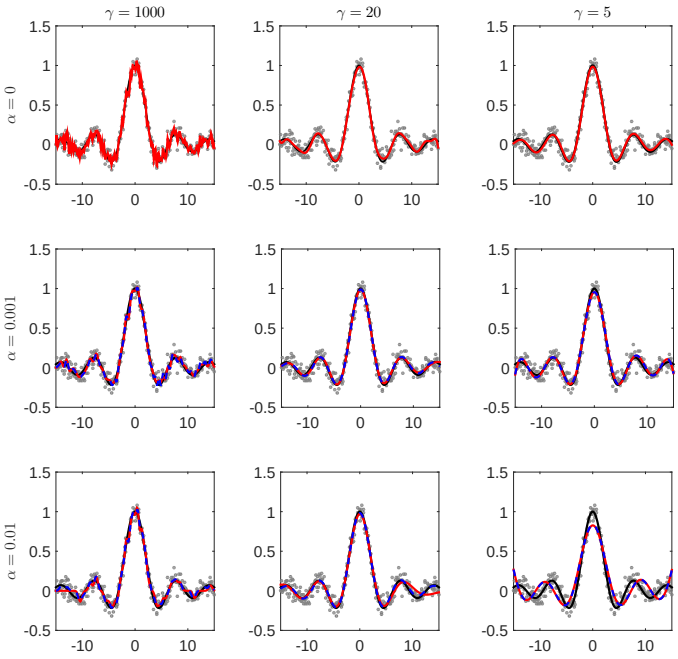

Figure 3: The sinc function (black) is estimated from and noisy data (gray dots). Results for models fitted with Algorithm 2 (blue) and with Algorithm 1 (red) are presented for parameters $\alpha = \{0, 0.001, 0, 01\}$ and $\gamma = \{1000, 20, 5\}$.

Table 1: Crossvalidation results for estimating the sinc function. Root mean square error for training (RMSE_t), validation (RMSE_v), on noise free data (RMSE_0) and number of nonzero parameters $\|\hat{w}\|_0$. Various values of α and γ were used for both problem formulations (P_1) and (P_2). The standard deviation of the noise is 0.07. Results have been rounded to four decimals.

		Optimization problem (P_2)				Optimization problem (P_1)			
α	γ	RMSE_t	RMSE_v	RMSE_0	$\|\hat{\mathbf{w}}\|_0$	RMSE_t	RMSE_v	RMSE_0	$\|\hat{\mathbf{w}}\|_0$
0	1000	0.0338	0.0891	0.0586	300	0.0330	0.0891	0.0586	300
	100	0.0573	0.0780	0.0359	300	0.0573	0.0780	0.0359	300
	50	0.0613	0.0753	0.0286	300	0.0613	0.0753	0.0286	300
	20	0.0619	0.0722	0.0241	300	0.0619	0.0722	0.0241	300
	10	0.0642	0.0711	0.0212	300	0.0642	0.0711	0.0212	300
	5	0.0733	0.0774	0.0357	300	0.0733	0.0774	0.0357	300
0.001	1000	0.0581	0.0765	0.0342	43	0.0557	0.0768	0.0365	58
	100	0.0644	0.0720	0.0221	16	0.0638	0.0720	0.0216	21
	50	0.0653	0.0704	0.0161	13	0.0651	0.0704	0.0200	13
	20	0.0664	0.0705	0.0155	11	0.0656	0.0697	0.0155	12
	10	0.0661	0.0694	0.0144	11	0.0660	0.0693	0.0145	11
	5	0.0706	0.0739	0.0276	11	0.0707	0.0739	0.0277	11
0.01	1000	0.0662	0.0796	0.0408	23	0.0618	0.0773	0.0347	30
	100	0.0726	0.0780	0.0395	8	0.0652	0.0733	0.0232	14
	50	0.0729	0.0772	0.0390	7	0.0656	0.0705	0.0189	11
	20	0.0759	0.0813	0.0403	9	0.0660	0.0697	0.0135	11
	10	0.0832	0.0840	0.0457	9	0.0662	0.0693	0.0139	11

4.2. Illustrative example: Classification

In this section we have validated our proposed methods on a classification benchmark problem, by training a multiple hidden layer, feed-forward, sigmoidal neural networks. A data set $\{\mathbf{x}_n, y_n\}_{n=1}^{200}$ was used, where $\mathbf{x}_n \in \mathbb{R}^2$ and $y_n \in \{0, 1\}$, with an equal amount of data points from each class. The data is illustrated in Figure 4. A two-hidden-layer neural network with 2-30-20-1 nodes in respective layers was used. This makes the initial total number of weights (excluding biases) $K = 680$, which is greater than the total number of training samples. Since $K > N$ the model complexity needs to be reduced. Thus, Algorithm 1, together with the backtracking batch-gradient-descent presented in 3, are used to train the ANN so that sparsity in the hidden layers are promoted. We denote the set of weights in the hidden layers $\mathbf{w}^\star$. Optimization problem (P_2) is re-formulated as

$$(P_2): \quad \underset{\mathbf{w}}{\text{minimize}} \quad \|\mathbf{\Lambda}\mathbf{w}^\star\|_2^2 + \alpha\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2, \tag{21}$$

so that only the hidden layer weights are regularized. Parameters $\alpha = 0.01$ and $\epsilon = 10^{-3}$ were used. The resulting boundaries separating the two classes is presented in Figure 4. The obtained sparse network, has the structure 2-16-6-1 after training, and has only 22-37-6 number of non-zero weights in each layer respectively. In other words, the number of non-zero parameters have been reduced from initially 680 to only 65, which is a 90% reduction in non-zero weights.

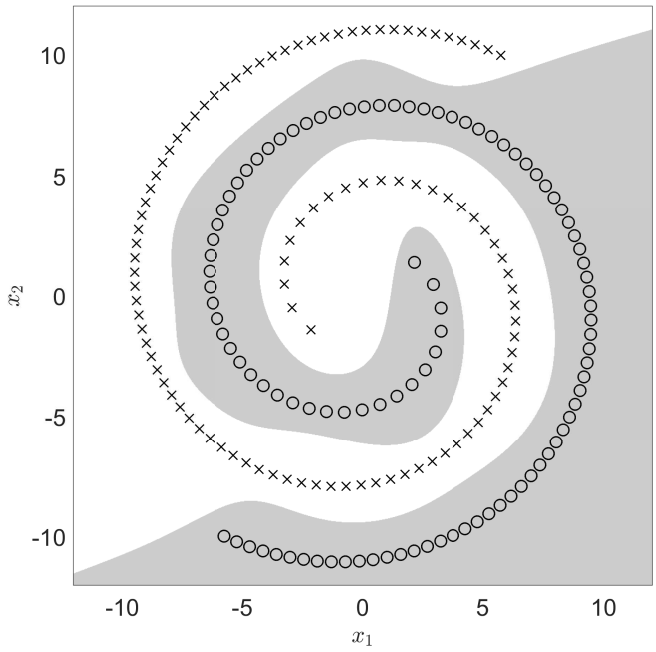

Figure 4: The training data for the two-spiral classification problem, where the two classes are represented with '×' and '∘' respectively. The classification output of the network is represented by the white/gray background.

4.3. Case study: Estimation of waste heat recovery in ships

We now present a case study for forecasting of waste heat recovery in ships. The diesel engine is the main source of power for the entire ship. However, not all energy in the fuel is converted into mechanical work, but a significant amount is also converted into thermal energy. Such thermal energy is here refereed to as waste heat. Waste heat can for example be re-utilized via heat exchangers. Utilizing excess heat from a system is referred to as waste heat recovery. Since the recovered heat can be reused for tasks such as general heating or fresh water production, the efficiency of the the system is indirectly increased. It is hence of interest to be able to accurately predict the amount of available recovered waste heat, so that on-board tasks can be scheduled optimally and the amount of required additional heating can be minimized.

Schematics of an engine cooling system is presented in Figure 5. Although the main purpose of the cooling system is to keep the engine temperature within an acceptable range, excess heat can be extracted via a heat exchanger (HE). The system has three splitting/mixing valves (V1, V2, V3) which allow control of engine temperature. Valve (V1) regulates the engine temperature by bypassing part of the flow around the heat exchanger, valve (V2) is a splitting valve which connects the high temperature (HT) and low temperature (LT) loops, and valve (V3) a mixing valve. We denote the added thermal heat flow rate from the engine with $\dot{Q}_{\mathrm{H}}$, and the extracted waste heat flow rate by $\dot{Q}_{\mathrm{R}}$. In practice one cannot measure $\dot{Q}_{\mathrm{H}}$ directly, but only the normalized electric load $P \in [0, 1]$ of the engine is known. The mapping between electric load and the heat load is a (non-linear) function due to varying efficiency of the engine at different loads. Hence we recognize that the heat flow rate is described by a function $g(P) : [0, 1] \to \mathbb{R}$, so that $\dot{Q}_{\mathrm{H}} = g(P)$. For simplicity we assume that all temperatures T can be measured and that the mass flow rates $\dot{m}$ in the systems are constant. We also assume that there is no heat loss in the valves and that there are no dynamics present in the system, which is realistic if the sampling time of the measurements is high. For valve (V1) we introduce a splitting constant $\kappa \in [0, 1]$ so that

$$\dot{m}_2 = \kappa \dot{m}_1 \tag{22}$$

$$\dot{m}_5 = (1 - \kappa)\dot{m}_1. \tag{23}$$

Since we assumed that mass flow rates are constant, it follows that

$$\dot{m}_3 = \dot{m}_2 = \kappa \dot{m}_1. \tag{24}$$

$$\dot{m}_4 = \dot{m}_3 + \dot{m}_5 = \dot{m}_1. \tag{25}$$

It is also clear that $T_2 = T_5 = T_1$ since we assumed no heat loss.

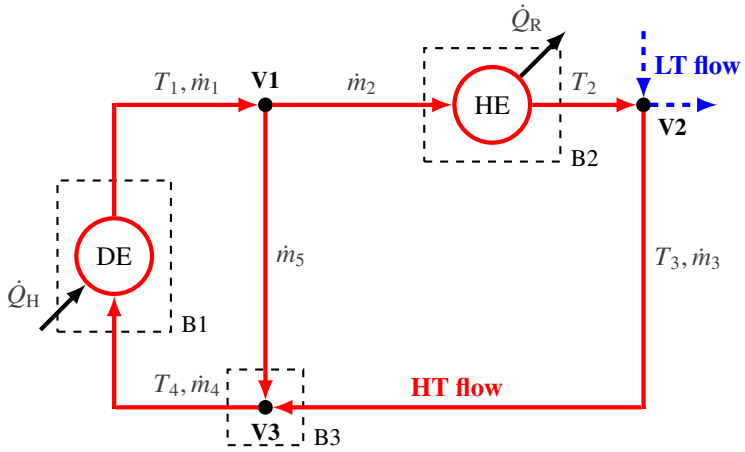

Figure 5: Simple schematics of the cooling system, consisting of a hight temperature (HT) and low temperature loop (LT). The HT-circuit consists of a diesel engine (DE), a heat exchanger (HE) and three mixing/splitting valves (V). Temperatures T, mass flow rates $\dot{m}$ and heat flow rates $\dot{Q}$ have been marked out in the figure. Three boundaries (B) for mass and heat balances have been marked with dashed lines.

For the cooling circuit illustrated in Figure 5, we set up three boundaries (B1, B2, B3) for which we form mass and heat balances. Introducing the specific heat capacity c_p, we get

$$\begin{aligned}
&B1: && c_p \dot{m}_1 T_1 = c_p \dot{m}_4 T_4 + g(P) \\
&\overset{(25)}{\Leftrightarrow} && (T_4 - T_1) = -\frac{1}{c_p \dot{m}_1} g(P) \qquad (26) \\
&B3: && c_p \dot{m}_4 T_4 = c_p \dot{m}_3 T_3 + c_p \dot{m}_5 T_5 \\
&\overset{(23,\,24,\,25)}{\Leftrightarrow} && \dot{m}_1 T_4 = \kappa \dot{m}_1 T_3 + (1-\kappa) \dot{m}_1 T_1 \\
&\Leftrightarrow && \kappa = \frac{T_4 - T_1}{T_3 - T_1} \qquad (27) \\
&B2: && c_p \dot{m}_2 T_2 = c_p \dot{m}_2 T_1 - \dot{Q}_R \\
&\overset{(22)}{\Leftrightarrow} && \dot{Q}_R = c_p \kappa \dot{m}_1 (T_1 - T_2) \\
& && \overset{(27)}{=} c_p \dot{m}_1 \frac{(T_4 - T_1)(T_1 - T_2)}{(T_3 - T_1)} \\
& && \overset{(26)}{=} -\frac{c_p \dot{m}_1 (T_1 - T_2)}{c_p \dot{m}_1 (T_3 - T_1)} g(P) = \frac{(T_2 - T_1)}{(T_3 - T_1)} g(P). \qquad (28)
\end{aligned}$$

Given a set of data points $\{T_n, \dot{Q}_{R,n}, P_n\}_{n=1}^N$, we define a new target vector

$$y := \frac{(T_3 - T_1)}{(T_2 - T_1)} \dot{Q}_R, \tag{29}$$

and form the following regression model

$$\hat{y}_n = g(P_n). \tag{30}$$

Since $g(P)$ is an unknown, possibly non-linear function, we can estimate it with an artificial neural network. It is worth mentioning that the target vector defined in (29) is always non singular since $T_1 > T_2$. Simulated data of a fresh water cooling system was provided by Zou et al. [39]. A total number of $N = 500$ data points, with sampling time 100 seconds, were used for estimation $g(P)$. The recovered energy was normalized before training with respect to the maximum engine load, which is 16.8 MW. We apply our proposed sparse optimization technique, according to Algorithm 1, to identify an appropriate network size and structure. A fully connected single hidden-layer sigmoid neural network with $M = 20$ hidden nodes was initially used. For $\epsilon = 10^{-3}$ and $\alpha = 0.01$ the final network consisted of only 2 nodes in the hidden layer. The low number of nodes left after training seem reasonable due to the close to linear appearance of the target data (see Figure 6). Results for the mapping between electric and heat load, and between electric load and recovered waste heat energy are presented in Figure 6.

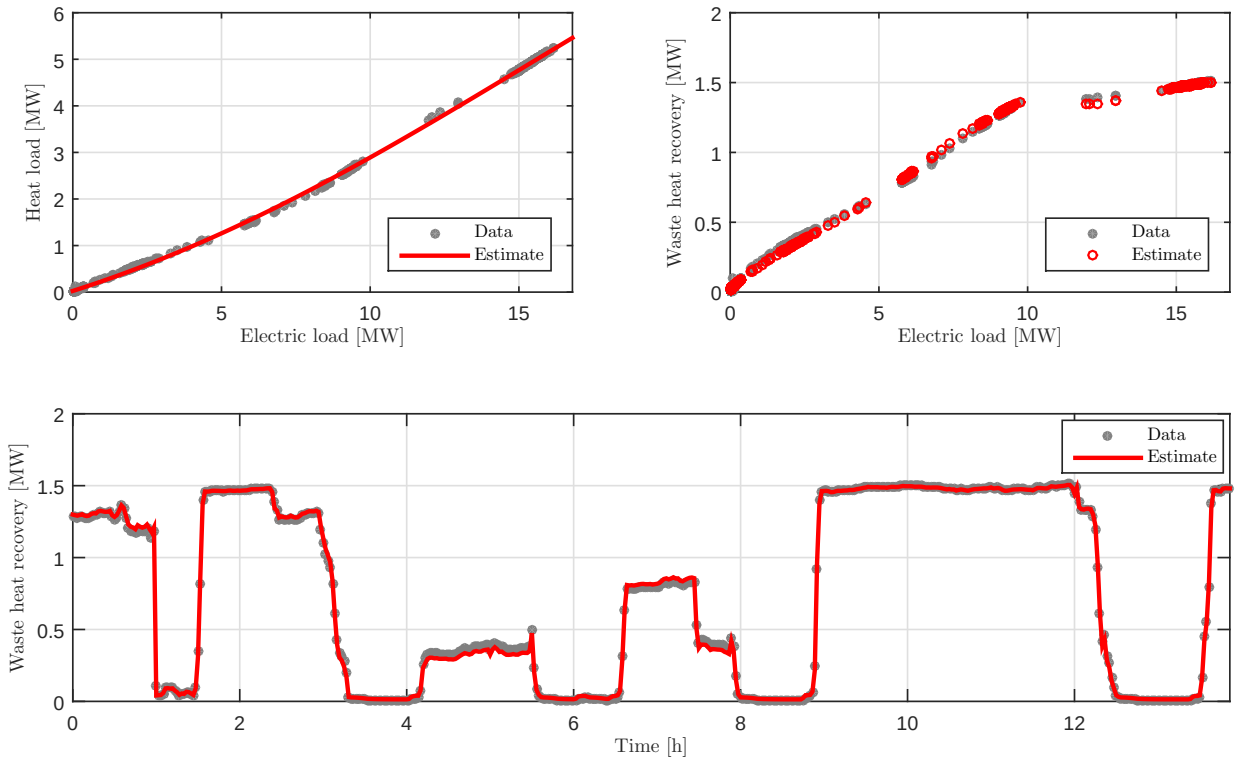

Figure 6: Estimated heat load and recovered waste heat energy. The first figure (top left), shows the mapping from electric load to heat load. The second figure (top right) displays the mapping between electric load obtained by Equation 29, given some temperature measurements. The third figure (bottom) shows the fit on training data.

5. Concluding remarks

In this paper we proposed a sparse optimization technique, based on iterative reweighing for structural learning in artificial neural networks. The proposed method was presented in a general form so that it applies to various types of network types, such as multi-layer perceptions, radial-basis-function networks or extreme learning machines. Experimental results for both regression as well as classification demonstrate the effectiveness of the algorithm. Model complexity was significantly reduced in all cases, and sparse network structures were obtained. Additional structural constraints, such as group sparsity, can easily be included in our proposed framework via group Lasso.

For RBF-type networks, where the basis functions are convex, the hidden layer can be interpreted as an positive-semi-definite kernel, which satisfies Mercer's conditions. This makes it possible to find explicit solutions for each iteration step, which makes the proposed algorithm computationally efficient. In this case, the proposed problem formulation shows strong similarities to the least-squares support-vector-machines, but the presented iterative reweighting procedure makes it possible to promote sparsity in the model.

Finding a rigorous convergence proof still remains an open problem, and further work could be directed in providing such. Future work could be directed towards evaluating the effectiveness of the reweighting strategy identifying sparse structures of deep neural networks.

Acknowledgements

This work was carried out in the Efficient Energy Use (EFEU) research program coordinated by CLEEN Ltd. with funding from the Finnish Funding Agency for Technology and Innovation, Tekes. We also acknowledge support from the Finnish Graduate School in Chemical Engineering (GSCE).

References

[1] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of control, signals and systems 2 (4) (1989) 303–314.
[2] I. V. Tetko, D. J. Livingstone, A. I. Luik, Neural network studies. 1. Comparison of overfitting and overtraining, Journal of Chemical Information and Computer Sciences 35 (5) (1995) 826–833.
[3] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, L. D. Jackel, Optimal brain damage, in: Advances in Neural Information Processing Systems, vol. 2, 598–605, 1989.
[4] B. Hassibi, D. G. Stork, G. J. Wolff, Optimal brain surgeon and general network pruning, in: IEEE International Conference on Neural Networks, IEEE, 293–299, 1993.
[5] C. M. Bishop, Pattern recognition and machine learning, Springer, 2006.
[6] M. Ishikawa, Structural learning with forgetting, Neural Networks 9 (3) (1996) 509–521.
[7] M. Elad, Sparse and redundant representations: from theory to applications in signal and image processing, Springer Science & Business Media, 2010.
[8] P. M. Williams, Bayesian regularization and pruning using a Laplace prior, Neural Computation 7 (1) (1995) 117–143.
[9] C. Goutte, L. K. Hansen, Regularization with a pruning prior, Neural Networks 10 (6) (1997) 1053–1059.
[10] M. A. Figueiredo, Adaptive sparseness for supervised learning, IEEE Transactions on Pattern Analysis and Machine Intelligence 25 (9) (2003) 1150–1159.
[11] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1) (2006) 489–501.
[12] G.-B. Huang, D. H. Wang, Y. Lan, Extreme learning machines: a survey, International Journal of Machine Learning and Cybernetics 2 (2) (2011) 107–122.
[13] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, A. Lendasse, OP-ELM: optimally pruned extreme learning machine, IEEE Transactions on Neural Networks 21 (1) (2010) 158–162.
[14] G.-B. Huang, H. Zhou, X. Ding, R. Zhang, Extreme learning machine for regression and multiclass classification, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 42 (2) (2012) 513–529.
[15] Z. Bai, G.-B. Huang, D. Wang, H. Wang, M. B. Westover, Sparse extreme learning machine for classification, IEEE Transactions on Cybernetics 44 (10) (2014) 1858–1870.
[16] H. Saxén, F. Pettersson, Method for the selection of inputs and structure of feedforward neural networks, Computers & Chemical Engineering 30 (6) (2006) 1038–1045.
[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting., Journal of Machine Learning Research 15 (1) (2014) 1929–1958.
[18] B. Recht, C. Re, S. Wright, F. Niu, Hogwild: A lock-free approach to parallelizing stochastic gradient descent, in: Advances in Neural Information Processing Systems, 693–701, 2011.
[19] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 1135–1143, 2015.
[20] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning Structured Sparsity in Deep Neural Networks, arXiv preprint arXiv:1608.03665 .
[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, M. Pensky, Sparse convolutional neural networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 806–814, 2015.
[22] M. Yuan, Y. Lin, Model selection and estimation in regression with grouped variables, Journal of the Royal Statistical Society: Series B (Statistical Methodology) 68 (1) (2006) 49–67.
[23] J. Friedman, T. Hastie, R. Tibshirani, A note on the group lasso and a sparse group lasso, arXiv preprint arXiv:1001.0736 .
[24] S. Scardapane, D. Comminiello, A. Hussain, A. Uncini, Group Sparse Regularization for Deep Neural Networks, arXiv preprint arXiv:1607.00485 .
[25] J. Feng, T. Darrell, Learning The Structure of Deep Convolutional Networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2749–2757, 2015.
[26] V. Lebedev, V. Lempitsky, Fast convnets using group-wise brain damage, arXiv preprint arXiv:1506.02515 .

[27] E. J. Candes, M. B. Wakin, S. P. Boyd, Enhancing sparsity by reweighted ℓ_1 minimization, Journal of Fourier Analysis and Applications 14 (5-6) (2008) 877–905.
[28] K. Huang, D. Zheng, J. Sun, Y. Hotta, K. Fujimoto, S. Naoi, Sparse learning for support vector classification, Pattern Recognition Letters 31 (13) (2010) 1944–1951.
[29] J. Lopez, K. De Brabanter, J. Dorronsoro, J. A. K. Suykens, Sparse LSSVMs with L_0-norm minimization, in: Proceedings of the European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2011), 189–194, 2011.
[30] R. Mall, J. A. K. Suykens, Very Sparse LSSVM Reductions for Large-Scale Data, IEEE transactions on neural networks and learning systems 26 (5) (2015) 1086–1097.
[31] I. F. Gorodnitsky, B. D. Rao, Sparse signal reconstruction from limited data using FOCUSS: A re-weighted minimum norm algorithm, IEEE Transactions on signal processing 45 (3) (1997) 600–616.
[32] S. Boyd, L. Vandenberghe, Convex optimization, Cambridge University Press, 2004.
[33] N. Cristianini, J. Shawe-Taylor, An introduction to support vector machines and other kernel-based learning methods, Cambridge University Press, 2000.
[34] J. A. K. Suykens, J. Vandewalle, Least squares support vector machine classifiers, Neural Processing Letters 9 (3) (1999) 293–300.
[35] J. A. K. Suykens, J. De Brabanter, L. Lukas, J. Vandewalle, Weighted least squares support vector machines: robustness and sparse approximation, Neurocomputing 48 (1) (2002) 85–105.
[36] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, J. Vandewalle, Least squares support vector machines, World Scientific, Singapore, 2002.
[37] M. Grant, S. Boyd, CVX: Matlab Software for Disciplined Convex Programming, version 2.1, http://cvxr.com/cvx, 2014.
[38] M. Grant, S. Boyd, Graph implementations for nonsmooth convex programs, in: V. Blondel, S. Boyd, H. Kimura (Eds.), Recent Advances in Learning and Control, Lecture Notes in Control and Information Sciences, Springer-Verlag Limited, 95–110, http://stanford.edu/~boyd/graph_dcp.html, 2008.
[39] G. Zou, M. Elg, A. Kinnunen, P. Kovanen, K. Tammi, K. Tervo, Modeling ship energy flow with multi-domain simulation, in: CIMAC Congress, Shanghai, CIMAC, 2013.