IRIS AperTO

UNIVERSITÀ DEGLI STUDI DI TORINO

AperTO - Archivio Istituzionale Open Access dell'Università di Torino

LOss-Based SensiTivity rEgulaRization: towards deep sparse neural networks

This is a pre print version of the following article:

Original Citation:

Availability:

This version is available http://hdl.handle.net/2318/1737767 since 2020-04-29T18:11:15Z

Terms of use:

Open Access

Anyone can freely access the full text of works made available as "Open Access". Works made available under a Creative Commons license can be used according to the terms and conditions of said license. Use of all other works requires consent of the right holder (author or publisher) if not exempted from copyright protection by the applicable law.

(Article begins on next page)

12 October 2024

LOss-Based SensiTivity rEgulaRization: towards deep sparse neural networks

Enzo Tartaglione [1] **Andrea Bragagnolo** [1] **Marco Grangetto** [1] **Skjalg Lepsøy**

Abstract

LOBSTER is a sensitivity-based regularizer, safely usable at training time, designed for making deep neural network models very sparse. LOBSTER drives some but not all weights towards zero: it will shrink those weights for which the loss derivative is small, such that many weights eventually become close to zero. Those that are close enough to zero will be deleted, i.e. set to zero.
In scenarios such as LeNet-300, LeNet-5, ResNet-32 and ResNet-102 trained on MNIST, Fashion-MNIST, CIFAR-10 and ImageNet, LOBSTER yields a significant and competitive reduction of the number of nonzero weights with a minimal computation overhead.

1. Introduction

Deep neural networks are nowadays the state-of-the-art tool to solve some complex tasks, ranging from image classification (Recht et al., 2019; Belilovsky et al., 2019; Tartaglione et al., 2019) to segmentation (Badrinarayanan et al., 2017; Li et al., 2018), natural language processing (Greaves-Tunnell & Harchaoui, 2019) and many more contexts because of their practical effectiveness. This, however, comes at a cost: the complexity of the model (in terms of number of learnable parameters). As an example, ResNet architectures require tens of millions of parameters, moving to hundreds of millions when talking about VGG-Net (Simonyan & Zisserman, 2014). This, of course, prevents their portability to embedded or more in general low-memory devices.

Recently, many attempts on reducing the complexity of an artificial neural network (ANN) model have been proposed: one of these is introducing *sparsity* in these models. Such a sparsity is introduced by *pruning* some connections in the ANN and, consequently, reducing the complexity of the model. Empirically, it was already known few years ago that the typical ANN models trained are over-parametrized (or, in other words, they have more parameters than necessary) (Brutzkus et al., 2018; Mhaskar & Poggio, 2016) and also recently it is suggested that the number of parameters effectively learning is reduced (Frankle & Carbin, 2019).

One of the approaches aiming at promoting sparsity in ANN model is based on the design of a proper *regularization* term, to be used during the learning process or fine-tuning. In a very general view, regularization replaces unstable (ill-posed) problems with nearby and stable (well-posed) ones by introducing a prior on the parameters (Groetsch, 1993).

The original contribution of this work is the design of a new, efficient regularization term, *LOss-Based SensiTivity rEgulaRization* (LOBSTER), an evolution of the classical Tikhonov regularization and of the sensitivity-based approach (Tartaglione et al., 2018). The aim of LOBSTER is to supervisedly drive the parameters which are not useful to the learning process (this is evaluated by a measure we call loss-based sensitivity) to have a very small value. In such a way, the contribution of non-relevant parameters becomes negligible. Contrarily to the formulation of sensitivity in (Tartaglione et al., 2018), LOBSTER can be safely used also at training time. We will also see that LOBSTER directly uses the gradient of the loss, introducing a minimal computation overhead. Besides this, a magnitude-based pruning method follows.

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 supervised method for training a neural network such that its interconnection matrix 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 architectures on a number of different datasets. Finally, Sec. 6 draws the conclusions while providing further directions for future

[1]Computer Science Dept., University of Torino, Italy. Correspondence to: Enzo Tartaglione <enzo.tartaglione@unito.it>.

research.

2. Related work

Recently a wide variety of works aimed at achieving sparsity in ANNs via pruning. Some of these, furthermore, show that some improvement in the generalization capability for these models can be introduced as a beneficial side effect. For example, in a recent work (Zhu & Gupta, 2018), have shown that a large sparse architecture improves the network generalization ability in a number of different scenarios.

Soft weight sharing is an interesting concept (Ullrich et al., 2019): through such an approach, it is possible to effectively introduce redundancy in the parameters value, resulting in a lower number of parameters to be stored.

The direct strategy to introduce sparsity in neural networks should be l_0 regularization which, however, is a non-differentiable measure. A recent work (Louizos et al., 2017) introduced a differentiable proxy measure to overcome this problem introducing, though, some relevant computational overhead. Having a similar overall approach, in another work a regularizer based on group lasso whose task is to cluster filters in convolutional layers is proposed (Wen et al., 2016). As a backbone, such a technique is not directly generalizeable to the bulky fully-connected layers, where most of the complexity (as number of parameters) lies.

Dropout-based approaches constitute another possibility to achieve sparsity. For example, *variational dropout* is proposed to promote sparsity (Molchanov et al., 2017), providing also an interesting Bayesian interpretation for gaussian dropout. Another very recent dropout-based approach is *targeted dropout* (Gomez et al., 2019): the training of the ANN model is self-reinforcing sparsity criterion by stochastically drop connections or units.

A more standard approach consists in the iterating of learning and pruning steps, regularizing the ANN model using the classical weight-decay (Han et al., 2015). Such a simple approach showed good and promising results.

The *sensitivity-based regularization* (Tartaglione et al., 2018) is the closest to our work: in their approach, sensitivity is a measure of how much the output changes for small perturbations of a given parameter (differently from what proposed in other works (Engelbrecht & Cloete, 1996; Mrázová & Reitermanová, 2011; Mrazova & Kukacka, 2012)). From such a measure, a regularization strategy, aiming at pushing low-sensitivity parameters towards zero, is designed, ensuring no output perturbation. Such a technique, when compared with other state-of-the-art pruning approaches, introduces a reduced computational overhead; however, it requires the network to be pre-trained.

In this work, we are overcoming this limitation introducing *loss-based sensitivity*, which is also beneficial for generalization and further reduces the computation overhead. A detailed discussion between the differences from the two formulations follows in Sec. 3.1.

3. Loss-driven sensitivity

In this section we are going to introduce our loss-driven sensitivity regularization. As a first step, we are going to discuss the state-of-the-art output-based sensitivity and its limitations, moving to out proposed regularization term (Sec. 3.1). A detailed analysis on the update rule (Sec. 3.2) follows.

3.1. From output-based sensitivity to loss-based sensitivity

A recent work introduced the concept of *sensitivity* of the output of the ANN model $\mathbf{y}$ to the variation of a given parameter:

$$S(\mathbf{y}, w_{n,i}) = \frac{1}{C} \sum_{k=1}^{C} \left| \frac{\partial y_k}{\partial w_{n,i}} \right| \tag{1}$$

where $w_{n,i}$ is the i-th parameter of the n-th layer. For instance, a high value of S results in a large variation of the output. If we assume that a *previously trained* ANN model yields optimal output, then we can target a sparser solution by removing parameters exhibiting low values for S and by keeping unmodified those with high S; towards this end, the following update rule at time t has been proposed to promote sparsity:

$$\begin{aligned} w_{n,i}^{t} := w_{n,i}^{t-1} - \eta \frac{\partial L}{\partial w_{n,i}^{t-1}} + \\ - \lambda w_{n,i}^{t-1} \left[1 - S(\mathbf{y}, w_{n,i}^{t-1})\right] P\left[S(\mathbf{y}, w_{n,i}^{t-1})\right] \end{aligned} \tag{2}$$

where

$$P(x) = \Theta\left[1 - |x|\right] \tag{3}$$

and $\Theta(\cdot)$ is the one-step function. The obvious limitation of (2) is that the output-based sensitivity (1) regularization term can not be applied at training time, as the outputs of the network might not be satisfactory.
For this reason, we need the sensitivity to be both:

- a measure of how much the output of the network is modified for small perturbations of a given parameter;

- a measure of how satisfactory the output of the network is.

We can modify (1) to this end by introducing a term which indicates whether the output of the network will be modified by the loss minimization:

$$S_L(L, \mathbf{y}, w_{n,i}) = \frac{1}{C}\sum_{k=1}^{C}\left|\frac{\partial L}{\partial y_k}\right| \cdot \left|\frac{\partial y_k}{\partial w_{n,i}}\right| \tag{4}$$

Substituting S with S_L in (2) the effect will be that a given parameter will be pushed towards zero both when the gradient of the loss is not willing to modify the output of the network *and* such a parameter is not relevant for the generation of the output.
However, computing (4) is extremely inefficient: nonetheless we can approximate it with the following lower bound

$$S_L \geq \left|\frac{\partial L}{\partial w_{n,i}}\right| \tag{5}$$

that simply neglects the effect of $w_{n,i}$ on the output. It is evident that the remaining term $\frac{\partial L}{\partial w_{n,i}}$ is classical loss gradient, necessary for any gradient-based optimization. Hence, we can conclude that (4) can be approximated by the proposed lower bound, i.e. the loss gradient: this simplification is computationally efficient and more importantly can be used to promote sparsification at training time.

3.2. Update rule

In the previous sub-section we have introduced a new measure for the sensitivity which, contrarily from the standard sensitivity measure in (1), can be used also at training time. In particular, plugging S_L in (2) we can rewrite the update rule as:

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

where

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

and where we assume η, λ two positive hyper-parameters. After some algebraic manipulations, we can equivalently rewrite (6) as

$$\begin{aligned} w_{n,i}^{t} = w_{n,i}^{t-1} - \lambda\Gamma\left(L, w_{n,i}^{t-1}\right) + \\ - \frac{\partial L}{\partial w_{n,i}^{t-1}}\left[\eta - \text{sign}\left(\frac{\partial L}{\partial w_{n,i}^{t-1}}\right)\lambda\Gamma\left(L, w_{n,i}^{t-1}\right)\right] \end{aligned} \tag{8}$$

In (8) we can distinguish two different contributions provided by the proposed regularization term:

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

$P\left(\frac{\partial L}{\partial w_{n,i}}\right)$	SIGN $\left(\frac{\partial L}{\partial w_{n,i}}\right)$	SIGN (w)	$\frac{\tilde{\eta}}{\eta}$
0	ANY	ANY	1
1	+	+	≤ 1
1	+	-	≥ 1
1	-	+	≥ 1
1	-	-	≤ 1

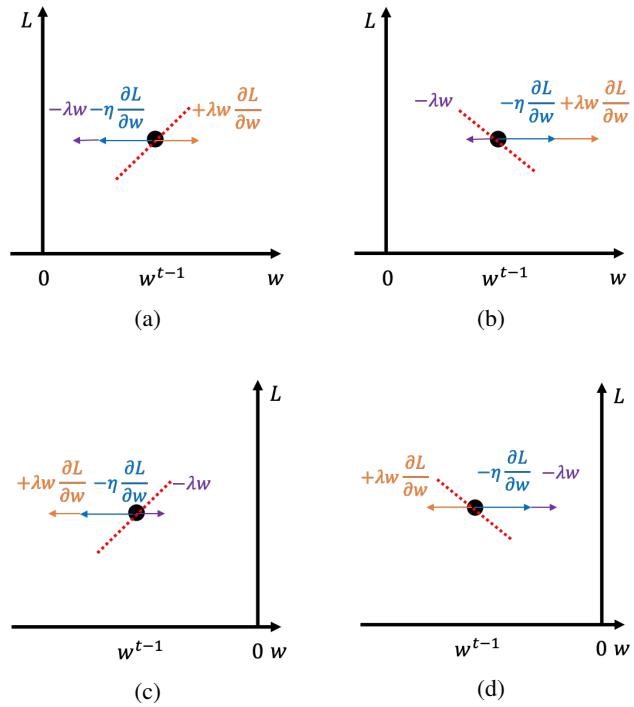

Figure 1. Update rule effect on the parameters. Red dotted is the tangent to the loss function in the black dot, in blue the standard SGD contribution, in purple the weight decay while in orange the LOBSTER contribution. Here we assume $P(L, w_{n,i}) = 1$.

- a weight decay-like term $\Gamma\left(L, w_{n,i}\right)$ 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

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

Assuming the hyper-parameters η, λ are positive, let us analyze the corrections in the learning rate. If $\left|\frac{\partial L}{\partial w_{n,i}}\right| \geq 1$ (highly sensitive parameter), then it follows that $P\left(\frac{\partial L}{\partial w_{n,i}}\right) = 0$ and $\Gamma\left(L, w_{n,i}\right) = 0$ and the dominant contribution comes from the gradient. Indeed, in this case

our update rule reduces to the classical GD:

$$w_{n,i}^{t} = w_{n,i}^{t-1} - \eta \frac{\partial L}{\partial w_{n,i}^{t-1}} \tag{10}$$

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

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

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 $\Gamma\left(L, w_{n,i}\right)$ 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 minimization tends at increasing the magnitude, the learning rate is increased, to compensate the contribution coming from $\Gamma\left(L, w_{n,i}\right)$. This mechanism allows us to actually *train* a deep model while introducing sparsity.

In the next section we are going to detail the overall training strategy, which cascades regularization and a *pruning* step.

4. Training procedure

In this section we are going to provide a general overview on how the training procedure is structured. Leveraging on the update rule, we can define an iterative procedure to prune ANNs. In particular, two phases represent the main core of the proposed training strategy:

- *regularization phase*, in which we regularize the ANN model according to the update rule (6),
- *pruning phase*, in which we prune some parameters.

Regularization phase lasts until we are reaching a plateau in the performance for PWE (we call it *plateau waiting epochs*) epochs. This measure is of course taken from a validation set we are sampling from the training set. Once we detect a plateau in the performance, we move to the pruning phase (more details on the pruning phase will be provided in Sec. 4.1). After this phase, we go back to the regularization phase.

4.1. Pruning phase

While the sensitivity-based regularization pushes less important parameters towards zero, a magnitude pruning step is required to actually remove them. Towards this end, the thresholding value T can be considered the variable we aim to maximize: in other words, we would like to find the highest T value such that the performance, evaluated as the loss on the validation set, worsens at most by a relative quantity we name *thresholding worsening tolerance* (TWT), from the un-pruned model. Once we find the optimal T, we apply a magnitude pruning on the parameters and we move back to the regularization phase. We would like to highlight that the proposed regularization pushes the parameters having low sensitivity towards zero, and then we perform magnitude pruning instead of "sensitivity ranking-based pruning" because the sensitivity is a *local* measure: if the parameter is not in the neighborhood of zero, such a measure can not be used to prune the parameter itself.

During testing, we found out that, because of the stochasticity introduced by mini-batch based optimizers, it is likely that parameters previously pruned might be re-introduced by the subsequent regularization step. To overcome such an effect, we have decided to make the pruning of a parameter permanent: during the process we are storing which parameters have been pruned and we never allow them to be re-introduced. We call this *parameter pinning*.

In the next section, some experiments will be presented, testing the effectiveness of the proposed training procedure.

5. Results

In this section we are going to test the effectiveness of LOBSTER. Towards this end, we have decided to experiment with different neural architectures and datasets commonly used as benchmark in the relevant literature; in particular, we focused on the following combination of networks and datasets:

- LeNet-300 on MNIST (Table 2),
- LeNet-5 on MNIST (Table 3),
- LeNet-5 on Fashion-MNIST (Table 4),
- ResNet-32 on CIFAR-10 (Table 5),
- ResNet-102 on ImageNet (Table 6).

To evaluate the performance of our method we measure the model sparsity versus the achieved classification error (indicated as Top-1 or Top-5). The network sparsity is defined as the ratio between the number of parameters in the

original network and the number of remaining parameters after sparsification (i.e. $\frac{|\theta|}{|\theta_{\neq 0}|}$, the higher the better), and it is also referred to as *compression ratio*. Our algorithms are implemented in Python, 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.[1]

5.1. LeNet-300 on MNIST

As a first test, we sparsified the well-known LeNet-300 (LeCun et al., 1998) architecture, which consists in three fully-connected layers with 300, 100 and 10 neurons respectively. We trained this 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 freshly 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 different groups in terms of achieved classification error: the first group is composed of methods whose error is around 1.65% while the second concerns methods with a classification error around 1.95%. Table 2 provides results for the proposed procedure with respect to both error ranges, separated by a horizontal line.

From the reported results we can observe that our method reaches a higher compression ratio than the the approaches found in literature. This is particularly noticeable in the 1.65% classification error group, where we almost double the sparsity of the second best method, from $27.87\times$ of (Tartaglione et al., 2018) to $50.04\times$ of our procedure. LOBSTER also achieves the highest result for the 1.95% error range, gaining especially in regards to the number of parameters removed from the first fully-connected layer (the largest, consisting of 235k parameters), in which just the 0.59% of the parameters survives.

5.2. LeNet-5 on MNIST

Next, we again experimented with the MNIST dataset, this time on the Caffe version of the LeNet-5 architecture, consisting in two convolutional and two fully-connected layers. Again, we used a freshly-initialized model, trained via SGD with learning rate $\eta = 0.1$, $\lambda = 10^{-4}$, $PWE = 20$ epochs and $TWT = 0.05$. The results are shown in Table 3.

Also with an architecture with convolutional layers, we are able to obtain a competitively small network with a compression ratio of $230\times$. The compression ratio approaches the state-of-the-art (Molchanov et al., 2017). Interestingly, LOBSTER is the approach that sparsifies the most the first layer (Conv1), while maintaining a competitive Top-1 error (DNS is the only technique that removes even more parameters in Conv1, but it allows a broader drop in performance and its compression ratio is non-competitive).

5.3. LeNet-5 on Fashion-MNIST

To scale-up the difficulty of the training task, we experimented on the classification of the Fashion-MNIST dataset (Xiao et al., 2017), using once more the LeNet5 network; 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 such a test we trained the network from scratch using SGD with $\eta = 0.1$, $\lambda = 10^{-4}$, $PWE = 20$ epochs and $TWT = 0.1$. The results of the test are shown in Table 4.

As expected, given the more challenging task, the achieved sparsity is lower than the one obtained in the MNIST case. Even then, the proposed method still reaches a higher compression ratio than (Tartaglione et al., 2018), removing an higher percentage of parameters, especially in the fully connected layers, while maintaining good generalization. In this case, the first layer is the least sparsified, and this is an effect of the higher complexity in extracting features from Fashion-MNIST: in such a sense, LOBSTER self-adapts to the complexity of the problem to be learned.

5.4. ResNet-32 on CIFAR-10

To evaluate how our method scales to deeper, modern architectures, we applied it on a PyTorch implementation[2] of the ResNet-32 network (He et al., 2015) that classifies the CIFAR-10 dataset. This dataset consists of 60k 32x32 RGB images divided in 10 classes, split in 50k training images and 10k test images.

For this test we started from the parameters configuration provided in the implementation's repository. We trained the network using SGD with momentum $\beta = 0.9$, $\lambda = 10^{-}6$, $PWE = 10$ and $TWT = 0$. To maximize both performance and compression, we employed a learning rate cycling strategy in which we cycled the value of the learning rate η between 10^{-2} and 10^{-3} every 300 epochs. The full training is performed for 11k epochs.

In Table 5 we show the results of this experiment, comparing it to Targeted Dropout (Gomez et al., 2019), Sparse Variational Dropout (Molchanov et al., 2017), l_0 regularization from (Louizos et al., 2017) and l_1 regularization proposed in (Han et al., 2015). To better show the effectiveness of

[1] The source code will be made available upon acceptance of the article.

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

Table 2. LeNet-300 trained on MNIST. Top: 1.65% error rate. Bottom: 1.95% error rate.

Approach	Remaining parameters(%)			$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}$	Top-1 (%)
	FC1	FC2	FC3		
Baseline	100	100	100	1×	**1.44**
(Han et al., 2016)	8	9	**26**	12.2×	1.6
(Tartaglione et al., 2018)	2.25	11.93	69.3	27.87×	1.65
LOBSTER	**1.46**	**5.29**	29.6	**50.04**×	1.65
(Louizos et al., 2017)	9.95	9.68	33	9.99×	1.8
Sparse VD (Molchanov et al., 2017)	1.1	2.7	38	68×	1.94
(Tartaglione et al., 2018)	0.93	**1.12**	5.9	103×	1.95
DNS (Guo et al., 2016)	1.8	1.8	**5.5**	56×	1.99
LOBSTER	**0.59**	2.53	18.3	**114.92**×	1.95

Table 3. LeNet-5 trained on MNIST.

Approach	Remaining parameters (%)				$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}$	Top-1 (%)
	Conv1	Conv2	FC1	FC2		
Baseline	100	100	100	100	1×	**0.68**
Sparse VD (Molchanov et al., 2017)	33	**2**	**0.2**	5	**280**×	0.75
(Han et al., 2016)	66	12	8	19	11.9×	0.77
(Tartaglione et al., 2018)	67.6	11.8	0.9	31.0	51.1×	0.78
DNS (Guo et al., 2016)	**14**	3	0.7	**4**	111×	0.91
SWS (Ullrich et al., 2019)	-	-	-	-	162×	0.97
(Louizos et al., 2017)	45	36	0.4	5	70×	1.00
LOBSTER	22	2.38	0.22	5.98	230×	0.79

our method, in Table 5 we report the achieved classification error for different compression ratios, corresponding to the pruning of about 40%, 60% and 80% of the original parameters.

As reported, our method performs particularly well on this task and outperforms other state-of-the-art techniques. Furthermore, with our procedure, we are able to improve the baseline classification error even when removing a significant portion of the network parameters (around 80%), going from a Top-1 error of 7.37% of the baseline to a 7.33% of the pruned model. 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.

5.5. ResNet-102 on ImageNet

Finally, we scale-up both the output and the complexity of the classification problem testing the proposed method on the ResNet-102 network over the well-known ImageNet dataset (ILSVRC-2012), composed of more than 1.2 million train images and 50k validation images, for a total of 1k classes. For this test we used SGD with momentum $\beta = 0.9$, $\lambda = 10^{-6}$ and $TWT = 0$. In this case we decided that, instead of waiting for the occurrence of a performance plateau, we would greedily perform the pruning step every time one fifth of the train set (around 7.9k iterations) is used in the regularization step. We again employ the cycling learning rate strategy between a learning rate $\eta_1 = 10^{-2}$ and $\eta_2 = 10^{-3}$. The epochs for η_1 as learning rate are fixed to 2, while the epochs for η_2 depends on the compression

Table 4. LeNet-5 trained on Fashion-MNIST.

Approach	Remaining parameters (%)				$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}$	Top-1 (%)
	Conv1	Conv2	FC1	FC2		
Baseline	100	100	100	100	$1\times$	**8.1**
(Tartaglione et al., 2018)	**76.2**	32.56	6.5	44.02	$11.74\times$	8.5
LOBSTER	78.6	**26.13**	**2.88**	**32.66**	$\mathbf{23.31}\times$	8.47

Table 5. ResNet-32 trained on CIFAR-10.

Approach	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert} = 1\times$ Top-1 (%)	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert} = 1.67\times$ Top-1 (%)	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert} = 2.5\times$ Top-1 (%)	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert} = 5\times$ Top-1 (%)
Baseline	**7.37**	-	-	-
Targeted Dropout (Gomez et al., 2019)	-	7.37	7.45	7.46
Variational Dropout (Molchanov et al., 2017)	-	7.88	8.52	16.56
l_0 (Louizos et al., 2017)	-	7.20	8.80	37.00
l_1 (Han et al., 2015)	-	14.68	30.52	76.29
LOBSTER	-	**7.16**	**7.29**	**7.33**

Table 6. ResNet-102 trained on ImageNet.

Approach	$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}=1\times$		$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}=1.67\times$		$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}=2.5\times$		$\frac{\lvert\theta\rvert}{\lvert\theta_{\neq 0}\rvert}=5\times$	
	Top-1(%)	Top-5(%)	Top-1(%)	Top-5(%)	Top-1(%)	Top-5(%)	Top-1(%)	Top-5(%)
Baseline	**22.63**	6.44	-	-	-	-	-	-
Targeted Dropout (Gomez et al., 2019)	-	-	26.50	-	49.50	-	99.60	-
l_1 (Han et al., 2015)	-	-	37.90	-	61.70	-	98.60	-
LOBSTER	-	-	**25.63**	7.81	**26.37**	8.42	**26.45**	8.24

ratio: when we detect a plateau on the compression ratio (in other words, when the regularization + pruning strategy is not pruning any further parameter), we increase the learning rate back to η_1 for 2 epochs. The full training lasts 95 epochs. Due to time constraints, we decided to use the pre-trained network offered by the torchvision library.[3] Notice that the validation set we are using to find the optimal T value also in this case is randomly sampled from the training set, and it does not contain *any* sample from the validation set provided by default in the dataset.

In Table 6 we show our results and compare them with (Gomez et al., 2019) Targeted Dropout and (Han et al., 2015) l_1 regularization. As in the previous section, we decided to report the compression bands corresponding to about 40%, 60% and 80% pruned parameters. Since an ImageNet test set is not publicly available, the classification errors are computed on the validation set. As explained in Sec. 4, during our procedure, we employ a validation set to perform the pruning stage. Using the actual ImageNet validation set may lead to over-fitting the images on which we evaluated the model performance; to avoid this issue, we manually built another validation set by sampling 5 elements of the train set for each class, resulting in a total of 5k images, and we used it in the pruning step.

This approach has proven to be particularly efficient since we are able to remove the 80% of the total parameters while also maintaining good generalization with a Top-1 error of 26.45% and a Top-5 error of 8.24%. These results are achieved on a complex architecture like ResNet-102 trained on a complex dataset like ImageNet, showing the broad effectiveness of LOBSTER. Contrarily from what observed in Sec. 5.4, in this case we are not observing an improvement in the baseline performance. We believe this is an effect of the greedy pruning strategy adopted for such a huge dataset.

6. Conclusion

In this work we have proposed LOBSTER, a new regularization strategy aiming to train sparse ANNs. Differently from other fine-tuning strategies, LOBSTER can be directly used at training time with no significant performance loss. Furthermore, as it directly uses the gradients coming from back-propagation, no significant computational overhead is introduced, with a computational complexity comparable to weight-decay regularization strategy. Differently from L2 regularization, LOBSTER is aware of the global contribution of the parameter 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 compression ratios for small architectures like LeNet-300 and LeNet-5 while it proves its effectiveness in sparsifying deep architectures, like ResNet-32 trained on CIFAR-10 and ResNet-102, trained on a more complex task like ImageNet: in these cases, LOBSTER significantly improves the current state-of-the-art results.

Future research includes the extension of LOBSTER to achieve a more structured sparsity and a study on the memory footprint of the pruned models.

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

References

Badrinarayanan, V., Kendall, A., and Cipolla, R. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

Belilovsky, E., Eickenberg, M., and Oyallon, E. Greedy layerwise learning can scale to imagenet. volume 2019-June, pp. 911–925, 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85077961538&partnerID=40&md5=1b6ad4b8f6abbe6a527d0ef9e9f54c6f. cited By 0.

Brutzkus, A., Globerson, A., Malach, E., and Shalev-Shwartz, S. Sgd learns over-parameterized networks that provably generalize on linearly separable data. 2018. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85061601180&partnerID=40&md5=47c7d44ed33d4afa1a4d39b8757650ea. cited By 14.

Engelbrecht, A. P. and Cloete, I. A sensitivity analysis algorithm for pruning feedforward neural networks. In *Neural Networks, 1996., IEEE International Conference on*, volume 2, pp. 1274–1278. IEEE, 1996.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85069453436&partnerID=40&md5=fd1a2b2384d79f66a49cc838a76343d3. cited By 8.

Gomez, A. N., Zhang, I., Swersky, K., Gal, Y., and Hinton, G. E. Learning sparse networks using targeted dropout.

CoRR, abs/1905.13678, 2019. URL http://arxiv.org/abs/1905.13678.

Greaves-Tunnell, A. and Harchaoui, Z. A statistical investigation of long memory in language and music. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 2394–2403, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/greaves-tunnell19a.html.

Groetsch, C. W. *Inverse Problems in the Mathematical Sciences*. Vieweg, 1993.

Guo, Y., Yao, A., and Chen, Y. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pp. 1379–1387, 2016.

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

Han, S., Mao, H., and Dally, W. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. 2016. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85060057269&partnerID=40&md5=0ea8db1c541d1c9abf366149fa4aa7cb. cited By 525.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

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

Li, X., Chen, H., Qi, X., Dou, Q., Fu, C.-W., and Heng, P.-A. H-denseunet: hybrid densely connected unet for liver and tumor segmentation from ct volumes. *IEEE transactions on medical imaging*, 37(12):2663–2674, 2018.

Louizos, C., Welling, M., and Kingma, D. P. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

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

Molchanov, D., Ashukha, A., and Vetrov, D. Variational dropout sparsifies deep neural networks. volume 5, pp. 3854–3863, 2017. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85048506601&partnerID=40&md5=c352a4786ef977ccea7e397bd7469f14. cited By 29.

Mrazova, I. and Kukacka, M. Can deep neural networks discover meaningful pattern features? *Procedia Computer Science*, 12:194–199, 2012.

Mrázová, I. and Reitermanová, Z. A new sensitivity-based pruning technique for feed-forward neural networks that improves generalization. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pp. 2143–2150. IEEE, 2011.

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. Do imagenet classifiers generalize to imagenet? volume 2019-June, pp. 9413–9424, 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85078277942&partnerID=40&md5=9654692723fc1784285d124343fb26c1. cited By 0.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

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

Tartaglione, E., Perlo, D., and Grangetto, M. Post-synaptic potential regularization has potential. In *International Conference on Artificial Neural Networks*, pp. 187–200. Springer, 2019.

Ullrich, K., Welling, M., and Meeds, E. Soft weight-sharing for neural network compression. 2019. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85071003624&partnerID=40&md5=dc00c36113f775ff4a6978b86543814d. cited By 2.

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

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017. URL http://arxiv.org/abs/1708.07747.

Zhu, M. and Gupta, S. To prune, or not to prune: Exploring the efficacy of pruning for model compression. 2018. URL https://www.scopus.com/inward/record.uri?eid=2-s2.0-85070986022&partnerID=40&

`md5=88c06ed239deec99b2e18925921c611f`.
cited By 1.