Neurocomputing 000 (2017) 1–9

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

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, Rome 00184 , Italy
[b] Division of Computing Science & Maths, School of Natural Sciences, University of Stirling, Stirling, Scotland FK9 4LA, UK

ARTICLE INFO

Article history:
Received 7 October 2016
Revised 24 January 2017
Accepted 7 February 2017
Available online xxx

Communicated by Ma Lifeng Ma

Keywords:
Deep networks
Group sparsity
Pruning
Feature selection

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 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.

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 [1–4]. 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 has 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 [15]. Feature selection and pruning are related problems: adding a new set of features to a task generally results in the need

* Corresponding author.
E-mail addresses: simone.scardapane@uniroma1.it, simonescardapane@gmail.com (S. Scardapane), danilo.comminiello@uniroma1.it (D. Comminiello), ahu@cs.stir.ac.uk (A. Hussain), aurelio.uncini@uniroma1.it (A. Uncini).

http://dx.doi.org/10.1016/j.neucom.2017.02.029
0925-2312/

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, dimensionality reduction of the input vector via linear or nonlinear mappings), including principal component analysis, mutual information [16], 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 [17]. 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 [18,19]. 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 [20]. 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 [21,22], 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 [23,24]. 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 [25], including multikernel [26] and multitask problems [27]. 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. [28] used a group sparse penalty to select groups of features co-occurring in a robotic control task. Similarly, Zhu et al. [29] have used a group sparse formulation to select informative groups of features in a multi-modal context. Liu et al. [30] 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:

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

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 loss 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 [31].

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' [32], 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

Please cite this article as: S. Scardapane et al., Group sparse regularization for deep neural networks, Neurocomputing (2017), http://dx.doi.org/10.1016/j.neucom.2017.02.029