Accepted refereed manuscript of:

Scardapane S, Comminiello D, Hussain A & Uncini A (2017) Group Sparse Regularization for Deep Neural Networks, *Neurocomputing*, 241, pp. 81-89.

DOI: 10.1016/j.neucom.2017.02.029

© 2017, Elsevier. Licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International
http://creativecommons.org/licenses/by-nc-nd/4.0/

Group Sparse Regularization for Deep Neural Networks

Simone Scardapane[a,*], Danilo Comminiello[a], Amir Hussain[b], Aurelio Uncini[a]

[a] *Department of Information Engineering, Electronics and Telecommunications (DIET), "Sapienza" University of Rome, Via Eudossiana 18, 00184 Rome, Italy*
[b] *Division of Computing Science & Maths, School of Natural Sciences, University of Stirling, Stirling FK9 4LA, Scotland, UK.*

Abstract

In this paper, we address the challenging task of simultaneously optimizing (i) the weights of a neural network, (ii) the number of neurons for each hidden layer, and (iii) the subset of active input features (i.e., feature selection). While these problems are traditionally dealt with separately, we propose an efficient regularized formulation enabling their simultaneous parallel execution, using standard optimization routines. Specifically, we extend the group Lasso penalty, originally proposed in the linear regression literature, to impose group-level sparsity on the network's connections, where each group is defined as the set of outgoing weights from a unit. Depending on the specific case, the weights can be related to an input variable, to a hidden neuron, or to a bias unit, thus performing simultaneously all the aforementioned tasks in order to obtain a compact network. We carry out an extensive experimental

*Corresponding author. Phone: +39 06 44585495, Fax: +39 06 4873300.
Email addresses: simone.scardapane@uniroma1.it (Simone Scardapane), danilo.comminiello@uniroma1.it (Danilo Comminiello), ahu@cs.stir.ac.uk (Amir Hussain), aurelio.uncini@uniroma1.it (Aurelio Uncini)

evaluation, in comparison with classical weight decay and Lasso penalties, both on a toy dataset for handwritten digit recognition, and multiple realistic mid-scale classification benchmarks. Comparative results demonstrate the potential of our proposed sparse group Lasso penalty in producing extremely compact networks, with a significantly lower number of input features, with a classification accuracy which is equal or only slightly inferior to standard regularization terms.

Keywords: Deep networks, Group sparsity, Pruning, Feature selection

---

1. Introduction

Recent growing interest in deep learning has made it feasible to train very deep (and large) neural networks, leading to remarkable accuracies in many high-dimensional problems including image recognition, video tagging, biomedical diagnosis, and others [1, 2, 3, 4]. While even five hidden layers were considered challenging until very recently, today simple techniques such as the inclusion of interlayer connections [5] and dropout [6] allow to train networks with hundreds (or thousands) of hidden layers, amounting to millions (or billions) of adaptable parameters. At the same time, it becomes extremely common to 'overpower' the network, by providing it with more flexibility and complexity than strictly required by the data at hand. Arguments that favor simple models instead of complex models for describing a phenomenon are quite known in the machine learning literature [7]. However, this is actually far from being just a philosophical problem of 'choosing the simplest model'. Having too many weights in a network can clearly increase the risk of overfitting; in addition, their exchange is the main bottleneck in

most parallel implementations of gradient descent, where agents must forward them to a centralized parameter server [8, 9]; and finally, the resulting models might not work on low-power or embedded devices due to excessive computational power needed for performing dense, large matrix-matrix multiplications [10].

In practice, current evidence points to the fact that the majority of weights in most deep networks are not necessary to its accuracy. As a representative example, Denil *et al.* [11] demonstrated that it is possible to learn only a small percentage of the weights, while the others can be predicted using a kernel-based estimator, resulting in most cases in a negligible drop in terms of classification accuracy. Similarly, in some cases it is possible to replace the original weight matrix with a low-rank approximation, and perform gradient descent on the factor matrices [12]. Driven by these observations, recently the number of works trying to reduce the network's weights have increased drastically (some of these works are reviewed more in depth in Section 5). Most of them either require strong assumptions on the connectivity (e.g, the low-rank assumption), multiple training steps, e.g. [13], or entirely separate optimization problems, e.g. [14].

When considering high-dimensional datasets, an additional problem is that of feature selection, where we search for a small subset of input features that brings most of the discriminative information [19]. Feature selection and pruning are related problems: adding a new set of features to a task generally results in the need of increasing the network's capacity (in terms of number of neurons), all the way up to the last hidden layer. Similarly to before, there are countless techniques for feature selection (or, in alternative, dimensional-

ity reduction of the input vector via linear or nonlinear mappings), including principal component analysis, mutual information [20], autoencoders, and many others. What we obtain, however, is a rather complex workflow of machine learning primitives: one algorithm to select features; an optimization criterion for training the network; and possibly another procedure to compress the weight matrices. This raises the following question, which is the main motivation for this paper: is there a principled way of performing all three tasks *simultaneously*, by minimizing a properly defined cost function? This is further motivated by the fact that, in a neural network, pruning a node and deleting an input feature are almost equivalent problems. In fact, it is customary to consider the input vector as an additional layer of the neural network, having no ingoing connections and having outgoing connections to the first hidden layer. In this sense, pruning a neuron from this initial layer can be considered the same as deleting the corresponding input feature.

Currently, the only principled way to achieve this objective is the use of ℓ_1 regularization, wherein we penalize the sum of absolute values of the weights during training. The ℓ_1 norm acts as a convex proxy of the non-convex, non-differentiable ℓ_0 norm [21]. Its use originated in the linear regression routine, where it is called the Lasso estimator, and it has been widely popularized recently thanks to the interest in compressive sensing [22, 23]. Even if it has a non differentiable point in 0, in practice this rarely causes problems to standard first-order optimizers. In fact, it is common to simultaneously impose both weight-level sparsity with the ℓ_1 norm, and weight minimization using the ℓ_2 norm, resulting in the so-called 'elastic net' penalization [24]. Despite its popularity, however, the ℓ_1 norm is only an indirect way of solving

the previously mentioned problems: a neuron can be removed if, and only if, all its ingoing or outgoing connections have been set to 0. In a sense, this is highly sub-optimal: between two equally sparse networks, we would prefer one which has a more *structured* level of sparsity, i.e. with a smaller number of neurons per layer.

In this paper, we show how a simple modification of the Lasso penalty, called the 'group Lasso' penalty in the linear regression literature [25, 26], can be used efficiently to this end. A group Lasso formulation can be used to impose sparsity on a group level, such that all the variables in a group are either simultaneously set to 0, or none of them are. An additional variation, called the sparse group Lasso, can also be used to impose further sparsity on the non-sparse groups [27, 28]. Here, we apply these ideas by considering all the outgoing weights from a neuron as a single group. In this way, the optimization algorithm is able to remove entire neurons at a time. Depending on the specific neuron, we obtain different effects, corresponding to what we discussed before: feature selection when removing an input neuron; pruning when removing an internal neuron; and also bias selection when considering a bias unit (see next section). The idea of group ℓ_1 regularization in machine learning is quite known when considering convex loss functions [29], including multikernel [30] and multitask problems [31]. However, to the best of our knowledge, such a general formulation was never considered in the neural networks literature, except for very specific cases. For example, Zhao *et al.* [32] used a group sparse penalty to select groups of features co-occurring in a robotic control task. Similarly, Zhu *et al.* [33] have used a group sparse formulation to select informative groups of features in a multi-modal context.

Liu *et al.* [34] apply a similar formulation to the specific case of convolutional networks.

On the contrary, in this paper we employ the group Lasso formulation as a generic tool for enforcing compact networks with a lower subset of selected features. Our experimental results show that best results are obtained using the sparse group term, with comparable accuracy to ℓ_2-regularized and ℓ_1-regularized networks, while simultaneously reducing, by a large margin, the number of neurons in every layer. In addition, the regularizer can be readily implemented in most existing software libraries, and it does not increase the computational complexity with respect to the traditional weight decay technique.

Outline of the paper

The paper is organized as follows. Section 2 describes standard techniques for regularizing a neural network during training, namely ℓ_2, ℓ_1 and composite ℓ_2/ℓ_1 terms. Section 3 describes our novel group Lasso and sparse group Lasso penalties, introducing the concept of groups in this context. Next, we evaluate our algorithms in Section 4 on a simple toy dataset of handwritten digits recognition, followed by multiple realistic experiments with standard deep learning benchmarks. Section 5 presents a further review of related pruning techniques, followed by some concluding remarks and future work proposals in Section 6.

Notation

In the rest of the paper, vectors are denoted by boldface lowercase letters, e.g. $\mathbf{a}$, while matrices are denoted by boldface uppercase letters, e.g. $\mathbf{A}$. All

vectors are assumed column vectors. The operator $\|\cdot\|_p$ is the standard ℓ_p norm on an Euclidean space. For $p = 2$ this is the Euclidean norm, while for $p = 1$ we obtain the Manhattan (or taxicab) norm defined for a generic vector $\boldsymbol{\beta} \in \mathbb{R}^B$ as $\|\boldsymbol{\beta}\|_1 = \sum_{k=1}^{B} |\beta_k|$.

2. Weight-level regularization for neural networks: overview of conventional approaches

Let us denote by $\mathbf{y} = \mathbf{f}(\mathbf{x}; \mathbf{w})$ a generic deep neural network, taking as input a vector $\mathbf{x} \in \mathbb{R}^d$, and returning a vector $\mathbf{y} \in \mathbb{R}^o$ after propagating it through H hidden layers. The vector $\mathbf{w} \in \mathbb{R}^Q$ is used as a shorthand for the column-vector concatenation of all adaptable parameters of the network. The generic kth hidden layer, $1 \leq k \leq H + 1$, operates on a L_k-dimensional input vector $\mathbf{h}_k$ and returns an L_{k+1}-dimensional output vector $\mathbf{h}_{k+1}$ as:

$$\mathbf{h}_{k+1} = g_k \left(\mathbf{W}_k \mathbf{h}_k + \mathbf{b}_k \right) , \tag{1}$$

where $\{\mathbf{W}_k, \mathbf{b}_k\}$ are the adaptable parameters of the layer, while $g_k(\cdot)$ is a properly chosen activation function to be applied element-wise. By convention we have $\mathbf{h}_1 = \mathbf{x}$. For training the weights of the network, consider a generic training set of N examples given by $\{(\mathbf{x}_1, \mathbf{d}_1), \ldots, (\mathbf{x}_N, \mathbf{d}_N)\}$. The network is trained by minimizing a standard regularized cost function:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \left\{ \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{d}_i, \mathbf{f}(\mathbf{x}_i; \mathbf{w})) + \lambda R(\mathbf{w}) \right\} , \tag{2}$$

where $L(\cdot, \cdot)$ is a proper cost function, $R(\cdot)$ is used to impose regularization, and the scalar coefficient $\lambda \in \mathbb{R}^+$ weights the two terms. Standard choices for

$L(\cdot,\cdot)$ are the squared error for regression problems, and the cross-entropy loss for classification problems [35].

By far the most common choice for regularizing the network, thus avoiding overfitting, is to impose a squared ℓ_2 norm constraint on the weights:

$$R_{\ell_2}(\mathbf{w}) \triangleq \|\mathbf{w}\|_2^2 \,. \tag{3}$$

In the neural networks' literature, this is commonly denoted as 'weight decay' [36], since in a steepest descent approach, its net effect is to reduce the weights by a factor proportional to their magnitude at every iteration. Sometimes it is also denoted as Tikhonov regularization. However, the only way to enforce sparsity with weight decay is to artificially force to zero all weights that are lower, in absolute terms, than a certain threshold. Even in this way, its sparsity effect might be negligible.

As we stated in the introduction, the second most common approach to regularize the network, inspired by the Lasso algorithm, is to penalize the absolute magnitude of the weights:

$$R_{\ell_1}(\mathbf{w}) \triangleq \|\mathbf{w}\|_1 = \sum_{k=1}^{Q} |w_k| \,. \tag{4}$$

The ℓ_1 formulation is not differentiable at 0, where it is necessary to resort to a subgradient formulation. Everywhere else, its gradient is constant, and in a standard minimization procedure it moves each weight by a constant factor towards zero (in the next section, we also provide a simple geometrical intuition on its behavior). While there exists customized algorithms to solve non-convex problems with ℓ_1 regularization [37], it is common in the neural

networks' literature to apply directly the same first-order procedures (e.g., stochastic descent with momentum) as for the weight decay formulation. As an example, all libraries built on top of the popular Theano framework [38] assigns a default gradient value of 0 to terms such that $w_k = 0$. Due to this, a thresholding step after optimization is generally required also in this case to obtain precisely sparse solutions [39], although the resulting level of sparsity is quite higher than using weight decay.

One popular variation is to approximate the ℓ_1 norm by a convex term, e.g. $\|\mathbf{w}\|_1 \approx \sum_{k=1}^{Q} \sqrt{w_k^2 + \beta}$ for a sufficiently small scalar factor β, to obtain a smooth problem. Another possibility is to consider a mixture of ℓ_2 and ℓ_1 regularization, which is sometimes denoted as elastic net penalization [24]. The problem in this case, however, is that it is required to select two different hyper-parameters for weighting differently the two terms.

3. Proposed neuron-level regularization with group sparsity

3.1. Formulation of the algorithm

Both ℓ_2 regularization in (3) and ℓ_1 regularization in (4) are efficient for preventing overfitting, but they are not optimal for obtaining compact networks. Generally speaking, a neuron can be removed from the architecture only if all its connections (either ingoing or outgoing) have been zeroed out during training. However, this objective is not actively pursued while minimizing the cost in (2). Between the many local minima, some might be equivalent (or almost equivalent) in terms of accuracy, while corresponding to more compact and efficient networks. As there is no principled way to converge to one instead of the other, when using these kind of regulariza-

tions the resulting network's design will simply be a matter of initialization of the optimization procedure.

The basic idea of this paper is to consider *group-level* sparsity, in order to force all outgoing connections from a single neuron (corresponding to a group) to be either simultaneously zero, or not. More specifically, we consider three different groups of variables, corresponding to three different effects of the group-level sparsity:

1. **Input groups** $\mathcal{G}_{\text{in}}$: a single element $\mathbf{g_i} \in \mathcal{G}_{\text{in}}, i = 1, \ldots, d$ is the vector of all outgoing connections from the ith input neuron to the network, i.e. it corresponds to the first row transposed of the matrix $\mathbf{W}_1$.
2. **Hidden groups** $\mathcal{G}_{\text{h}}$: in this case, a single element $\mathbf{g} \in \mathcal{G}_{\text{h}}$ corresponds to the vector of all outgoing connections from one of the neurons in the hidden layers of the network, i.e. one row (transposed) of a matrix $\mathbf{W}_k, k > 1$. There are $\sum_{k=2}^{H+1} N_k$ such groups, corresponding to neurons in the internal layers up to the final output one.
3. **Bias groups** $\mathcal{G}_{\text{b}}$: these are one-dimensional groups (scalars) corresponding to the biases on the network, of which there are $\sum_{k=1}^{H+1} N_k$. They correspond to a single element of the vectors $\{\mathbf{b}_1, \ldots, \mathbf{b}_{H+1}\}$.

Overall, we have a total of $G = 2\sum_{k=1}^{H+1} N_k$ groups, corresponding to three specific effects on the resulting network. If the variables of an input group are set to zero, the corresponding feature can be neglected during the prediction phase, effectively corresponding to a feature selection procedure. Then, if the variables in a hidden group are set to zero, we can remove the corresponding neuron, thereby obtaining a pruning effect and a thinner hidden layer. Finally, if a variable in a bias group is set to zero, we can remove the

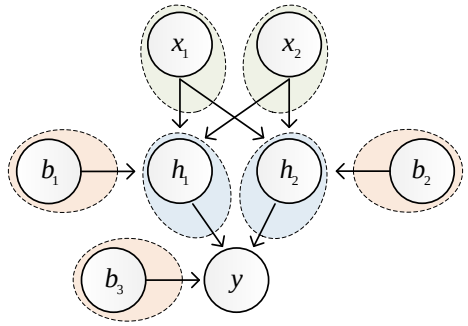


Figure 1: Schematic representation of a group Lasso regularization with two inputs (top), two hidden neurons with biases (middle), and one output (bottom). We have three groups of connections. **Green**: input groups; **blue**: hidden groups; **red**: bias groups.

corresponding bias from the neuron. We note that having a separate group for every bias is not the unique choice. We can consider having a single bias unit for every layer feeding every neuron in that layer. In this case, we would have a single bias group per layer, corresponding to keeping or deleting every bias in it. Generally speaking, we have not found significant improvements in one way or the other.

A visual representation of this weight grouping strategy is shown in Fig. 1 for a simple network with two inputs (top of the figure), one hidden layer with two units (middle of the figure), and a single output unit (bottom of the figure). In the figure, input groups are shown with a green background; hidden groups (which in this case have a single element per group) are shown with a blue background; while the 3 bias groups are surrounded in a light red background.

Let us define for simplicity the total set of groups as

$$\mathcal{G} = \mathcal{G}_{\text{in}} \cup \mathcal{G}_{\text{h}} \cup \mathcal{G}_{\text{b}} \,.$$

Group sparse regularization can be written as [25]:

$$R_{\ell_{2,1}}(\mathbf{w}) \triangleq \sum_{\mathbf{g}\in\mathcal{G}} \sqrt{|\mathbf{g}|}\, \|\mathbf{g}\|_2 \ , \tag{5}$$

where $|\mathbf{g}|$ denotes the dimensionality of the vector $\mathbf{g}$, and it ensures that each group gets weighted uniformly. Note that, for one-dimensional groups, the expression in (5) simplifies to the standard Lasso. Similarly to the ℓ_1 norm, the term in (5) is convex but non-smooth, since its gradient is not defined if $\|\mathbf{g}\|_2 = 0$. The sub-gradient of a single term in (5) is given by:

$$\frac{\partial \left\{ \sqrt{|\mathbf{g}|}\, \|\mathbf{g}\|_2 \right\}}{\partial \mathbf{g}} = \begin{cases} \sqrt{|\mathbf{g}|} \dfrac{\mathbf{g}}{\|\mathbf{g}\|_2} & \text{if } \mathbf{g} \neq \mathbf{0} \\ \left\{ \sqrt{|\mathbf{g}|}\mathbf{t} : \|\mathbf{t}\|_2 \leq 1 \right\} & \text{otherwise} \end{cases} . \tag{6}$$

As for the ℓ_1 norm, we have found very good convergence behaviors using standard first-order optimizers, with a default choice of $\mathbf{0}$ as sub-gradient in the second case. Also here, a final thresholding step is required to obtain precisely sparse solutions. Note that we have used the $\ell_{2,1}$ symbol in (5) as the formulation is closely related to the $\ell_{2,1}$ norm defined for matrices.

The formulation in (5) might still be sub-optimal, however, since we lose guarantees of sparsity at the level of single connections among those remaining after removing some of the groups. To force this, we also consider the following composite 'sparse group Lasso' (SGL) penalty [27, 28]:

$$R_{\text{SGL}}(\mathbf{w}) \triangleq R_{\ell_{2,1}}(\mathbf{w}) + R_{\ell_1}(\mathbf{w}) \, . \tag{7}$$

The SGL penalty has the same properties as its constituting norms, namely, it

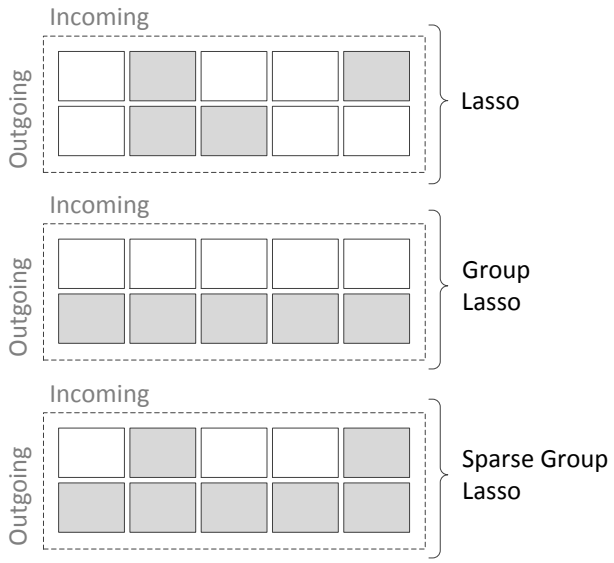
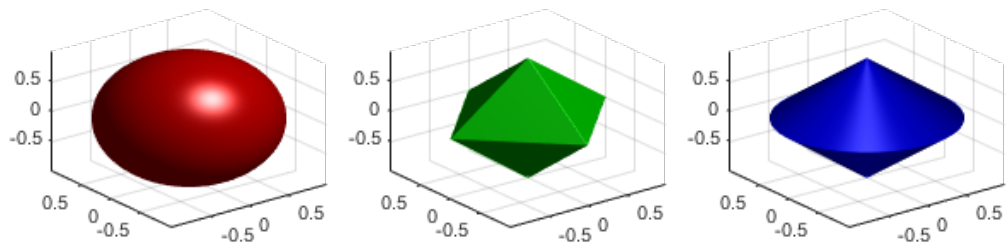


Figure 2: Comparison between Lasso, group Lasso, and sparse group Lasso applied to a single weight matrix. In gray we represent the removed connections.

is convex but non-differentiable. Differently from an elastic net penalization, we have found that optimal results can be achieved by considering a single regularization factor for both terms in (7). It is interesting to note that the computational complexity of the different regularization terms (i.e., $\ell_2, \ell_1, \ell_{2,1}$ and SGL) is equivalent and it is given by $\mathcal{O}(Q)$, where Q is the number of NN's parameters.

A visual comparison between ℓ_1, $\ell_{2,1}$, and SGL penalizations is given in Fig. 2. The dashed box represents one weight matrix connecting a 2-dimensional input layer to a 5-dimensional output layer. In gray, we show a possible combination of matrix elements that are zeroed out by the corresponding penalization. The Lasso penalty removes elements without optimizing neuron-level considerations. In this example, we remove 4 connections (thus obtaining a 40% level of sparsity), and we might remove the second



(a) ℓ_2 norm (b) ℓ_1 norm (c) $\ell_{2,1}$ norm

Figure 3: Isosurface for three different regularization terms, with $\mu_\lambda = 1$. (a) Standard squared ℓ_2 norm. (b) ℓ_1 norm enforcing sparsity. (c) $\ell_{2,1}$ norm applied to the groups $\{1, 2\}$ and $\{3\}$ (without considering the scaling factors).

neuron from the second layer (only if the bias unit to the neuron has also been deleted). The group Lasso penalization removes all connections exiting from the second neuron, which can now be safely removed from the network. The sparsity level is just slightly higher than in the first case, but the resulting connectivity is more structured. Finally, the SGL formulation combines the advantages of both formulation: we remove all connections from the second neuron in the first layer *and* two of the remaining connections, thus achieving a 70% level of sparsity in the layer and an extremely compact (and power-efficient) network.

3.2. Graphical interpretation of group sparsity

The group Lasso penalty admits a very interesting geometrical interpretation whenever the first term in (2) is convex (see for example [23, Section 1]). Although this is not the case of neural networks, whose model is highly non-convex due to the presence of the hidden layers, this interpretation does help in visualizing why the resulting formulation provides a group sparse solution. For this reason, we briefly describe it here for the sake of understanding.

For a convex loss in (2), standard arguments from duality theory show that the problem can be reformulated as follows [40]:

$$\begin{aligned} \arg\min_{\mathbf{w}} \quad & L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{d}_i, \mathbf{f}(\mathbf{x}_i; \mathbf{w})) \\ \text{subject to} \quad & R(\mathbf{w}) \leq \mu_\lambda \end{aligned} \tag{8}$$

where μ_λ is a scalar whose precise value depends on λ, and whose existence is guaranteed thanks to the absence of duality gap. In machine learning, this is sometimes called Ivanov regularization, in honor of the Russian mathematician Nikolai V. Ivanov [41]. For a small value of μ_λ, such that the constraint in (8) is active at the optimum $\mathbf{w}^*$, it can be shown that the set of points for which $L(\mathbf{w})$ is equal to $L(\mathbf{w}^*)$ is tangent to $\mathcal{B} = \{\mathbf{w} : R(\mathbf{w}) \leq \mu_\lambda\}$. Due to this, an empirical way to visualize the behavior of the different penalties is to consider the shape of $\mathcal{B}$ corresponding to them. The shapes corresponding to ℓ_2 regularization, Lasso, and group Lasso are shown in Fig. 3 for a simple problem with three variables. The shape of $\mathcal{B}$ for a weight decay penalty is a sphere (shown in Fig. 3a), which does not favor any of the solutions. On the contrary, the Lasso penalty imposes a three-dimensional diamond-shaped surface (shown in Fig. 3b), whose vertices lie on the axes and correspond to all the possible combinations of sparse solutions. Finally, consider the shape imposed by the group Lasso penalty (shown in Fig. 3c), where we set one group composed of the first two variables, and another group composed of the third variable. The shape now has infinitely many singular points, corresponding to solutions having zeroes either on the first and second variables simultaneously, or in the third variable.

4. Experimental results

4.1. Experimental setup

In this section, we evaluate our proposal on different classification benchmarks. Particularly, we begin with a simple toy dataset to illustrate its general behavior, and then move on to more elaborate, real-world datasets. In all cases, we use ReLu activation functions [42] for the hidden layers of the network:

$$g_k(s) = \max(0, s)\,,\ 1 \leq k \leq H\,, \tag{9}$$

while we use the standard one-hot encoding for the different classes, and a softmax activation function for the output layer. Denoting as $\mathbf{s}$ the values in input to the softmax, its ith output is computed as:

$$g_{H+1}(s_i) = \frac{\exp\{s_i\}}{\sum_{j=1}^{o} \exp\{s_j\}}\,. \tag{10}$$

The weights of the network are initialized according to the method described in [43], and the networks are trained using the popular Adam algorithm [44], a derivation of stochastic gradient descent with both adaptive step sizes and momentum. In all cases, parameters of the Adam procedure are kept as the default values described in [44], while the size of the mini-batches is varied depending on the dimensionality of the problem. Specifically, we minimize the loss function in (2) with the standard cross-entropy loss given by:

$$L(\mathbf{d}, \mathbf{f}(\mathbf{x}; \mathbf{w})) = -\sum_{i=1}^{o} d_i \log\left(f_i(\mathbf{x}; \mathbf{w})\right)\,, \tag{11}$$

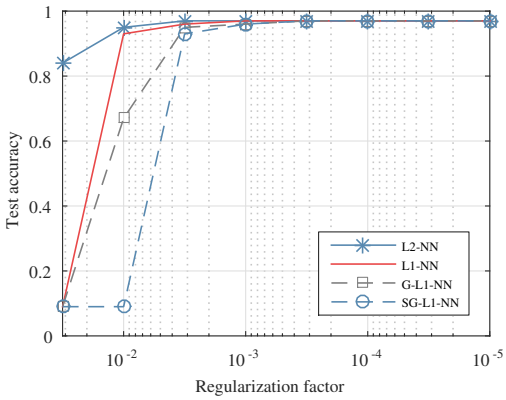
and multiple choices for the regularization penalty. Dataset loading, preprocessing and splitting is made with the sklearn library [45]. First, every input column is normalized in the range $[0, 1]$ with an affine transformation. Then for every run we randomly keep 25% of the dataset for testing, and we repeat each experiment 25 times in order to average out statistical variations. For training, we exploit the Lasagne framework,[1] which is built on top of the Theano library [38]. Open source code to replicate the experiments is available on the web under BSD-2 license.[2]

4.2. Comparisons with the DIGITS dataset

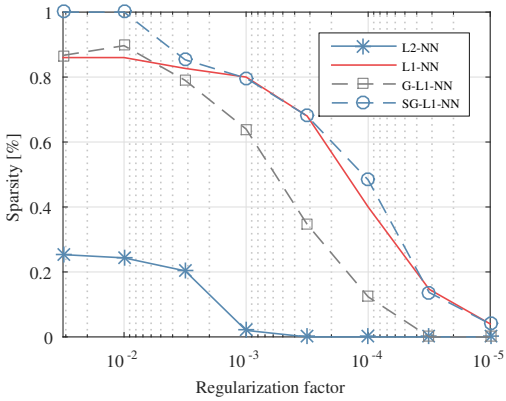
To begin with, we evaluate our algorithm on a toy dataset of handwritten digit recognition, namely the DIGITS dataset [46]. It is composed of 1797 8×8 grey images of handwritten digits collected from several dozens different people. We compare four neural networks, trained respectively with the weight decay in (3) (denoted as L2-NN), the Lasso penalty in (4) (denoted as L1-NN), the proposed group Lasso penalty in (5) (denoted as G-L1-NN), and finally its sparse variation in (7) (denoted as SG-L1-NN). In all cases, we use a simple network with two hidden layers having, respectively, 40 and 20 neurons. We run the optimization algorithm for 200 epochs, with mini-batches of 300 elements. After training, all weights under 10^{-3} in absolute value are set to 0.

The aim of this preliminary test is to evaluate what we obtain from the different penalties when varying the regularization factor λ. To this end, we run each algorithm by choosing λ in the exponential range 10^{-j}, with j

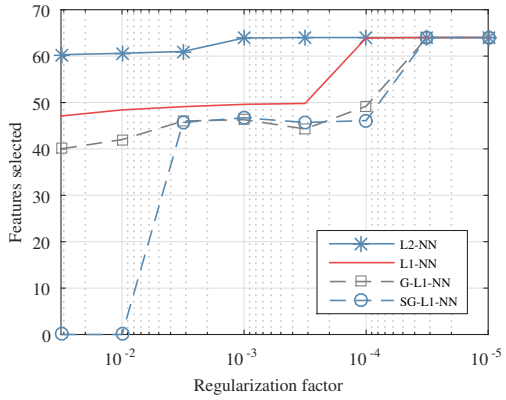
[1] `https://github.com/Lasagne/Lasagne`
[2] `https://bitbucket.org/ispamm/group-lasso-deep-networks`



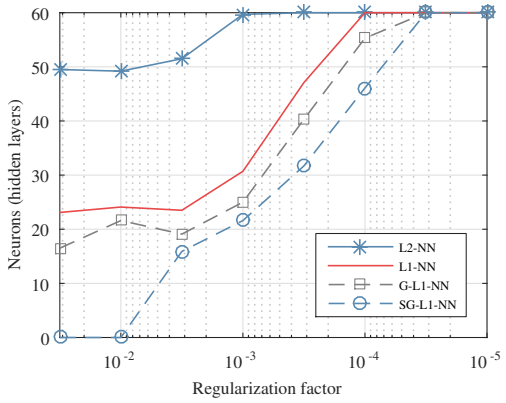
(a) Test accuracy



(b) Sparsity

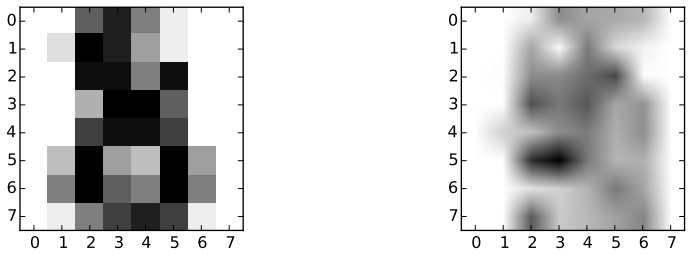


(c) Features selected



(d) Neurons (hidden layers)

Figure 4: Results for the digits dataset, when varying the regularization coefficient in 10^{-j}, $j = 1, \ldots, 5$. (a) Test accuracy. (b) Sparsity of the internal connections (in percentage). (c) Number of selected input features. (d) Number of neurons in the hidden layers (total).



(a) Example of digit (b) Selected features

Figure 5: Visualization of the selected features for the digits dataset. (a) Example of input pattern to the network (number 8). (b) Overall strength of outgoing weights from the respective input pixel (white are lowest, black are highest).

going from 1 to 5. Results of this set of experiments are shown in Fig. 4. There are several key observations to be made from the results. To begin with, the overall behavior in terms of test accuracy with respect to the four penalties, shown in Fig. 4a, is similar among the algorithms, as they rapidly converge to the optimal accuracy (slightly lower than 100%) for sufficiently small regularization factors. In particular, from 10^{-3} onwards, their results are basically indistinguishable. Fig. 4b shows the level of sparsity that we obtain, which is evaluated as the percentage of zero weights with respect to the total number of connections. The sparsity of L2-NN is clearly unsatisfactory, oscillating from 20% in the best case to 0% in average. The sparsity of G-L1-NN is lower than the corresponding sparsity of L1-NN, while the results of SG-L1-NN (shown with a dashed blue line) are equal or superior than all alternatives. In particular, for $\lambda = 10^{-3}$ both L1-NN and SG-L1-NN are able to remove four fifths of the connections. At the same time, the resulting sparsity is significantly more structured for the proposed algorithm, which is able to consistently remove more features, as shown in Fig. 4c, and

Table 1: Schematic description of the datasets.

Origin	Name	Features	Size	N. Classes	Desired output	Ref.
UCI	Sensorless Drive Diagnosis (SDD)	48	58508	11	Motor condition	[47]
MLData	MNIST Handwritten Digits	784	70000	10	Digit (0-9)	[48]
MLData	Forest Covertypes (COVER)	54	581012	7	Cover type	[49]

neurons in the hidden layers, as shown in Fig. 4d.

Since the input to the classifier is an image, it is quite interesting to visualize which features (corresponding to pixels of the original image) are neglected in the proposed approaches, in order to further validate empirically the proposal. This is shown for one representative run in Fig. 5. In Fig. 5a we see a characteristic image in input to the system, representing in this case the number 8. We see that the digit covers all the image with respect to its height, while there is some white space to its left and right, which is not interesting from a discriminative point of view. In Fig. 5b we visualize the results of G-L1-NN (which is very similar to SG-L1-NN), by plotting the cumulative intensity of the weights connecting the input layer to the first hidden layer (where white color represents an input with all outgoing connections set to 0). We see that the algorithm does what we would have expected in this case, by ignoring all pixels corresponding to the outermost left and right regions of the image.

4.3. Comparisons with mid-scale datasets

We now evaluate our algorithm on three more realistic datasets, which require the use of deeper, larger networks. A schematic description of them is given in Table 1 in terms of features, number of patterns, and number of

Table 2: Parameters for the neural networks used in the experiments.

Dataset	Neurons	Regularization	Mini-batch size
SSD	40/40/30/11	10^{-4}	500
MNIST	400/300/100/10	10^{-4}	400
COVER	50/50/20/7	10^{-4}	1000

output classes. The first is downloaded from the UCI repository,[3] while the second and third ones are downloaded from the MLData repository.[4] In the SSD dataset, we wish to predict whether a motor has one or more defective components, starting from a set of 48 features obtained from the motor's electric drive signals (see [47] for details on the feature extraction process). The dataset is composed of 58508 examples obtained under 11 different operating conditions. The MNIST database is an extremely well-known database of handwritten digit recognition [48], composed of 70 thousands 28×28 gray images of the digits 0-9. Finally, the COVER dataset is the task of predicting the actual cover type of a forest (e.g. ponderosa pine) from a set of 52 features extracted from cartographic data (see [49, Table 1] for a complete list of cover types). This dataset has roughly a half million training examples, but only 7 possible classes compared to 11 and 10 classes for SSD and MNIST, respectively.

Details on the network's architecture, regularization factor and mini-batch size for the three datasets are given in Table 2. Generally speaking,

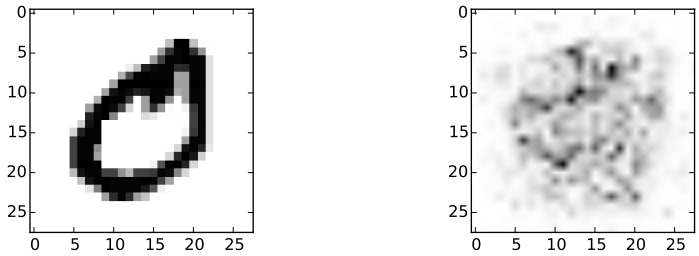
[3] `http://archive.ics.uci.edu/ml/`
[4] `http://mldata.org/`

we use the same regularization factor for all algorithms, as it was shown to provide the best results in terms of classification accuracy and sparsity of the network. The network architecture is selected based on an analysis of previous works and is given in the second column of Table 2, where $x/y/z$ means a network with one x-dimensional hidden layer, a second y-dimensional hidden layer, and a z-dimensional output layer. We stress that our focus is on comparing the different penalties, and very similar results can be obtained for different choices of the network's architecture and the regularization factors. Additionally, we only consider SG-L1-NN as the previous section has shown that it can consistently outperform the simpler G-L1-NN.

The results for these experiments are given in Table 3, where we show the average training and test accuracy, training time, sparsity of the network, and final size of each hidden layer (which is highlighted with a light blue background, and it is averaged throughout the different repetitions). As a note on training times, results for the smaller SSD dataset are obtained on an Intel Core i3 @ 3.07 GHz with 4 GB of RAM, while results for MNIST and COVER are obtained on an Intel Xeon E5-2620 @ 2.10 GHz, with 8 GB of RAM and a CUDA back-end employing an Nvidia Tesla K20c. For the number of neurons, we do not show the size of the final softmax as this is not influenced by our technique. We see that the results in terms of test accuracy are comparable between the three algorithms, with a negligible loss on the MNIST dataset for SG-L1-NN. However, SG-L1-NN results in networks which are extremely sparse and more compact than its two competitors. Let us consider as an example the MNIST dataset. In this case, the algorithm removes more than 200 features in average from the input vector (compared

Table 3: Average results on the datasets of Table 1. With a light blue background we highlight the (average) resulting size of the network's layers, including the input one while excluding the softmax one. Times with a † symbol were obtained without using the CUDA backend (see text).

Dataset	Measure	L2-NN	L1-NN	SG-L1-NN
SSD	Training accuracy [%]	0.98	0.99	0.97
	Test accuracy [%]	0.98	0.98	0.97
	Training time [secs.]	$445^{\dagger}$	$496^{\dagger}$	$416^{\dagger}$
	Sparsity [%]	$[0.17, 0.36, 0.36, 0.16]$	$[0.51, 0.64, 0.61, 0.43]$	$[0.64, 0.81, 0.76, 0.54]$
	Neurons	$[48.0, 35.5, 24.8, 26.3]$	$[47.9, 27.5, 19.6, 20.2]$	$[47.4, 19.0, 14.8, 15.9]$
MNIST	Training accuracy [%]	0.99	0.99	0.98
	Test accuracy [%]	0.98	0.97	0.97
	Training time [secs.]	81	83	93
	Sparsity [%]	$[0.60, 0.60, 0.34, 0.08]$	$[0.91, 0.98, 0.94, 0.44]$	$[0.96, \approx 1.0, 0.98, 0.48]$
	Neurons	$[676.4, 311, 249.9, 93.7]$	$[658.2, 84.8, 85.1, 73.3]$	$[581.8, 44.7, 41.0, 60.6]$
COVER	Training accuracy [%]	0.85	0.84	0.83
	Test accuracy [%]	0.84	0.83	0.83
	Training time [%]	454	479	551
	Sparsity [%]	$[0.04, 0.10, 0.22, 0.14]$	$[0.19, 0.48, 0.61, 0.34]$	$[0.45, 0.82, 0.84, 0.49]$
	Neurons	$[54.0, 49.0, 47.3, 18.7]$	$[53.0, 46.0, 31.0, 14.3]$	$[52.7, 30.0, 16.0, 11.3]$



(a) Example of digit (b) Selected features

Figure 6: Visualization of the selected features for the MNIST dataset. (a) Example of input pattern to the network (number 0). (b) Overall strength of outgoing weights from the respective input pixel (white are lowest, black are highest).

to approximately 126 for L1-NN). Also, the resulting network only has 146 neurons in the hidden layers compared to 243 for L1-NN and 654 for L2-NN. Also in this case, we can visually inspect the resulting features selected by the algorithm, which are shown in Fig. 6. In Fig. 6a we see an example of input pattern (corresponding to the digit 0), while in Fig. 6b we plot the cumulative intensity of the outgoing weights from the input layer. Differently from the DIGITS dataset, the images in this case have a large white margin on all sides, which is efficiently neglected by the proposed formulation, as shown by the white portions of Fig. 6b.

One last observation can be made regarding the required training time. The SGL penalty is actually faster to compute than both the ℓ_2 and ℓ_1 norms when the code is run on the CPU, while we obtain a slower training time (albeit by a small margin) when it is executed on the CUDA back-end. The reason for this is the need to compute two square root operations per group in (7). This gap can be removed by exploiting several options for faster mathematical computations (at the cost of precision) on the GPU, e.g. by

using the '–prec-sqrt' flag on the Nvidia CUDA compiler.

Overall, the results presented in this section show how the sparse group Lasso penalty can easily allow to obtain networks with a high level of sparsity, a low number of neurons (both on the input layer and on the hidden layers), while incurring no or negligible losses in accuracy.

5. Related works

Before concluding our paper, we describe a few related works that we briefly mentioned in the introduction, in order to highlight some common points and differences. Recently, there has been a sustained interest in methods that randomly decrease the complexity of the network during training. For example, dropout [6] randomly removes a set of connections; stochastic depth skips entire layers [50]; while [42] introduced the possibility of applying the ℓ_1 penalty to the activations of the neurons in order to further sparsify its firing patterns. However, these methods are only used to simplify the training phase, while the entire network is needed at the prediction stage. Thus, they are only tangentially related to what we discuss here.

A second class of related works group all the so-called pruning methods, which can be used to simplify the network's structure *after* training is completed. Historically, the most common method to achieve this is the optimal brain damage algorithm introduced by LeCun [16], which removes connections by measuring a 'saliency' quantity related to the second-order derivatives of the cost function at the optimum. Other methods require instead to compute the sensitivity of the error to the removal of each neuron, in order to choose an optimal subset of neurons to be deleted [51]. More recently, a

two-step learning process introduced by Han *et al.* [13] has also gained a lot of popularity. In this method, the network is originally trained considering an ℓ_2 penalty, in order to learn which connections are 'important'. Then, the non-important connections, namely all weights under a given threshold, are set to zero, and the network is retrained by keeping fixed the zeroed out weights. This procedure can also be iteratively repeated to further reduce the size of the network. None of these methods, however, satisfy what we considered in the introduction, i.e. they either require a separate pruning process, they do not act directly at the neuron-level, and they might make some heuristic assumptions that should hold at the pruning phase. As an example, the optimal brain damage algorithm is built on the so-called diagonal approximation, stating that the error modification resulting from modifying many weights can be computed by summing the individual contributions from each weight perturbation.

A third class of methods is not interested in learning an optimal topology, insofar as to reduce the actual number of parameters and/or the storage requirements of the network. The most common method in this class is the low-rank approximation method [12], where a weight matrix $\mathbf{W}_k \in \mathbb{R}^{L_k \times L_{k+1}}$ is replaced by a low-rank factorization $\mathbf{W}_k = \mathbf{AB}$, $\mathbf{A} \in \mathbb{R}^{L_k \times r}, \mathbf{B} \in \mathbb{R}^{r \times L_{k+1}}$, where the rank r must be chosen by the user. Optimization is then performed directly on the two factors instead of the original matrix. The choice of the rank allows to balance between compression and accuracy. As an example, if we wish to compress the network by a factor p, we can choose [12]:

$$r < \frac{pL_k L_{k+1}}{L_k + L_{k+1}} \,. \tag{12}$$

However, this approximation is not guaranteed to work efficiently, and may result in highly worse results for a poor choice of r.

There are endless other possibilities, e.g. (i) we can use 'distillation' to train a separate, smaller network that imitates the original one, as popularized by Hinton *et al.* [15]; (ii) we can work with limited numerical precision to reduce storage [17] (up to the extreme of a single bit per weight [10]); (iii) we can use hash functions to force weight sharing [18]; and so on.

6. Conclusions

In this paper, we have introduced a new simple and efficient approach for simultaneously carrying out pruning and feature selection whilst optimizing the weights of a neural network. Our proposed sparse group Lasso penalty can be implemented efficiently and easily in most software libraries, with an equivalent overhead with respect to standard ℓ_2 or ℓ_1 formulations. Comparative experimental results also demonstrate its superior performance for producing highly compact networks, with definite savings in terms of storage requirements and power consumption in embedded devices.

Future research will explore a number of interesting areas. First, there is the open problem in the optimization literature, of studying the interaction between a sparse ℓ_1 formulation (originally proposed for the case of convex costs), with a non-convex cost as in (2). It would be interesting to investigate possible improvements with the use of a non-convex sparse regularizer, such as the ℓ_p norm with fractional p. Alternatively, one may attempt to improve the sparse behavior of (4) and (5) by iteratively solving a convex approximation to the original non-convex problem, e.g. by exploiting techniques

presented in [52], following our previous work on semi-supervised support vector machines [53]. Secondly, we are interested in exploring group Lasso formulations for other types of neural networks, such as deep convolutional neural networks and recurrent neural networks.

Finally, we are aiming to extend our previous work on ℓ_1 sparse regularization in reservoir computing architectures [54], which showed that sparse connectivity can help in creating clusters of neurons resulting in heterogeneous features extracted from the recurrent layer.

Acknowledgments

The authors wish to thank Dr. Paolo Di Lorenzo for his insightful comments.

References

[1] J. Schmidhuber, Deep learning in neural networks: An overview, Neural Networks 61 (2015) 85–117.

[2] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.

[3] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, M. S. Lew, Deep learning for visual understanding: A review, Neurocomputing 187 (2016) 27–48.

[4] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, F. E. Alsaadi, A survey of deep neural network architectures and their applications, Neurocomputing, in press.

[5] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, arXiv preprint arXiv:1512.03385.

[6] 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.

[7] P. Domingos, A few useful things to know about machine learning, Communications of the ACM 55 (10) (2012) 78–87.

[8] 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, 2011, pp. 693–701.

[9] F. Seide, H. Fu, J. Droppo, G. Li, D. Yu, On parallelizability of stochastic gradient descent for speech dnns, in: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2014, pp. 235–239.

[10] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in Neural Information Processing Systems, 2015, pp. 3123–3131.

[11] M. Denil, B. Shakibi, L. Dinh, N. de Freitas, Predicting parameters in deep learning, in: Advances in Neural Information Processing Systems, 2013, pp. 2148–2156.

[12] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, B. Ramabhadran, Low-rank matrix factorization for deep neural network training with

high-dimensional output targets, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2013, pp. 6655–6659.

[13] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.

[14] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, arXiv preprint arXiv:1412.6115.

[15] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv preprint arXiv:1503.02531.

[16] 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, 1989, pp. 598–605.

[17] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: Proceedings of The 32nd International Conference on Machine Learning, 2015, pp. 1737–1746.

[18] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: Proceedings of The 32nd International Conference on Machine Learning, 2015, pp. 2285–2294.

[19] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Journal of machine learning research 3 (Mar) (2003) 1157–1182.

[20] N. Kwak, C.-H. Choi, Input feature selection for classification problems, IEEE Transactions on Neural Networks 13 (1) (2002) 143–159.

[21] R. Tibshirani, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society. Series B (Methodological) (1996) 267–288.

[22] E. J. Candès, M. B. Wakin, An introduction to compressive sampling, IEEE Signal Processing Magazine 25 (2) (2008) 21–30.

[23] F. Bach, R. Jenatton, J. Mairal, G. Obozinski, Optimization with sparsity-inducing penalties, Foundations and Trends® in Machine Learning 4 (1) (2012) 1–106.

[24] H. Zou, T. Hastie, Regularization and variable selection via the elastic net, Journal of the Royal Statistical Society. Series B (Statistical Methodology) 67 (2) (2005) 301–320.

[25] 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.

[26] M. Schmidt, Graphical model structure learning with l1-regularization, Ph.D. thesis, University of British Columbia (Vancouver) (2010).

[27] J. Friedman, T. Hastie, R. Tibshirani, A note on the group lasso and a sparse group lasso, arXiv preprint arXiv:1001.0736.

[28] N. Simon, J. Friedman, T. Hastie, R. Tibshirani, A sparse-group lasso,

Journal of Computational and Graphical Statistics 22 (2) (2013) 231–245.

[29] R. Jenatton, J.-Y. Audibert, F. Bach, Structured variable selection with sparsity-inducing norms, Journal of Machine Learning Research 12 (Oct) (2011) 2777–2824.

[30] F. R. Bach, Consistency of the group lasso and multiple kernel learning, Journal of Machine Learning Research 9 (Jun) (2008) 1179–1225.

[31] J. Liu, S. Ji, J. Ye, Multi-task feature learning via efficient l 2, 1-norm minimization, in: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, AUAI Press, 2009, pp. 339–348.

[32] L. Zhao, Q. Hu, W. Wang, Heterogeneous feature selection with multimodal deep neural networks and sparse group lasso, IEEE Transactions on Multimedia 17 (11) (2015) 1936–1948.

[33] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, X. Xie, Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks, arXiv preprint arXiv:1603.07772.

[34] 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, 2015, pp. 806–814.

[35] S. S. Haykin, Neural networks and learning machines, 3rd Edition, Pearson, 2009.

[36] J. Moody, S. Hanson, A. Krogh, J. A. Hertz, A simple weight decay can improve generalization, Advances in Neural Information Processing Systems 4 (1995) 950–957.

[37] P. Ochs, A. Dosovitskiy, T. Brox, T. Pock, On iteratively reweighted algorithms for nonsmooth nonconvex optimization in computer vision, SIAM Journal on Imaging Sciences 8 (1) (2015) 331–372.

[38] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, Y. Bengio, Theano: A cpu and gpu math compiler in python, in: Proc. 9th Python in Science Conf, 2010, pp. 1–7.

[39] Y. Bengio, Practical recommendations for gradient-based training of deep architectures, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 437–478.

[40] A. Ben-Tal, A. Nemirovski, Lectures on modern convex optimization: analysis, algorithms, and engineering applications, Vol. 2, SIAM, 2001.

[41] K. Pelckmans, J. A. Suykens, B. De Moor, Morozov, ivanov and tikhonov regularization based ls-svms, in: International Conference on Neural Information Processing, Springer, 2004, pp. 1216–1222.

[42] X. Glorot, A. Bordes, Y. Bengio, Deep sparse rectifier neural networks, in: International Conference on Artificial Intelligence and Statistics, 2011, pp. 315–323.

[43] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feed-forward neural networks, in: International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256.

[44] D. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[45] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.

[46] F. Alimoglu, E. Alpaydin, Methods of combining multiple classifiers based on different representations for pen-based handwritten digit recognition, in: Proceedings of the Fifth Turkish Artificial Intelligence and Artificial Neural Networks Symposium (TAINN, Citeseer, 1996.

[47] C. Bayer, O. Enge-Rosenblatt, M. Bator, U. Monks, Sensorless drive diagnosis using automated feature extraction, significance ranking and reduction, in: 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA), IEEE, 2013, pp. 1–4.

[48] L. Deng, The mnist database of handwritten digit images for machine learning research, IEEE Signal Processing Magazine 29 (6) (2012) 141–142.

[49] J. A. Blackard, D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from

cartographic variables, Computers and electronics in agriculture 24 (3) (1999) 131–151.

[50] G. Huang, Y. Sun, Z. Liu, D. Sedra, K. Weinberger, Deep networks with stochastic depth, arXiv preprint arXiv:1603.09382.

[51] K. Suzuki, I. Horiba, N. Sugie, A simple neural network pruning algorithm with application to filter synthesis, Neural Processing Letters 13 (1) (2001) 43–53.

[52] G. Scutari, F. Facchinei, P. Song, D. P. Palomar, J.-S. Pang, Decomposition by partial linearization: Parallel optimization of multi-agent systems, IEEE Transactions on Signal Processing 62 (3) (2014) 641–656.

[53] S. Scardapane, R. Fierimonte, P. Di Lorenzo, M. Panella, A. Uncini, Distributed semi-supervised support vector machines, Neural Networks 80 (2016) 43–52.

[54] F. M. Bianchi, S. Scardapane, A. Uncini, A. Rizzi, A. Sadeghian, Prediction of telephone calls load using echo state network with exogenous variables, Neural Networks 71 (2015) 204–213.