LOss-Based SensiTivity rEgulaRization: Towards deep sparse neural networks

Published version:

DOI:10.1016/j.neunet.2021.11.029

LOss-Based SensiTivity rEgulaRization: Towards Deep Sparse Neural Networks

Enzo Tartaglione[a,b], Andrea Bragagnolo[a], Attilio Fiandrotti[a], Marco Grangetto[a]

[a] *Università degli Studi di Torino, corso Svizzera 185, Torino (Italy)*
[b] *LTCI, Télécom Paris, Institut Polytechnique de Paris*

Abstract

LOBSTER (LOss-Based SensiTivity rEgulaRization) is a method for training neural networks having a sparse topology. Let the sensitivity of a network parameter be the variation of the loss function with respect to the variation of the parameter. Parameters with low sensitivity, i.e. having little impact on the loss when perturbed, are shrunk and then pruned to sparsify the network. Our method allows to train a network from scratch, i.e. without preliminary learning or rewinding. Experiments on multiple architectures and datasets show competitive compression ratios with minimal computational overhead.

Keywords: pruning, regularization, deep learning, sparsity
2010 MSC: 00-01, 99-00

1. Introduction

Artificial Neural Networks (ANNs) achieve state-of-the-art performance in several tasks at the price of complex topologies with millions of learnable parameters. As an example, ResNet [1] includes tens of millions of parameters, soaring to hundreds of millions for VGG-Net [2]. A large parameter count jeopardizes however the possibility to deploy a network over a memory-constrained (e.g., embedded, mobile) device, calling for leaner architectures with fewer parameters.

The complexity of ANNs can be reduced enforcing a *sparse* network topology. Namely, some connections between neurons can be *pruned* by wiring the corresponding parameters to zero. Besides parameters count reduction, some works also suggested other benefits including improved performance in transfer learning scenarios [3]. Popular methods such as [4], for example, add a *regularization* term to the cost function with the goal to shrink to zero some parameters. Next, a threshold operator pinpoints the shrunk parameters to zero, eventually enforcing the sought sparse topology. However, such methods require that the topology to be pruned has been preliminarily trained via standard gradient descent, which sums up to the total learning time.

This work contributes LOBSTER (*LOss-Based SensiTivity rEgulaRization*), a method for learning sparse neural topologies. In this context, let us define the *sensitivity* of the parameter of an ANN as the derivative of the loss function with respect to some target parameter. Intuitively, low-sensitivity parameters have a negligible impact on the loss function when perturbed, and so are fit to be shrunk without compromising the network performance. Practically, LOBSTER shrinks to zero parameters with low sensitivity with a regularize-and-prune approach, achieving a sparse network topology. With respect to similar literature [5, 6, 7], LOBSTER does not require a preliminary training stage to learn the dense reference topology to prune. Moreover, differently to other sensitivity-based approaches, LOBSTER computes the sensitivity exploiting the already available gradient of the loss function, avoiding additional derivative computations [8, 9], or second-order derivatives [10]. Our experiments performed over different network topologies and datasets show that LOBSTER outperforms several competitors in multiple tasks.

The rest of this paper is organized as follows. In Sec. 2 we review the relevant literature concerning sparse neural architectures. Next, in Sec. 3 we describe our method for training a neural network such that its topology is sparse. We provide a general overview on the technique in Sec. 4. Then, in Sec. 5 we experiment with our proposed training scheme over some deep ANNs on a number of different datasets. Finally, Sec. 6 draws the conclusions while providing further directions for future research.

2. Related Works

It is well known that many ANNs, trained on some tasks, are typically over-parametrized [11, 12]. Attempts to reduce the number of parameters from learned models deepen their roots in the past. In 1989, Mozer and Smolensky proposed *skeletonization*, a technique to identify less relevant neurons in a trained model and to remove them [8]. This is accomplished evaluating the global effect of removing a given neuron, evaluated as error function penalty from a pre-trained model. In the same years, LeCun et al. [10] proposed a work where the information from the second order derivative of the error function is leveraged to rank the parameters of the trained model on a *saliency* basis. This allows to select a trade-off between size of the network (in terms of number of parameters) and performance.

2.1. Non-pruning strategies

Reducing the number of parameters in a deep model does not necessarily involve pruning the network topology. *Soft Weight Sharing* (SWS) [13], for example, shares redundant parameters among layers, resulting in fewer parameters to be stored. Other recent approaches towards reducing a neural network's size rely on knowledge distillation, like *Few Samples Knowledge Distillation* (FSKD) [14]. In FSKD, for example, it is possible to successfully train a small student network from a larger teacher, which has been directly trained on the task. Quantization can also be considered for reducing the model's size: Yang et al., for example, considered the problem of ternarizing a pre-trained deep model [15]. These approaches are different roads towards model's size reduction; however, pruning has a major advantage of being applicable alongside any of the above strategies, as it detects parameters not useful to determine the target task, and remove them.

2.2. Pruning methods

The goal of pruning techniques is to achieve the highest *sparsity* (i.e. the maximum percentage of removed parameters) with minimal performance loss (accuracy loss versus the "un-pruned" model). Towards this end, a number of different approaches have been proposed. *Dropout-based approaches* constitute an intuitive possibility to achieve sparsity. For example, *Sparse VD* relies on variational dropout to promote sparsity [16], providing also a Bayesian interpretation for Gaussian dropout. Another dropout-based approach is *Targeted Dropout* [7], where fine-tuning the ANN model is self-reinforcing its sparsity by stochastically dropping connections (or entire units). Broadening this category are worth of mention all the ensembling-like techniques aiming at find redundancy in the layers and remove it by knocking it off, like SCOP [17].

Recently, it has been observed that only some parameters are actually updated during training [18], suggesting that other parameters can be removed from the learning process without affecting the network performance. These latter parameters can however be only determined a-posteriori, while other pruning strategies can achieve higher sparsity in general [19]. Lots of efforts have recently been devoted towards making pruning mechanisms more efficient: for example, Wang et al. show that some sparsity is achievable pruning weights at the very beginning of the training process [20], Liebenwein et al. build saliency scores to rank filters to be pruned [21], or Lee et al., with their "SNIP", are able to prune weights in a one-shot fashion [22]. However, these approaches achieve limited sparsity only, while strategies based on iterative pruning usually enable higher sparsity [19] when compared to the above mentioned *one-shot* or *few-shot* approaches.

A popular pruning strategy involves the introduction of a *regularization* function which promotes sparsity iteratively, at fine-tuning time. Some attempts to regularize ANN parameters based on the ℓ_0 norm have been reported; however, such norm is a non-differentiable measure and cannot be simply plugged into the optimizer. A recent work [23] proposes a differentiable proxy measure to overcome this problem introducing, though, some relevant computational overhead. In [24] a regularizer based on group lasso, whose goal is to cluster filters in convolutional layers, is proposed. However, such a technique cannot be generalized to bulky fully-connected layers, where most of the complexity (in terms of number of parameters) lies. A sound approach towards pruning parameters consists in exploiting a ℓ_2 regularizer in a shrink-and-prune framework. In particular, a standard ℓ_2 regularization term is included in the minimized cost function (to penalize the magnitude of the parameters): all the parameters dropping below some threshold are pinpointed to zero, thus learning a sparser topology [4]. Such approach is effective since regularization replaces unstable (ill-posed) problems with nearby and stable (well-posed) ones by introducing a prior on the parameters [25]. However, as a drawback, this method requires a preliminary training to learn the threshold value; furthermore, all the parameters are blindly, equally-penalized by their ℓ_2 norm: some parameters, which can introduce large error (if removed), might drop below the threshold because of the regularization term: this introduces sub-optimalities as well as instabilities in the pruning process. Guo et al. attempted to address this issue with their DNS [6], where they propose an algorithmic procedure to corrects eventual over-pruning by enabling the recovery of severed connections, or another possible approach has been proposed by He et al. with a "soft pruning" strategy [26, 27]. In other recent works [9, 28], it was proposed to measure how much the network output changes for small perturbations of some parameters, and to iteratively penalize just those which generate little or no performance loss. However, such method requires the network to be already trained so to measure the variation of the network output when a parameter is perturbed, increasing the overall learning time.

In this work, we overcome the basic limitation of pre-training the network, introducing the concept of *loss-based sensitivity*: it only penalizes the parameters whose small perturbation introduces little or no performance loss at training time.

3. Proposed Regularization

In this section we first express the sensitivity of a network parameter as the variation in the loss function as a function of a perturbation applied to the parameter. Then, we propose a parameter update rule that includes a regularization term accounting for each parameter sensitivity to drive towards zero parameters with small sensitivity.

3.1. Loss-based Sensitivity

ANNs are typically trained via gradient descent based optimization, i.e. minimizing the loss function. Methods based on mini-batches of samples have gained popularity as they allow better generalization than stochastic learning while also being memory and time efficient. In such a framework, a network parameter w_i is updated towards the averaged direction which minimizes the averaged loss for the minibatch, i.e. using the well known stochastic gradient descent or its variations. Our ultimate goal is to assess to which extent a variation of the value of w_i would affect the error on the network output $\mathbf{y}$: the parameters not affecting the network output can be hardwired to zero, i.e. pruned away.
We make a first attempt towards this end introducing a small perturbation Δw_i over w_i and measuring the variation of $\mathbf{y}$ as

$$\Delta \mathbf{y} = \sum_k |\Delta y_k| \approx \Delta w_i \sum_k \left| \frac{\partial y_k}{\partial w_i} \right|. \tag{1}$$

Unfortunately, evaluating (1) is computationally expensive, because it would require a complexity growing linearly with the number of the output classes [9]. We can, however, estimate directly the variations of the error function instead, using some differentiable proxy function, i.e. the loss function L:

$$\Delta L \approx \Delta w_i \left| \frac{\partial L}{\partial \mathbf{y}} \cdot \frac{\partial \mathbf{y}}{\partial w_i} \right| = \Delta w_i \left| \frac{\partial L}{\partial w_i} \right|. \tag{2}$$

$P\left(\frac{\partial L}{\partial w_i}\right)$	$\text{sign}\left(\frac{\partial L}{\partial w_i}\right)$	$\text{sign}\left(w\right)$	$\frac{\tilde{\eta}}{\eta}$
0	any	any	1
1	0	any	1
1	+	+	≤ 1
1	+	-	≥ 1
1	-	+	≥ 1
1	-	-	≤ 1

Table 1: Behavior of $\tilde{\eta}$ compared to η ($\eta > 0$).

The use of (2) in place of (1) shifts the focus from the output to the error of the network. Let us define the *sensitivity* S for a given parameter w_i as

$$S(L, w_i) = \left|\frac{\partial L}{\partial w_i}\right|. \tag{3}$$

Large S values indicate large variations of the loss function for small perturbations of w_i.

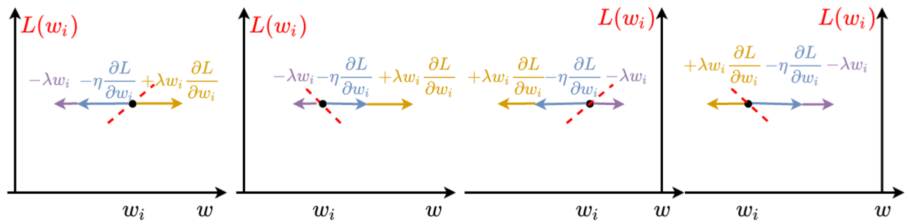
3.2. Update Rule

Given the sensitivity definition in (3), we can promote sparse topologies by pruning parameters with both low sensitivity S (i.e., in a flat region of the loss function gradient, where a small perturbation of the parameter has a negligible effect on the loss) and low magnitude. Towards this end, we propose the following parameter update rule to promote sparsity:

$$w_i^{t+1} := w_i^t - \eta \frac{\partial L}{\partial w_i^t} - \lambda w_i^t \left[1 - S(L, w_i^t)\right] P\left[S(L, w_i^t)\right], \tag{4}$$

where

$$P(x) = \Theta\left[1 - |x|\right], \tag{5}$$



(a) (b) (c) (d)

Figure 1: Update rule effect on the parameters. The red dashed line is the tangent to the loss function in the black dot, in blue the standard SGD contribution, in purple the weight decay while in orange the LOBSTER contribution. Here we assume $P(L, w_i) = 1$.

$\Theta(\cdot)$ is the one-step function and η, λ two positive hyper-parameters. Plugging (3) in (4) we can rewrite the update rule as:

$$w_i^{t+1} = w_i^t - \eta \frac{\partial L}{\partial w_i^t} - \lambda \Gamma \left(L, w_i^t\right) \left[1 - \left|\frac{\partial L}{\partial w_i^t}\right|\right], \tag{6}$$

where

$$\Gamma\left(y, x\right) = x \cdot P\left(\frac{\partial y}{\partial x}\right). \tag{7}$$

After some algebraic manipulations, we can rewrite (6) as

$$w_i^{t+1} = w_i^t - \lambda \Gamma\left(L, w_i^t\right) - \frac{\partial L}{\partial w_i^t}\left[\eta - \text{sign}\left(\frac{\partial L}{\partial w_i^t}\right) \lambda \Gamma\left(L, w_i^t\right)\right]. \tag{8}$$

In (8), we observe two different components of the proposed regularization term:

- a weight decay-like term $\Gamma\left(L, w_i\right)$ which is enabled/disabled by the magnitude of the gradient on the parameter;
- a correction term for the learning rate. In particular, the full learning process follows an *equivalent* learning rate

$$\tilde{\eta} = \eta - \text{sign}\left(\frac{\partial L}{\partial w_i}\right) \lambda \Gamma\left(L, w_i\right). \tag{9}$$

Let us analyze the corrections in the learning rate. If $\left|\frac{\partial L}{\partial w_i}\right| \geq 1$ (w_i has large sensitivity), it follows that $P\left(\frac{\partial L}{\partial w_i}\right) = 0$ and $\Gamma\left(L, w_i\right) = 0$ and the dominant contribution comes from the gradient. In this case our update rule reduces to the classical GD:

$$w_i^{t+1} = w_i^t - \eta \frac{\partial L}{\partial w_i^t}. \tag{10}$$

When we consider less sensitive w_i with $\left|\frac{\partial L}{\partial w_i}\right| < 1$, we get $\Gamma\left(L, w_i\right) = w_i$ (weight decay term) and we can distinguish two sub-cases for the learning rate:

- if $\text{sign}\left(\frac{\partial L}{\partial w_i}\right) = \text{sign}\left(w_i\right)$, then $\tilde{\eta} \leq \eta$ (Fig. 1a and Fig. 1d),
- if $\text{sign}\left(\frac{\partial L}{\partial w_i}\right) \neq \text{sign}\left(w_i\right)$, then $\tilde{\eta} \geq \eta$ (Fig. 1b and Fig. 1c).

Finally, let us consider the corner case where the network has "fully-converged" over the training set, i.e. $\left|\frac{\partial L}{\partial w_i}\right| = 0 \forall w_i$. In this case, the update rule in (4) reduces to

$$w_i^{t+1} := (1-\lambda) w_i^t \tag{11}$$

as $P[S(L, w_i^t)] = 1$. The only term remaining here is a weight decay-like term, which greedily tends to push the parameters towards zero. A schematics of all these cases can be found in Table 1 and the representation of the possible effects are shown in Fig. 1. The contribution coming from $\Gamma\left(L, w_i\right)$ aims at minimizing the parameter magnitude, disregarding the loss minimization. If the loss minimization tends to minimize the magnitude as well, then the equivalent learning rate is reduced. On the contrary, when the gradient descent tends to increase the magnitude, the learning rate is increased, to compensate the contribution coming from $\Gamma\left(L, w_i\right)$. This mechanism allows us to succeed in the learning task while introducing sparsity.

3.3. Regularization function minimized

Let us now investigate more precisely the objective function we are minimizing by imposing the update rule (6). To this end we can follow the approach as in [9], and we can compute the regularization function R_i (computed for the single parameter w_i) by solving

$$R_i = \int \left(w_i - w_i \left|\frac{\partial L}{\partial w_i}\right|\right) \Theta\left(1 - \left|\frac{\partial L}{\partial w_i}\right|\right) dw_i. \tag{12}$$

We rewrite R_i as the ℓ_2 regularization followed by a correction term as

$$R_i = \Theta\left(1 - \left|\frac{\partial L}{\partial w_i}\right|\right)\left(\frac{w_i^2}{2} + \tilde{R}_i\right), \tag{13}$$

where

$$\tilde{R}_i = -\int w_i \frac{\partial L}{\partial w_i} \text{sign}\left(\frac{\partial L}{\partial w_i}\right) dw_i. \tag{14}$$

Let us integrate (14) by parts:

$$\tilde{R}_i = -\frac{w_i^2}{2}\frac{\partial L}{\partial w_i}\text{sign}\left(\frac{\partial L}{\partial w_i}\right) + \int \frac{w_i^2}{2}\frac{\partial^2 L}{\partial w_i^2}\text{sign}\left(\frac{\partial L}{\partial w_i}\right) dw_i. \tag{15}$$

If we integrate a further step, we obtain:

$$\tilde{R}_i = -\frac{w_i^2}{2}\frac{\partial L}{\partial w_i}\text{sign}\left(\frac{\partial L}{\partial w_i}\right) + \frac{w_i^3}{6}\frac{\partial^2 L}{\partial w_i^2}\text{sign}\left(\frac{\partial L}{\partial w_i}\right) - \int \frac{w_i^3}{6}\frac{\partial^3 L}{\partial w_i^3}\text{sign}\left(\frac{\partial L}{\partial w_i}\right) dw_i. \tag{16}$$

Applying infinite steps of integration by parts we have

$$\tilde{R}_i = \text{sign}\left(\frac{\partial L}{\partial w_i}\right)\sum_{k=1}^{\infty}(-1)^k \frac{\partial^k L}{\partial w_i^k}\frac{w_i^{k+1}}{(k+1)!}. \tag{17}$$

Overall, the regularization function to minimize at training time, over all the w_i, is

$$R = \sum_i \Theta\left(1 - \left|\frac{\partial L}{\partial w_i}\right|\right)\left[\frac{w_i^2}{2} + \text{sign}\left(\frac{\partial L}{\partial w_i}\right)\sum_{k=1}^{\infty}(-1)^k \frac{\partial^k L}{\partial w_i^k}\frac{w_i^{k+1}}{(k+1)!}\right]. \tag{18}$$

According to (13) and, for instance, (18), we observe that overall the regularization function we are minimizing is the standard ℓ_2 regularization, corrected by a loss-dependent term, defined within our proposed LOBSTER framework. In the next section we provide a practical procedure to learn a sparse neural network topology exploiting the above regularization function at training time, followed by a pruing stage.

4. Training Procedure

This section describes a procedure to train a sparse neural network $\mathcal{N}$ leveraging the sensitivity-based rule above to update the network parameters. We assume that the parameters have been randomly initialized, albeit the procedure holds also if the network has been pre-trained. The procedure is illustrated in Fig. 2 and iterates over two stages as follows.

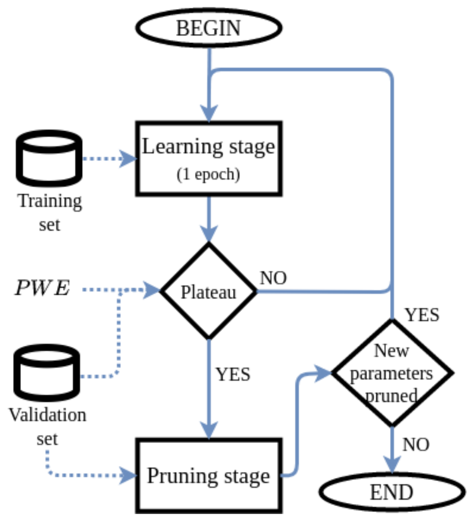


Figure 2: The complete training procedure of LOBSTER.

4.1. Learning Stage

During the learning stage, the ANN is iteratively trained according to the update rule (4) on some training set. Let j indicate the current learning stage iteration (i.e., epoch) and $\mathcal{N}^j$ represent the network (i.e., the set of learnable parameters) at the end of the j-th iteration. Also let L^j be the loss measured on some validation set at the end of the j-th iteration and $\widehat{L}$ be the best (lowest) loss measured so far on $\widehat{\mathcal{N}}$ (network with lowest validation loss so far). As initial condition, we assume, $\widehat{\mathcal{N}} = \mathcal{N}^0$. If $L^j < \widehat{L}$, the reference to the best network is updated as $\widehat{\mathcal{N}} = \mathcal{N}^j$, $\widehat{L} = L^j$. We iterate again the learning stage $\mathcal{N}$ until the best validation loss L^j has not decreased for PWE iterations of the learning stage in a row (we say the regularizer has reached a performance *plateau*). At such point, we move to the pruning stage.

We provide $\widehat{\mathcal{N}}$ as input for the pruning stage, where a number of parameters have been shrunk towards zero by our sensitivity-based regularizer.

4.2. Pruning Stage

In a nutshell, during the pruning stage parameters with magnitude below a threshold value T are pinpointed to zero, eventually sparsifying the network

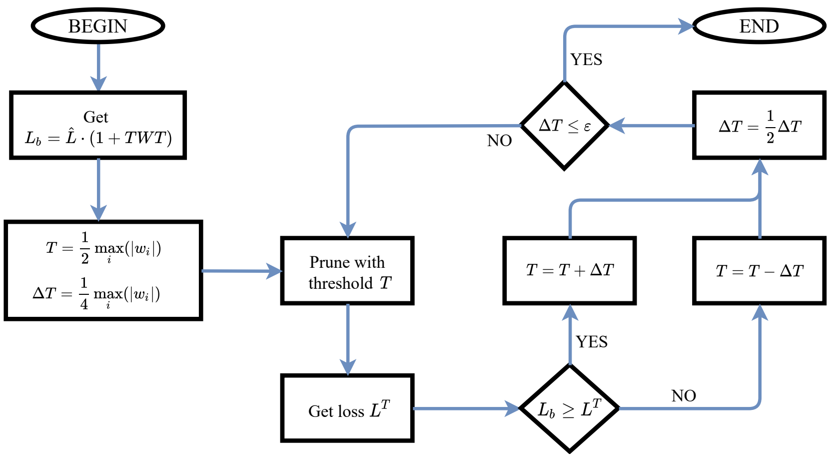


Figure 3: The pruning stage.

topology as shown in Fig. 3. Namely, we look for the largest T that worsens the classification loss $\widehat{L}$ at most by a relative quantity TWT:

$$L_b = (1 + TWT)\,\widehat{L}, \tag{19}$$

where L_b is called *loss boundary*. T is found using the bisection method, initializing T with the average magnitude of the non-null parameters in the network. Then, we apply the threshold T to $\widehat{\mathcal{N}}$ obtaining the pruned network $\mathcal{N}^T$ with its loss L^T on the validation set. Before the pruning procedure begins, we initialize the threshold T to half of the maximum magnitude for the parameters in $\widehat{\mathcal{N}}$. We also initialize ΔT to $\frac{T}{2}$. Then, we proceed in the research of T as follows:

1. We prune $\widehat{\mathcal{N}}$ with threshold T, obtaining $\mathcal{N}^T$;
2. We compute the loss L^T on the validation set:
   - if $L_b \geq L^T$ the network tolerates that more parameters be pruned, so T is increased by ΔT;
   - if $L_b < L^T$ then too many parameters have been pruned. This means that we have to restore the parameters pruned at the previous step. Then, we decrease T by ΔT.
3. We update ΔT, dividing its value by half;

4. We test over ΔT value:
   - if $\Delta T \leq \varepsilon$, where ε is some small value for which $L^T = L^{T+\varepsilon}$ (typically 10^{-10}), we end our pruning stage;
   - otherwise, we go back to point 1.

Once T is found, all the parameters whose magnitude is below T are pinpointed to zero, i.e. they are pruned for good. If at least one parameter has been pruned during the last iteration of the pruning stage, a new iteration of the regularization stage follows; otherwise, the procedure ends returning the trained, sparse network.

5. Results

In this section we experimentally evaluate LOBSTER over multiple architectures and datasets commonly used as benchmark in the literature:

- LeNet-300 on MNIST (Fig. 4a),
- LeNet-5 on MNIST (Fig. 4b),
- LeNet-5 on Fashion-MNIST (Fig. 4c),
- ResNet-32 on CIFAR-10 (Fig. 4d),
- ResNet-18 on ImageNet (Fig. 4e),
- ResNet-101 on ImageNet (Fig. 4f),
- U-Net on ISIC skin lesion segmentation (Table 2).

We compare with other state-of-the-art approaches introduced in Sec. 2 wherever numbers are publicly available. Besides these, we also perform an ablation study with a ℓ_2-based regularizer and our proposed pruning strategy (as discussed in Sec. 4.2). Performance is measured as the achieved model sparsity versus classification error (Top-1 or Top-5 error). The network sparsity is defined here ad the percentage of pruned parameters in the ANN model. Our algorithms are implemented in Python[1], using PyTorch 1.2 and simulations are run over an RTX2080 Ti NVIDIA GPU. All the hyper-parameters have been tuned via grid-search. The validation set size for all the experiments is 5k large. For all datasets, the learning and pruning stages take place on a random split of the training set, whereas the numbers reported below are related to the test set.

5.1. LeNet-300 on MNIST

As a first experiment, we train a sparse LeNet-300 [29] architecture, which consists of three fully-connected layers with 300, 100 and 10 neurons respectively. We trained the network on the MNIST dataset, made of 60k training images and 10k test gray-scale 28×28 pixels large images, depicting handwritten digits. Starting from a randomly initialized network, we trained LeNet-300 via SGD with learning rate $\eta = 0.1$, $\lambda = 10^{-4}$, $PWE = 20$ epochs and $TWT = 0.05$. The related literature reports several compression results that can be clustered in two groups corresponding to classification error rates of about 1.65% and 1.95%, respectively. Fig. 4a provides results for the proposed procedure. Our method reaches higher sparsity than the the approaches found in literature. This is particularly noticeable around 1.65% classification error (low left in Fig. 4a), where we achieve almost twice the sparsity of the second best method. LOBSTER also achieves the highest sparsity for the higher error range (right side of the graph), gaining especially in regards to the number of parameters removed from the first fully-connected layer (the largest, consisting of 235k parameters), in which we observe just the 0.59% of the parameters survives.

5.2. LeNet-5 on MNIST and Fashion-MNIST

Next, we experiment on the caffe version of the LeNet-5 architecture, consisting in two convolutional and two fully-connected layers. Again, we use a randomly-initialized network, trained via SGD with learning rate $\eta = 0.1$,

[1] The source code is available at `https://github.com/EIDOSlab/LOBSTER.git`

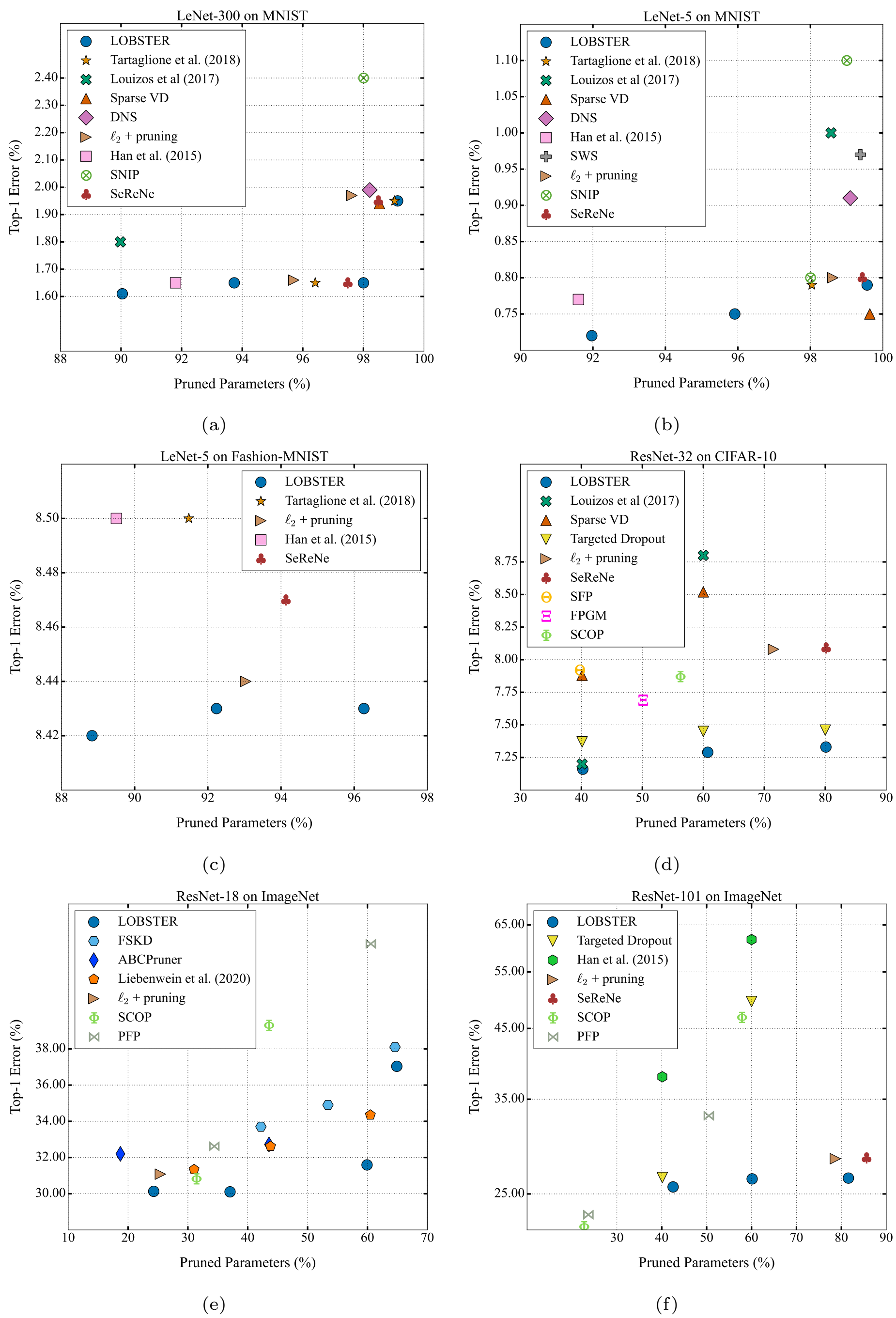


Figure 4: Performance (Top-1 error) vs ratio of pruned parameters for LOBSTER and other state of the art methods over different architectures and datasets.

$\lambda = 10^{-4}$, $PWE = 20$ epochs and $TWT = 0.05$. The results are shown in Fig. 4b. Even with a convolutional architecture, we obtain a competitively small network with a sparsity of 99.57%. At higher compression rates, Sparse VD slightly outperforms all other methods in the LeNet5-MNIST experiment. We observe that LOBSTER, in this experiment, sparsifies the first convolutional layer (22% sparsity) more than Sparse VD solution (33%). In particular, LOBSTER prunes 14 filters out of the 20 original in the first layer (or in other words, just 6 filters survive, and contain all the un-pruned parameters). We hypothesize that, in the case of Sparse VD and for this particular dataset, extracting a larger variety of features at the first convolutional layer, both eases the classification task (hence the lower Top-1 error) and allows to drop more parameters in the next layers (a slightly improved sparsity). However, since we are above 99% of sparsity, the difference between the two techniques is minimal.

To scale-up the difficulty of the training task, we experimented on the classification of the Fashion-MNIST dataset [30], using again LeNet5. This dataset has the same size and image format of the MNIST dataset, yet it contains images of clothing items, resulting in a non-sparse distribution of the pixel intensity value. Since the images are not as sparse, such dataset is notoriously harder to classify than MNIST. For this experiment, we trained the network from scratch using SGD with $\eta = 0.1$, $\lambda = 10^{-4}$, $PWE = 20$ epochs and $TWT = 0.1$. The results are shown in Fig. 4c.

F-MNIST is an inherently more challenging dataset than MNIST, so the achievable sparsity is lower. Nevertheless, the proposed method still reaches higher sparsity than other approaches, removing an higher percentage of parameters, especially in the fully connected layers, while maintaining good generalization. In this case, we observe that the first layer is the least sparsified: this is an effect of the higher complexity of the classification task, which requires more features to be extracted.

5.3. ResNet-32 on CIFAR-10

To evaluate how our method scales to deeper, modern architectures, we applied it on a PyTorch implementation of the ResNet-32 network [31] that classifies the CIFAR-10 dataset.[2] This dataset consists of 60k 32×32 RGB images divided in 10 classes (50k training images and 10k test images). We trained the network using SGD with momentum $\beta = 0.9$, $\lambda = 10^{-6}$, $PWE = 10$ and $TWT = 0$. The full training is performed for 11k epochs.Our method performs well on this task and outperforms other state-of-the-art techniques. Furthermore, LOBSTER improves the network generalization ability reducing the baseline Top-1 error from 7.37% to 7.33% of the sparsified network while removing 80.11% of the parameters. This effect is most likely due to the LOBSTER technique itself, which self-tunes the regularization on the parameters as explained in Sec. 3.2.

5.4. ResNet on ImageNet

Finally, we further scale-up both the output and the complexity of the classification problem testing the proposed method on network over the well-known ImageNet dataset (ILSVRC-2012), composed of more than 1.2 million train images, for a total of 1k classes. For this test we used SGD with momentum $\beta = 0.9$, $\lambda = 10^{-6}$ and $TWT = 0$. The full training lasts 95 epochs. Due to time constraints, we decided to use the pre-trained network offered by the torchvision library.[3] Fig. 4e shows the results for ResNet-18 while Fig.4f shows the results for ResNet-101. Even in this scenario, LOBSTER proves to be particularly efficient: we are able to remove, with no performance loss, 37.04% of the parameters from ResNet-18 and 81.58% from ResNet-101.

5.5. U-Net on ISIC skin lesion segmentation

Besides classification tasks, we want to show how LOBSTER behaves for different tasks. Towards this end, we have trained the U-Net architecture [32] to

[2] `https://github.com/akamaster/pytorch_resnet_cifar10`

[3] `https://pytorch.org/docs/stable/torchvision/models.html`

Table 2: esults on the ISIC 2018 Skin Lesion Segmentation using U-Net architecture.

Method	Dice score	Intersection over Union	Sparsity (%)
Baseline	**0.8282**	**0.7073**	0
Sparse VD [16]	0.8245	0.7030	32.14
ℓ_2+pruning	0.8273	0.7062	79.43
LOBSTER	0.8269	0.7057	**82.13**

segment skin lesions [33, 34]. The ISIC skin lesion segmentation dataset consists of 2594 training images and 100 test images having resolution 1024×768 pixels, in RGB format. Models are trained with weight decay = 10^{-4}, momentum = 0.9 starting learning rate $\eta = 0.1$. LOBSTER and ℓ_2+pruning models were obtained with $PWE = 10$ and $TWT = 0$. For LOBSTER we used $\lambda = 10^{-4}$. All the models are trained minimizing a Jaccard loss function.
Results are shown in Table 2. Even in segmentation tasks LOBSTER is able to remove a very large amount of parameters, namely the 82.13%. We observe, however, that in this specific scenario the main contribution is given by the pruning algorithm we propose in Sec. 4.2, as the sparsity achieved with plain ℓ_2 regularization is not distant though lower than LOBSTER (82.13%) when other techniques which perform well for classification tasks, like Sparse VD, in this case achieve just 32.14% performance. For these cases, a proper tuning of the threshold T results determinant towards achieving high performance with little performance drop.

5.6. Ablation study

As a final ablation study, we replace our sensitivity-based regularizer with a simpler ℓ_2 regularizer in our learning scheme in Fig. 2. Such scheme "ℓ_2+pruning" uniformly applies an ℓ_2 penalty to all the parameters regardless their contribution to the loss. This scheme is comparable with [4], yet enhanced with the same pruning strategy with adaptive thresholding shown in Fig. 3. A comparison between LOBSTER and ℓ_2+pruning is reported in Table 3. We report

Dataset	Architecture	ℓ_2+pruning			LOBSTER		
		Top-1 (%)	Sparsity (%)	FLOPs	Top-1 (%)	Sparsity (%)	FLOPs
MNIST	LeNet-300	1.97	97.62	22.31k	1.95	99.13	10.63k
	LeNet-5	0.80	98.62	589.75k	0.79	99.57	207.38k
F-MNIST	LeNet-5	8.44	93.04	1628.39k	8.43	96.27	643.22k
CIFAR-10	ResNet-32	8.08	71.51	44.29M	7.33	80.11	32.90M
ImageNet	ResNet-18	31.08	25.40	2.85G	30.10	37.04	2.57G
	ResNet-101	28.33	78.67	3.44G	26.44	81.58	3.00G

Table 3: Comparing LOBSTER against standard ℓ_2+pruning as in Fig. 4 (best sparsity results are reported).The sensitivity-based regularization term allows higher sparsification rates for improved accuracy.

in table some operation points, for which ℓ_2+pruning and LOBSTER have the same Top-1 performance: for lower sparsity the performance typically increases. In all the experiments we observe that dropping the sensitivity based regularizer impairs the performance. This experiment verifies the role of the sensitivity-based regularization in the performance of our scheme. Finally, Table 3 also reports the corresponding inference complexity in FLOPs. For the same or lower Top-1 error LOBSTER yelds benefits as fewer operations at inference time and suggesting the presence of some structure in the sparsity achieved by LOBSTER.

6. Conclusion

We presented LOBSTER, a regularization method suitable to train neural networks with a sparse topology without a preliminary training. Differently from ℓ_2 regularization, LOBSTER is aware of the global contribution of the parameter on the loss function and self-tunes the regularization effect on the parameter depending on factors like the ANN architecture or the training problem itself (in other words, the dataset). Moreover, tuning its hyper-parameters is easy and the optimal threshold for parameter pruning is self-determined by the proposed approach employing a validation set. LOBSTER achieves competitive results from shallow architectures like LeNet-300 and LeNet-5 to deeper topologies like ResNet over ImageNet. In these scenarios we have observed the boost provided by the proposed regularization approach towards less-unaware approaches like ℓ_2 regularization, in terms of achieved sparsity.

Future research includes the extension of LOBSTER to achieve sparsity with a structure and a thorough evaluation of the savings in terms of memory footprint.

References

[1] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[2] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, arXiv preprint arXiv:1409.1556.

[3] J. Liu, Y. Wang, Y. Qiao, Sparse deep transfer learning for convolutional neural network, in: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017, pp. 2245–2251.

[4] 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.

[5] S. Han, H. Mao, W. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[6] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient dnns, in: Advances In Neural Information Processing Systems, 2016, pp. 1379–1387.

[7] A. N. Gomez, I. Zhang, K. Swersky, Y. Gal, G. E. Hinton, Learning sparse networks using targeted dropout, CoRR abs/1905.13678. arXiv:1905.13678.

[8] M. C. Mozer, P. Smolensky, Skeletonization: A technique for trimming the fat from a network via relevance assessment, in: Advances in neural information processing systems, 1989, pp. 107–115.

[9] E. Tartaglione, S. Lepsøy, A. Fiandrotti, G. Francini, Learning sparse neural networks via sensitivity-driven regularization, in: Advances in Neural Information Processing Systems, 2018, pp. 3878–3888.

[10] Y. LeCun, J. S. Denker, S. A. Solla, Optimal brain damage, in: Advances in neural information processing systems, 1990, pp. 598–605.

[11] H. N. Mhaskar, T. Poggio, Deep vs. shallow networks: An approximation theory perspective, Analysis and Applications 14 (06) (2016) 829–848.

[12] A. Brutzkus, A. Globerson, E. Malach, S. Shalev-Shwartz, Sgd learns over-parameterized networks that provably generalize on linearly separable data, 2018.

[13] K. Ullrich, M. Welling, E. Meeds, Soft weight-sharing for neural network compression, 2019.

[14] T. Li, J. Li, Z. Liu, C. Zhang, Few sample knowledge distillation for efficient network compression, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14639–14647.

[15] L. Yang, Z. He, D. Fan, Harmonious coexistence of structured weight pruning and ternarization for deep neural networks., in: AAAI, 2020, pp. 6623–6630.

[16] D. Molchanov, A. Ashukha, D. Vetrov, Variational dropout sparsifies deep neural networks, Vol. 5, 2017, pp. 3854–3863.

[17] Y. Tang, Y. Wang, Y. Xu, D. Tao, C. Xu, C. Xu, C. Xu, Scop: Scientific control for reliable neural network pruning, arXiv preprint arXiv:2010.10732.

[18] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2019.

[19] E. Tartaglione, A. Bragagnolo, M. Grangetto, Pruning artificial neural networks: a way to find well-generalizing, high-entropy sharp minima, arXiv preprint arXiv:2004.14765.

[20] Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, X. Hu, Pruning from scratch., in: AAAI, 2020, pp. 12273–12280.

[21] L. Liebenwein, C. Baykal, H. Lang, D. Feldman, D. Rus, Provable filter pruning for efficient neural networks, in: International Conference on Learning Representations, 2020.

[22] N. Lee, T. Ajanthan, P. Torr, Snip: Single-shot network pruning based on connection sensitivity, ArXiv abs/1810.02340.

[23] C. Louizos, M. Welling, D. P. Kingma, Learning sparse neural networks through l_0 regularization, arXiv preprint arXiv:1712.01312.

[24] W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 2074–2082.

[25] C. W. Groetsch, Inverse Problems in the Mathematical Sciences, Vieweg, 1993.

[26] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, arXiv preprint arXiv:1808.06866.

[27] Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, Filter pruning via geometric median for deep convolutional neural networks acceleration, in: Proceedings of

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 4340–4349.

[28] E. Tartaglione, A. Bragagnolo, F. Odierna, A. Fiandrotti, M. Grangetto, Serene: Sensitivity-based regularization of neurons for structured sparsity in neural networks, IEEE Transactions on Neural Networks and Learning Systems (2021) 1–14doi:10.1109/TNNLS.2021.3084527.

[29] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proceedings of the IEEE 86 (11) (1998) 2278 – 2324.

[30] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, CoRR abs/1708.07747. arXiv:1708.07747.

[31] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CoRR abs/1512.03385. arXiv:1512.03385.

[32] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical image computing and computer-assisted intervention, Springer, 2015, pp. 234–241.

[33] N. Codella, V. Rotemberg, P. Tschandl, M. E. Celebi, S. Dusza, D. Gutman, B. Helba, A. Kalloo, K. Liopyris, M. Marchetti, et al., Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic), arXiv preprint arXiv:1902.03368.

[34] P. Tschandl, C. Rosendahl, H. Kittler, The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions, Scientific data 5 (1) (2018) 1–9.