# A round-trip journey in pruned artificial neural networks

Andrea Bragagnolo[1,*], Enzo Tartaglione[2], Gianluca Dalmasso[1,3] and Marco Grangetto[3]

[1]*Synesthesia s.r.l, Turin, Italy*
[2]*LTCI, Télécom Paris, Institut Polytechnique de Paris, France*
[3]*Computer Science Dept., University of Turin, Italy*

**Abstract**
In the last decade, deep learning models competed for performance at the price of tremendous computational costs. Such a critical aspect recently attracted more attention for both the training and inference phases. The latter is obviously orders of magnitude lower than the training complexity, but on the other hand, it contributes many times, which impacts efficiency on edge or embedded devices. Inference can be made efficient through neural network pruning, which consists of parameters and neurons' removal from the model's topology while maintaining the model's accuracy. This results in reduced resource and energy requirements for the models. This paper describes two pruning procedures for lowering the operations required during the inference phase and a method to exploit the resulting sparsity. The same cannot be applied at training time: we show it is possible to borrow similar ideas to reduce the cost of gradient backpropagation by disabling the computation for selected neurons.

**Keywords**
Deep Learning, Pruning, Efficiency

## 1. Introduction

To achieve state-of-the-art performance, deep neural networks are widely used in various tasks, such as speech recognition and computer vision. However, modern architectures require many parameters to generalize well, resulting in large model sizes, high computational and memory resources, and significant energy consumption during training and inference.

In this paper, we present our research on neural network pruning, which involves removing the less essential elements of the network to reduce the model resource requirements. Specifically, we explore the design of pruning procedures (Sec. 2 and Sec. 3), the effect of pruning on network features (Sec. 4), and the practical application of pruned networks to reduce energy consumption (Sec. 5).

We present two pruning techniques capable of squeezing the model size incrementally during training: LOBSTER [1], an unstructured approach that uses parameter sensitivity as a regularizer, and SeReNe [2], a structured procedure that evaluates the contribution of neurons to the network's output. Pruned networks obtained with these techniques were used to assess the benefits of pruning at inference time.

To reduce the computational resources required to train neural networks, we introduce NEq [3], a technique to disable the computation of gradients of neurons that have reached equilibrium: this amounts to pruning the backpropagation graph and decreasing the number of operations during training. This technique can reduce the cost of training modern neural networks.

Our results demonstrate that neural network pruning can significantly reduce the model size, computational and memory resources, and energy consumption, while maintaining or even improving performance. Our approach can be used to develop energy-efficient neural networks, making them more sustainable and applicable in real-world scenarios where energy consumption is a critical factor.

## 2. LOBSTER

In this section, we present LOBSTER [1] (LOss-Based SensiTivity rEgulaRization), an unstructured and gradual pruning procedure.

LOBSTER uses a sensitivity-based regularization to promote sparsity in the network topology. Specifically, we define the sensitivity of a network parameter as the derivative of the loss function with respect to that parameter. Parameters with low sensitivity have little impact on the loss function when perturbed and can be pruned without compromising performance. LOBSTER achieves sparsity by gradually shrinking parameters with low sensitivity using a regularize-and-prune approach. The sensitivity is defined as

*Ital-IA 2023: 3rd National Conference on Artificial Intelligence, organized by CINI, May 29–31, 2023, Pisa, Italy*
*Corresponding author.
andrea.bragagnolo@synesthesia.it (A. Bragagnolo); enzo.tartaglione@telecom-paris.fr (E. Tartaglione); gianluca.dalmasso@unito.it (G. Dalmasso); marco.grangetto@unito.it (M. Grangetto)
0000-0002-8619-1586 (A. Bragagnolo); 0000-0003-4274-8298 (E. Tartaglione); 0000-0002-2709-7864 (M. Grangetto)

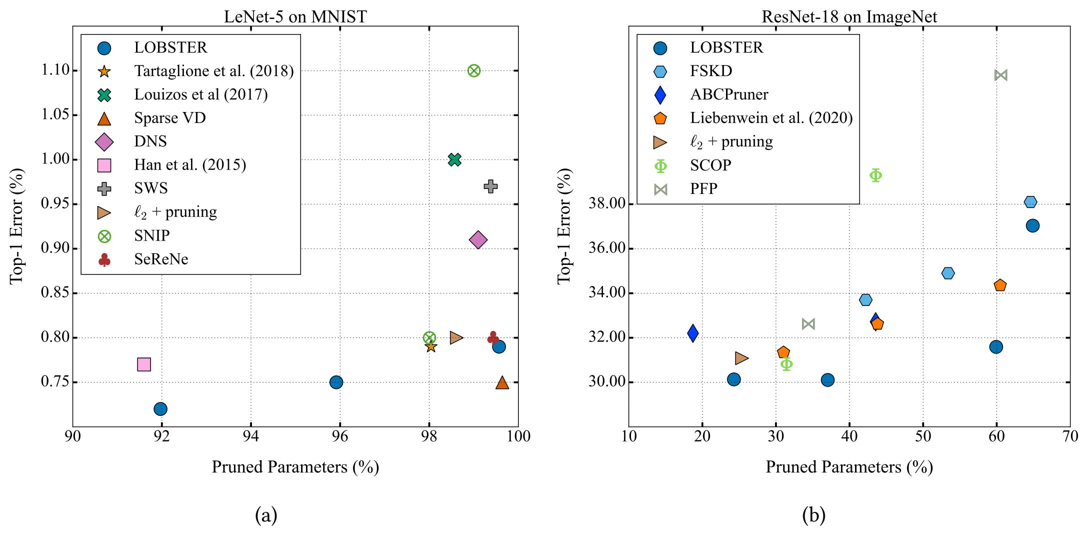

**Figure 1:** Performance (Top-1 error) vs. ratio of pruned parameters for LOBSTER and other state-of-the-art methods over different architectures and datasets. Source: [1].

$$S(\mathscr{L}, w_{n,i,j}) = \left|\frac{\partial \mathscr{L}}{\partial w_{n,i,j}}\right|, \tag{1}$$

with $\mathscr{L}$ representing the loss function and $w_{n,i,j}$ a parameter of the network.

LOBSTER allows training a network from scratch, thanks to its loss-based sensitivity formulation. Moreover, it avoids additional derivative computations or second-order derivatives, unlike other sensitivity-based approaches.

Experiments on multiple architectures and datasets demonstrate that LOBSTER outperforms several competitors in multiple tasks. It achieves competitive compression ratios with minimal computational overhead and without compromising performance. The results of the pruning procedure for LeNet-5 trained on the MNIST dataset and ResNet-18 trained on ImageNet are shown in Figure 1. LOBSTER achieves state-of-the-art sparsification and classification errors for both architectures. Sparse VD [4] slightly outperforms all other methods in the LeNet5-MNIST experiment at higher compression rates.

## 3. SeReNe

Although LOBSTER can achieve high sparsity rates, the sparsity is unstructured, meaning that the architecture may not remove entire neurons, and the resulting model can only be accelerated using specialized hardware and software.

SeReNe [2] solves this issue by producing sparse network topologies with a structure, hence consisting of fewer neurons and, therefore, fewer operations during inference. Our approach involves driving all the parameters of a neuron toward zero, allowing us to prune entire neurons from the network. To achieve this, we leverage the concept of neuron's sensitivity, defined as the variation of the network output with respect to the neuron's activity:

$$S_{n,i}(\boldsymbol{y}_N, p_{n,i}) = \frac{1}{C}\sum_{k=1}^{C}\left|\frac{\partial y_{N,k}}{\partial p_{n,i}}\right|, \tag{2}$$

where $\boldsymbol{y}_N$ represents the network's output and $p_{n,i}$ the $n$-th neuron of the $i$-th layer activity.

During training, all the parameters of low-sensitivity neurons are shrunk, making it possible to remove them from the network. When the $\ell_2$ norm of a neuron's parameters approach zero, the neuron no longer emits signals (except for the bias), and can be pruned. We propose an iterative two-step procedure to prune parameters belonging to low-sensitivity neurons. We ensure controlled performance loss for the original architecture using a cross-validation strategy.

Our approach allows us to learn network topologies that are not only sparse, i.e., with few non-zero parameters, but also with fewer neurons. This can speed-up the network execution by better using cache locality and memory access patterns. We demonstrate the effectiveness of SeReNe on multiple learning tasks and network architectures, outperforming state-of-the-art references. Finally, we show that structured sparsity provides ben-

**Table 1**
LeNet-5 trained on MNIST. Source: [2].

| Approach | Remaining parameters (%) | | | | Compr. | Neurons | Network size [kB] | | | Top-1 |
|---|---|---|---|---|---|---|---|---|---|---|
| | Conv1 | Conv2 | FC1 | FC2 | ratio | | `.onnx` | | `.7z` | (%) |
| Baseline | 100 | 100 | 100 | 100 | 1x | [20]-[50]-[500]-[10] | 1686 | → | 1510 | **0.68** |
| Sparse VD [4] | 33 | **2** | **0.2** | 5 | **280x** | - | | - | | 0.75 |
| Han *et al.* [5] | 66 | 12 | 8 | 19 | 11.9x | - | | - | | 0.77 |
| SWS [6] | - | - | - | - | 162x | - | | - | | 0.97 |
| Tartaglione *et al.* [7] | 67.6 | 11.8 | 0.9 | 31.0 | 51.1x | [20]-[48]-[344]-[10] | | - | | 0.78 |
| DNS [8] | **14** | 3 | 0.7 | **4** | 111x | - | | - | | 0.91 |
| $\ell_2$+pruning | 60.20 | 7.37 | 0.61 | 22.14 | 72.3 | [19]-[37]-[214]-[10] | 577 | → | 46 | 0.8 |
| SeReNe | 33.75 | 3.25 | 0.27 | 10.22 | 177.05x | **[11]-[26]-[113]-[10]** | **208** | → | **19** | 0.8 |

**Table 2**
ResNet-101 trained on ImageNet. Source: [2].

| Approach | Remaining parameters (%) [neurons] | | | | | | Compr. | Network size [MB] | | | Top-1 | Top-5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Conv1 | Block1 | Block2 | Block3 | Block4 | FC1 | ratio | `.onnx` | | `.7z` | (%) | (%) |
| Baseline | 100 | 100 | 100 | 100 | 100 | 100 | 1x | 174.49 | → | 156.67 | **22.63** | **6.44** |
| | [64] | [1408] | [3584] | [36352] | [11264] | [1000] | | | | | | |
| Sparse VD [4] | - | - | - | - | - | - | 2.48x | | - | | 35.76 | 13.45 |
| $\ell_2$+pruning | **53.12** | 25.42 | 25.57 | 13.71 | 17.74 | 51.94 | 5.75x | 172.94 | → | 32.93 | 28.33 | 9.18 |
| | [49] | [1241] | [3280] | [33278] | [11250] | [1000] | | | | | | |
| SeReNe | 55.36 | **24.27** | **23.79** | **11.24** | **14.81** | **40.82** | **6.94**x | **172.15** | → | **27.84** | 28.41 | 9.45 |
| | [49] | [**1197**] | [**3142**] | [**31948**] | [**11249**] | [1000] | | | | | | |

efits when storing the neural network topology and parameters. Table 1 shows the results obtained applying SeReNe on the LeNet-5 architecture trained on MNIST. SeReNe achieves a high compression ratio and pruned neurons, outperforming the considered references. The structured sparsity results in a significant decrease in the uncompressed network storage footprint, with only a slight 0.12% performance drop after compression. We also tested our method on more challenging architectures and datasets: Table 2 shows the results for ResNet-101 trained on ImageNet. The pruning procedure results in around 86% of the parameters being pruned, and the resulting network size is reduced from 156.67 MB to only 27.84 MB.

## 4. Structured pruning for low-power devices

In this section, we present some empirical results that demonstrate how pruning (especially structured pruning) can produce a network model that requires fewer resources to perform inference. To achieve this, we built the `simplify` library [9], a PyTorch-compatible tool that automates the process of optimizing the inference code for pruned neural networks by removing the zeroed neurons from the architecture. The resulting models do not require any particular software or hardware to speed up their inference.

We were able to perform benchmarks for both mobile devices [10] and FPGA platforms [11], which demonstrates the effectiveness of our approach. Specifically, we evaluated the performance of the pruned neural networks on a range of devices, varying in terms of processing power and memory capacity. Our results show that the combination of pruning and Simplify optimization outperforms the other techniques in terms of both inference speed and memory footprint. Table 3 and Table 4 shows the results for pruned and simplified network on mobile devices and FPGAs respectively.

Overall, these results demonstrate the feasibility of deploying pruned neural networks on various resource-constrained devices, opening up new opportunities for bringing deep learning to the edge.

## 5. Neurons at Equilibrium (NEq)

All the works presented up to this point focused on reducing the neural network's inference time. However, in this section, we present NEq [3], an approach that enables us to shrink the cost of training by reducing the number of

**Table 3**
Experimental results for different network architectures and pruning strategies. Left: percentage of pruned parameters, size of the simplified network topology, and size of the compressed bitstream. Right: inference time on different embedded devices: Raspberry Pi 3B (RPi 3B), Huawei P20 (P20), Xiaomi MI 9 (MI9), and Samsung Galaxy S6 lite (S6L). Source: [10].

| **Dataset** | **Architecture** | **Pruning** | **Pruning ratio [%]** | **Simplified topology [MB]** | **Compressed bitstream [MB]** | **Inference time [ms]** | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | RPi 3B | P20 | MI9 | S6L |
| CIFAR-10 | VGG-16 | No pruning | - | 60.0 | 3.6 | 647 | 204 | 153 | 251 |
| | | LOBSTER | **92.44** | 58.61 | 1.61 | 610 | 191 | 146 | 242 |
| | | SeReNe | 47.16 | **31.02** | **0.34** | **594** | **99** | **85** | **106** |
| | ResNet-32 | No pruning | - | 2.0 | 0.30 | 580 | 32 | 30 | 31 |
| | | LOBSTER | **81.19** | 1.96 | 0.12 | 545 | 32 | 26 | 30 |
| | | SeReNe | 52.80 | **1.0** | **0.09** | **536** | **25** | **17** | **25** |
| CIFAR-100 | AlexNet | No pruning | - | 94.6 | 10.1 | 246 | 131 | 84 | 168 |
| | | LOBSTER | **98.90** | 48.84 | 0.40 | 224 | 95 | 67 | 120 |
| | | SeReNe | 59.87 | **37.07** | **0.20** | **186** | **75** | **53** | **96** |
| ImageNet | ResNet-101 | No pruning | - | 178.4 | 26.24 | 11919 | 958 | 416 | 1008 |
| | | LOBSTER | **87.39** | 173.87 | 9.24 | 11879 | 956 | 403 | 985 |
| | | SeReNe | 1.09 | **172.53** | **7.51** | **11699** | **929** | **371** | **974** |

**Table 4**
Inference time, FPS, and speedup for ISIC classification and segmentation models for CPU and FPGA. Source: [11].

| **Task** | **Model** | **Device** | **Inference time [ms]** | **FPS** | **Speedup** |
|---|---|---|---|---|---|
| Classification | VGG16 | CPU | 1430.29 | 0.70 | - |
| | VGG16 | FPGA | 552.28 | 1.81 | 2.59× |
| | VGG16-Quantized | FPGA | 229.93 | 4.35 | 6.22× |
| | VGG16-HA | FPGA | 99.33 | 10.07 | 14.40× |
| | VGG16-HCR | FPGA | 15.09 | 66.25 | 94.76× |
| Segmentation | SegNet | CPU | 3165.97 | 0.32 | - |
| | SegNet | FPGA | 757.22 | 1.32 | 4.18× |
| | SegNet-Pruned | FPGA | 282.23 | 3.54 | 11.22× |

operations of the backpropagation algorithm and the optimizer: if a neuron has achieved equilibrium, it does not require weights update. Unlike pruning techniques, NEq does not remove neurons from the model's architecture: it only prevents unnecessary weight updates, reducing the number of operations performed by the optimizer and by back-propagation. Our approach targets the reduction of computational complexity during training.

To determine if a neuron reaches its equilibrium, we evaluate its velocity term as

$$v^t_{\Delta\phi_i} = \begin{cases} \phi_i^t + \sum_{\alpha=1}^{t} (-1)^\alpha \left[(\mu_{eq})^{\alpha-1} + (\mu_{eq})^{\alpha}\right] \phi_i^{t-\alpha} & \mu_{eq} \neq 0 \\ \phi_i^t - \phi_i^{t-1} & \mu_{eq} = 0, \end{cases} \tag{3}$$

where $\phi_i^t = \sum_{\xi\in\Xi_{val}} \sum_{m=1}^{M_i} \hat{y}^t_{n,i,m,\xi} \cdot \hat{y}^{t-1}_{n,i,m,\xi}$ is the cosine similarity between all the outputs of the $i$-th neuron at time $t$ and at time $t-1$ for the whole validation set $\Xi_{val}$. We can say that the $i$-th neuron is at equilibrium when it can satisfy $\left|v^t_{\Delta\phi}\right| < \varepsilon$ for some $\varepsilon \geq 0$.

Table 5 presents the results of some of our experiments, where we compare NEq with a "stochastic" approach that randomly halts the update of a neuron at every epoch with some probability $p$. We test three different probabilities: 0.2, 0.5, and a probability that is as close as possible to the average achieved by NEq, denoted with an asterisk (*). We evaluate the effectiveness of our approach by analyzing the average computational complexity of the backpropagation for a single update iteration, expressed in FLOPs, and the network's generalization capability at the end of training. Our results show that NEq consistently reduces the number of FLOPs with minimal or no performance drop. While the amount of saved computation is similar for the stochastic approach with fixed probabilities in all the considered scenarios, the loss in performance varies depending on the architecture and dataset. In contrast, NEq adapts to the particular setup and saves the largest FLOPs for a given performance, with a lower performance loss even when the stochastic approach is tested with the same FLOPs saving.

**Table 5**
Results of the application of NEq, compared to the stochastic approach. We report the average FLOPs per iteration at backpropagation, and the final performance of the model evaluated on the test set (values annotated with † report the classification accuracy, values annotated with ‡ report the mean IoU). Source: [3].

| Dataset | Model | Approach | Bprop. FLOPs per iteration | Performance |
|---|---|---|---|---|
| ImageNet-1K | ResNet-18 | Baseline | 3.64G ± 0.0G | 69.90% ± 0.04%† |
| | | Stochastic ($p = 0.2$) | 2.94G ± 0.00G (-19.26%) | 69.42% ± 0.16% (-0.48%)† |
| | | Stochastic ($p = 0.5$) | 1.85G ± 0.00G (-49.11%) | 69.18% ± 0.03% (-0.72%)† |
| | | Stochastic* | 2.82G ± 0.00G (-22.58%) | 69.45% ± 0.06% (-0.45%)† |
| | | Neq | 2.80G ± 0.03G (-23.08%) | 69.62% ± 0.06% (-0.28%)† |
| COCO | DeepLabv3 | Baseline | 305.06G ± 0.0G | 67.71% ± 0.02%‡ |
| | | Stochastic ($p = 0.2$) | 248.69G ± 0.00G (-18.48%) | 67.11% ± 0.02% (-0.60%)‡ |
| | | Stochastic ($p = 0.5$) | 163.42G ± 0.00G (-46.43%) | 66.91% ± 0.04% (-0.80%)‡ |
| | | Stochastic* | 229.00G ± 0.00G (-24.93%) | 67.02% ± 0.03% (-0.69%)‡ |
| | | Neq | 217.29G ± 0.04G (-28.77%) | 67.22% ± 0.04% (-0.49%)‡ |

## 6. Conclusion

In this paper, we shared the research experiences we developed in the context of compressing large neural models. Our story has begun with classical unstructured pruning of model parameters, e.g. connections between neurons, where the target is the highest sparsification with the lowest performance impairment. This approach, while very sound from a theoretical point of view, does not guarantee significant efficiencing of the inference phase, when the model is deployed on actual devices. Therefore, we described the structured pruning alternatives that aim at removing whole neurons, thus uncovering the real pruning potential in saving memory and reducing the latency. Finally, we show that pruning can also be exploited at training time to cut the cost of backward propagation. In particular, we introduced NEq, a technique to disable the computation of gradients of neurons that have reached equilibrium: this amounts to pruning the backpropagation graph, and decreasing the number of operations during training. This technique can reduce the cost of training modern neural networks.

## References

[1] E. Tartaglione, A. Bragagnolo, A. Fiandrotti, M. Grangetto, Loss-based sensitivity regularization: Towards deep sparse neural networks, Neural Networks 146 (2022) 230–237. URL: https://www.sciencedirect.com/science/article/pii/S0893608021004706. doi:https://doi.org/10.1016/j.neunet.2021.11.029.

[2] 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 33 (2022) 7237–7250. doi:10.1109/TNNLS.2021.3084527.

[3] A. Bragagnolo, E. Tartaglione, M. Grangetto, To update or not to update? neurons at equilibrium in deep models, in: A. H. Oh, A. Agarwal, D. Belgrave, K. Cho (Eds.), Advances in Neural Information Processing Systems, 2022. URL: https://openreview.net/forum?id=LGDfv0U7MJR.

[4] D. Molchanov, A. Ashukha, D. Vetrov, Variational dropout sparsifies deep neural networks, in: Proceedings of the 34th International Conference on Machine Learning-Volume 70, JMLR. org, 2017, pp. 2498–2507.

[5] 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.

[6] K. Ullrich, M. Welling, E. Meeds, Soft weight-sharing for neural network compression, 5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings (2019).

[7] 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.

[8] Y. Guo, A. Yao, Y. Chen, Dynamic network surgery for efficient dnns, Advances in Neural Information Processing Systems (2016) 1387–1395.

[9] A. Bragagnolo, C. A. Barbano, Simplify: A python library for optimizing pruned neural networks, SoftwareX 17 (2022) 100907. URL: https://www.sciencedirect.com/science/article/pii/S2352711021001576. doi:https://doi.org/10.1016/j.softx.2021.100907.

[10] A. Bragagnolo, E. Tartaglione, A. Fiandrotti, M. Grangetto, On the role of structured pruning

for neural network compression, in: 2021 IEEE International Conference on Image Processing (ICIP), IEEE, 2021, pp. 3527–3531.

[11] J. Flich, L. Medina, I. Catalán, C. Hernández, A. Bragagnolo, F. Auzanneau, D. Briand, Efficient inference of image-based neural network models in reconfigurable systems with pruning and quantization, in: 2022 IEEE International Conference on Image Processing (ICIP), 2022, pp. 2491–2495. doi:10.1109/ICIP46576.2022.9897752.