EXPLORING THE EFFECT OF ℓ_0/ℓ_2 REGULARIZATION IN NEURAL NETWORK PRUNING USING THE LC TOOLKIT

Yerlan Idelbayev *Miguel Á. Carreira-Perpiñán*

Dept. of Computer Science and Engineering, University of California, Merced

{yidelbayev, mcarreira-perpinan}@ucmerced.edu

http://eecs.ucmerced.edu

ABSTRACT

The LC Toolkit is an open-source library written in Python and PyTorch that allows to compress any neural network using several compressions including quantization, pruning, and low-rank. The versatility of the framework is rooted in the principled mathematical formulation of the underlying network compression problems with subsequent optimization by learning-compression (LC) algorithm. In this paper, we utilize the LC toolkit's common algorithmic base to take a deeper look into ℓ_0-constrained pruning problems defined as follows: given a budget of κ non-zero weights, which weights should be pruned in the final network? We observe that ℓ_0-pruned networks have a different connectivity structure compared to pruning results using ℓ_1 norm. We propose a change to the formulation of the problem involving a small amount of ℓ_2 weight decay which has a favorable effect on connectivity structure. We study the properties of the proposed $\ell_0 + \ell_2$ formulation using the LC toolkit and empirically demonstrate that such a scheme achieves a competitive sparsity-error tradeoff while having better structural sparsity.

Index Terms— neural network compression toolkits, weight pruning

1. INTRODUCTION

Neural networks have become an important computational solution to day-to-day practical applications involving many machine learning tasks in computer vision, natural language understanding, signal processing and other fields. As the complexity and task accuracy of the neural networks has increased over the years, so have the sizes and computational demands of these models. Therefore, the problem of compressing large neural networks to reduce their computational, storage, and power demands has gained significant interest in the community. Although many papers have been published in the last years discussing specialized algorithms, there is a need for a common framework to study these compression problems theoretically and practically.

In this paper we study the problem of neural network pruning using the algorithmic footing of the learning-compression (LC) algorithm and its open-source PyTorch-based implementation: the LC toolkit [1], which is available on Github[1]. At the current stage, the LC toolkit includes compressions like quantization [2], pruning [3], low-rank [4], and the additive [5] and mix-and-match combinations of those; with the functionality of targeting the device-specific constraints [6]. The toolkit is extensible by design and allows us to add a new, user defined compression type with minimal work. Mathematically, this is achieved by a generic formulation of the model compression as a constrained optimization problem using auxiliary variables and alternating optimization solution that separates model learning (L step) from model compression (C step). *Decoupling of the learning and compression steps in the LC algorithm comes quite handy not only in the practical application of the framework, but is also useful for a deeper theoretical understanding of the compressions themselves as we demonstrate in the example of ℓ_0-pruning.*

[1]https://github.com/UCMerced-ML/LC-model-compression

Method	Test err	Remaining weights	Remaining neurons
reference	2.01%	100%	784-300-100-10
ℓ_0	2.33%	2.0%	344-210-100-10
ℓ_1	2.44%	2.4%	476-37-84-10
$\ell_0 + \ell_2$ (ours)	**1.76%**	2.0%	420-70-82-10

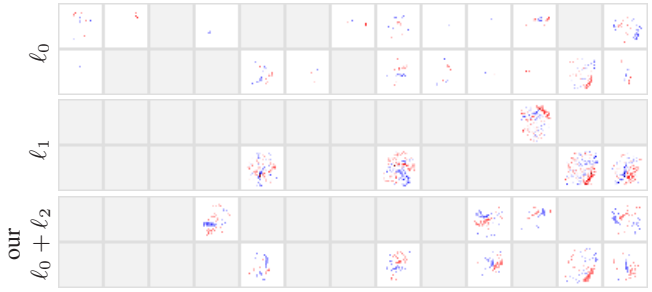


Fig. 1. *Top:* Pruning of LeNet300 with ℓ_0- and ℓ_1-constrained formulation, and with a proposed scheme of $\ell_0 + \ell_2$. We set $\kappa = 2\%$ for ℓ_0 methods, and for ℓ_1 we chose the parameters to yield approximately 2%-sparse net. *Even though ℓ_0- and ℓ_1-pruned networks have similar sparsities, the connectivity structure is very different*: with ℓ_0 pruning, we observe many alive neurons, whereas the ℓ_1-pruned network has few, but relatively dense neurons. With our proposed method of $\ell_0 + \ell_2$ regularization we achieve a much fewer number of alive neurons across layers when compared to the pure ℓ_0. *Bottom:* Visualization of the same subset of the first layer neurons corresponding to the networks in the table. Each small square is a single neuron looking at an input of 28×28; colors represent the following: white (pruned weights), red (positive weights), blue (negative weights), gray (completely dead neuron, i.e., all weights are pruned).

2. SETUP OF THE PRUNING PROBLEM

Due to overparametrization of the neural networks, often a significant portion of the weights can be removed with a minimal impact to the network's task loss. This process is known as the network

pruning and can be formulated in various ways. Given a network with weights $\mathbf{w}$ and task loss L, one possible formulation of pruning is to use a sparsifying penalty $P(\mathbf{w})$ and solve a regularized or constrained formulation given as

$$\min_{\mathbf{w}} L(\mathbf{w}) + \lambda P(\mathbf{w}) \quad \text{or} \quad \min_{\mathbf{w}} L(\mathbf{w}) \text{ s.t. } P(\mathbf{w}) \leq \lambda \tag{1}$$

for some $\lambda > 0$. A usual choice for the pruning cost $P(\mathbf{w})$ is to use a convex sparsity-inducing norm like ℓ_1 or $\ell_{1,2}$. Yet, a downside of such formulation is the necessity to manually tune the strength of the penalty as it is unclear which value of λ will yield a network with a required sparsity.

A more direct way of formulating the pruning problem is by precisely setting the total amount of allowed non-zero weights. Such pruning problem is defined using the ℓ_0-constraints as

$$\min_{\mathbf{w}} \quad L(\mathbf{w}) \quad \text{s.t.} \quad \|\mathbf{w}\|_0 \leq \kappa, \tag{2}$$

where κ is the total budget of the remaining parameters. The formulation (2) is a particular case of the constrained problem in equation (1) with $P(\mathbf{w}) = \|\mathbf{w}\|_0$. However, compared to a general pruning formulation, the problem (2) has several advantages: 1) it eliminates cumbersome trial-and-error process of tuning the hyperparameter λ to achieve a desired sparsity level (e.g., what value of λ results in 1% sparsity?) and 2) it allows to fit the network into the exact budget dictated by the hardware constraints, e.g., memory budget or sparsity constraint. For instance, NVIDIA's recent Ampere GPUs can efficiently run the networks that has exactly 2 zeros in every block of 4 consecutive weights [7], which is a particular case of eq. (2).

In principle, for a given level of sparsity the formulation (1) with $P(\mathbf{w}) = \|\mathbf{w}\|_1$ should yield more or less the same networks as the ℓ_0 formulation (2). However, we empirically observe that this is not the case. To illustrate this phenomenon, we used the LC toolkit to reduce the LeNet300 network to the sparsity of $\approx 2\%$ using ℓ_0- and ℓ_1-constrained formulations (see Fig. 1). *Although both networks achieve about the same accuracy and sparsity, we notice that connectivity structure is drastically different.* For example, the ℓ_1-pruned LeNet300 has only 37 alive neurons in the first layer (out of 300), and these neurons are relatively dense; in contrast the ℓ_0-pruned LeNet300 has 210 alive neurons in the same layer. Upon further inspection, we observed that the difference in the number of alive neurons comes from the fact that the ℓ_0-pruned network has many neurons with few small weights which seemingly do not perform any meaningful work. This observation leads us to the following question: can we change the constrained ℓ_0 formulation (2) to result into a pruned network with at most κ non-zero weights, but having a nicer connectivity structure, like with ℓ_1?

We propose modifying the problem (2) by adding a small amount of ℓ_2 weight decay as follows:

$$\min_{\mathbf{w}} \quad L(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \quad \text{s.t.} \quad \|\mathbf{w}\|_0 \leq \kappa, \tag{3}$$

with the motivation that ℓ_2 penalty will nudge the small valued weights towards zero and allow more neurons to be pruned. We study our modified formulation using the framework of the LC algorithm and demonstrate that: 1) the pruning formulation (3) has a similar effect on the structure of the model weights as the ℓ_1 penalized formulation, however, unlike the ℓ_1-version the number of non-zero weights is controlled precisely; 2) the resulting networks achieve better error-pruning tradeoff when compared to pure ℓ_0- and ℓ_1-pruned networks; and 3) the newly introduced hyperparameter λ does not need a heavy tuning, and choosing the value of $\lambda \in [10^{-5}\, 10^{-4}]$ is sufficient.

3. RELATED WORK

In recent years, many algorithmic and procedural solutions were proposed to solve the problem (1) and related pruning formulations. However, we limit our review to the pruning methods that are similar in spirit to formulation (2): the ones that allow to target a specific, exact number of remaining weights at the end of the pruning.

Saliency ranking methods The methods in this category assign an importance measure to weights of the pre-trained neural network, prune a certain fraction of less important weights and optionally retrain afterwards. These methods are used as robust baselines due to their simplicity and often show competitive results [8, 9]. Many saliency measures have been proposed over the years: methods based on the first- and second-order information of the loss function [10–12]; methods based on magnitudes of the weights [13, 14, 8], and others. All of these methods approximately solve, depending on the quality of the saliency measure, the problem (2) iteratively by pruning certain fraction at every step, until the κ-budget is reached.

Optimization methods The ℓ_0 constraint has been studied in the context of pruning mechanisms [3, 15], and in a general network compression context with the goal of fitting a model into specific hardware constraints [16, 17] where the underlying optimization relied on the methods of multipliers with alternating optimization. Alternatively, ℓ_0-based pruning can be achieved in a Bayesian setting using priors of special forms [18, 19].

On the usage of ℓ_2 penalty The ℓ_2 penalty has a strong regularizing effect, and can be seen as having a Gaussian prior on the weights in the Bayesian treatment. The additive combination of the ℓ_2 norm has been extensively studied to improve the sparsification properties of LASSO problems and generally known as elastic net models [20, ch. 4.2]. In this paper, we study the properties of a different combination of ℓ_0 + ℓ_2 with the application to the pruning of neural networks, which constitutes a new, and yet unexplored, avenue of the pruning research.

4. OPTIMIZATION OF THE PROPOSED FORMULATION

The formulation (3) we are studying is a challenging problem due to the non-differentiable constraint given by the ℓ_0 norm and the non-convexity of the task L. To make it amenable to standard optimization software and to get an insight into the structure of the pruning, we proceed by turning the problem into the learning-compression (LC) formulation [21]. We introduce a duplicate variable $\boldsymbol{\theta}$ with an equality constraint of $\mathbf{w} = \boldsymbol{\theta}$ and then apply alternating optimization. Depending on how we introduce the duplicates, we end up with two different versions of the optimization. As we will show next, one version of the algorithm reveals the shrinkage property of the $\ell_0 + \ell_2$ combination, however, the other version of the algorithm can already be implemented in the LC toolkit without any special modifications.

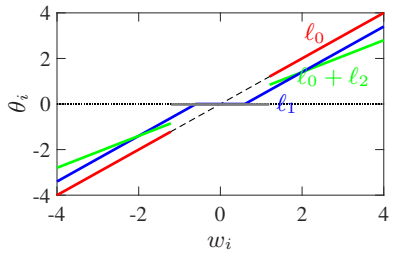
4.1. LC algorithm, version 1 (penalty in the C step)

When introducing the variable $\boldsymbol{\theta}$, let us leave $L(\mathbf{w})$ as is, and replace all other $\mathbf{w}$ occurrences with $\boldsymbol{\theta}$ and jointly optimize the following:

$$\min_{\mathbf{w},\boldsymbol{\theta}} \quad L(\mathbf{w}) + \lambda \|\boldsymbol{\theta}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \leq \kappa, \quad \mathbf{w} = \boldsymbol{\theta} \tag{4}$$

Now, to derive an efficient algorithm we apply the Quadratic Penalty [22, ch. 17] (or augmented Lagrangian, which requires minor modification) to the equality constraints and optimize an equivalent prob-

formulation	solution for θ_i
ℓ_0 constrained [3]	$\theta_i = w_i \cdot I(\lvert w_i\rvert \geq \eta_\kappa)$
ℓ_1 constrained [3]	$\theta_i = \left(w_i - \operatorname{sgn}(w_i)\,\frac{\lambda}{\mu}\right) \cdot I(\lvert w_i\rvert \geq \frac{\lambda}{\mu})$
ℓ_0 constrained. + ℓ_2 penalty	$\theta_i = \frac{\mu}{\mu+2\lambda} w_i \cdot I(\lvert w_i\rvert \geq \eta_\kappa)$



f

Fig. 2. C-step solutions of different pruning operators (top) and its graphical representation (bottom). Here, κ and λ are hyperparameters of pruning operators, μ is the penalty parameter of Quadratic Penalty(or augmented Lagrangian), I is the indicator function, and η_k is κ^{th} largest value in magnitude among all weights. The pruning operator consisting of only ℓ_0-norm does not impose any shrinkage on the weights, however combining the ℓ_0 with ℓ_1 or ℓ_2^2 penalty will make surviving weights smaller.

lem while driving $\mu \to \infty$:

$$\min_{\mathbf{w},\boldsymbol{\theta}} \quad L(\mathbf{w}) + \lambda \|\boldsymbol{\theta}\|_2^2 + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\theta}\|^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \leq \kappa$$

This reformulation is advantageous as it allows to apply alternating optimization over $\mathbf{w}$ and $\boldsymbol{\theta}$ where steps are given in the forms that can be efficiently solved:

- **L step** of $\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\theta}\|^2$.
  This step has the form of the neural network training, but with an additional ℓ_2 penalty. Such training can be handled by any deep learning framework, and does not require any special treatment. Following the LC paper [21] we call this step a learning (L) step.
- **C step** of $\min_{\boldsymbol{\theta}} \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}\|_2^2$ s.t. $\|\boldsymbol{\theta}\|_0 \leq \kappa$
  This step has the form of finding the optimal constrained $\boldsymbol{\theta}$ (with at most κ non zeros) minimizing a convex objective function, and can be solved exactly. This step was called a compression (C) step in [21].

Then, our optimization procedure is as follows: while slowly driving $\mu \to \infty$ we alternate two simple steps: the step of neural network training, and the step of optimal pruning (compression) of its weights ($\mathbf{w}$) into the duplicating $\boldsymbol{\theta}$. This duplicating variables should be considered as *pruned copy* of the weights: indeed, at the limit of $\mu \to \infty$ they both agree by satisfying $\mathbf{w} = \boldsymbol{\theta}$.

Solution of the C step To find optimum $\boldsymbol{\theta}^*$ of the C-step problem (4.1), let us rewrite the ℓ_0-norm using the index set Ω of size κ: we say $\theta_i = 0$ if and only if $i \notin \Omega$, and collectively refer to non-zero weights as $\boldsymbol{\theta}_\Omega$. Then, the C-step problem for $\mu > 0$ can be written as

$$\min_{\boldsymbol{\theta},\Omega} \quad \frac{\mu}{2}\sum_{i\in\Omega}\left((w_i - \theta_i)^2 + \frac{2\lambda}{\mu}\theta_i^2\right) + \frac{\mu}{2}\sum_{i\notin\Omega} w_i^2 \tag{5}$$

We note that for a fixed index set Ω, the minimization of (5) wrt $\boldsymbol{\theta}$ is a convex problem with the solution of

$$\boldsymbol{\theta}^*_\Omega = \frac{\mu}{\mu + 2\lambda}\mathbf{w}_\Omega, \tag{6}$$

and the loss of:

$$P(\boldsymbol{\theta}^*_\Omega) = \frac{\lambda\mu}{\mu+2\lambda}\sum_{i\in\Omega} w_i^2 + \frac{\mu}{2}\sum_{i\notin\Omega} w_i^2.$$

Therefore, to find the global optimum of eq. (5) it is sufficient to find the set Ω that gives a minimal value of $P(\boldsymbol{\theta}^*_\Omega)$:

$$\Omega^* = \arg\min_{\Omega} P(\boldsymbol{\theta}^*_\Omega).$$

Let us add and subtract the value of $\frac{\mu}{2}\sum_{i\in\Omega} w_i^2$ to $P(\boldsymbol{\theta}^*_\Omega)$ and make some simplifications:

$$\begin{aligned}
\Omega^* &= \arg\min_{\Omega} P(\boldsymbol{\theta}^*_\Omega) + \frac{\mu}{2}\sum_{i\in\Omega} w_i^2 - \frac{\mu}{2}\sum_{i\in\Omega} w_i^2 \\
&= \arg\min_{\Omega} \frac{\lambda\mu}{\mu+2\lambda}\sum_{i\in\Omega} w_i^2 + \underbrace{\frac{\mu}{2}\sum_{i\notin\Omega} w_i^2 + \frac{\mu}{2}\sum_{i\in\Omega} w_i^2}_{\frac{\mu}{2}\sum_i w_i^2} - \frac{\mu}{2}\sum_{i\in\Omega} w_i^2 \\
&= \arg\min_{\Omega} \sum_{i\in\Omega}\frac{-\mu^2}{2(\mu+2\lambda)} w_i^2 + \sum_i w_i^2 = \arg\max_{\Omega}\sum_{i\in\Omega} w_i^2.
\end{aligned}$$

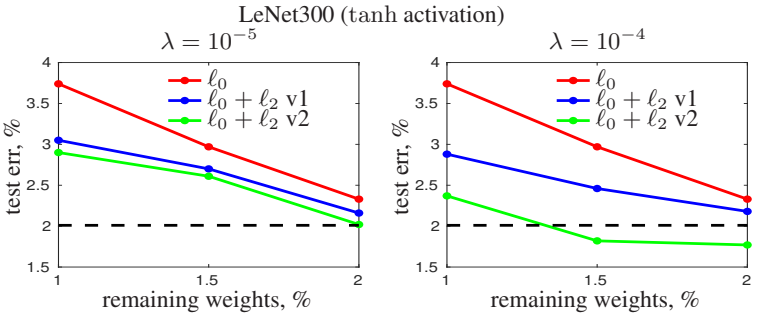
Here, the last step comes from the fact that $\sum_i w_i^2$ is constant wrt Ω and the term $\frac{-\mu^2}{2(\mu+2\lambda)}$ is always negative. Thus, finding Ω^* is equivalent to selecting κ-sized subset with the largest sum among items w_i^2. This, in turn, has the solution of selecting the top κ weights in their magnitude among $\mathbf{w} = [w_1, \ldots, w_N]$:

$$\Omega^* = \{i : w_i \in \text{Top-}\kappa \text{ largest in magnitude items of } \mathbf{w}\}. \tag{7}$$

The set Ω^* can be identified in $\mathcal{O}(N)$ time using medians-of-medians modification of QUICKSELECT algorithm [23, ch. 9.3]. Overall, the C-step solution is given by first computing the Ω^* using eq. (7) and then computing the $\boldsymbol{\theta}^*_\Omega$ using eq. (6).

Pruning properties The pruning properties of this C step is quite interesting: we prune all but top-κ weights of $\mathbf{w}$, and the remaining values *will get shrunk proportionally to their magnitude*. This is in contrast to the regular ℓ_0 formulation of the pruning where all unpruned weights remain as is; and rather reminiscent of the shrinking properties of the ℓ_1 pruning (see illustration in Fig. 2). Additionally, our ℓ_0 + ℓ_2 formulation has the advantages of both methods: we can specify the amount of pruning precisely (unlike ℓ_1 formulation), while experiencing a shrinkage effect (unlike ℓ_0 formulation).

Behavior of the algorithm Overall, the algorithm operates in the following way. We slowly drive the value of $\mu \to \infty$ and alternate two simple steps: the L step of neural network training, and the C step of optimal pruning of the weights $\mathbf{w}$ into the duplicating $\boldsymbol{\theta}$. The L step drives $\mathbf{w}$ closer to $\boldsymbol{\theta}$, however, the actual pruning happens at the limit of $\mu \to \infty$. Such pruning is advantageous as weights are not irrevocably destroyed: the C-step solution (i.e., the set of pruned weights with their values) will change over the iterations to select a better subset that has a lower loss. To run this version of the algorithm we simply add our C-step implementation to the LC toolkit (which is supported by design).



Selected pruning results on LeNet300-ReLU

	remaining w.	test-err
reference network	100%	1.87%
variational dropout [18]	1.47%	1.92%
DNS [24]	1.78%	1.99%
DeepHoyer [25]	2.27%	1.60%
our $\ell_0 + \ell_2$, $\lambda = 10^{-4}$	2.00%	**1.60%**
our $\ell_0 + \ell_2$, $\lambda = 10^{-4}$	1.50%	1.99%

Fig. 3. Left: Results of pruning LeNet300-tanh using the ℓ_0 pruning of [3], and our modified scheme $\ell_0 + \ell_2$ with both versions of the algorithm and different amount (λ) of ℓ_2-penalty. Our modified scheme $\ell_0 + \ell_1$ improves over regular ℓ_0 pruning. Right: our pruned LeNet300-ReLU models in comparison to selected results from the literature.

4.2. LC Algorithm, version 2 (penalty in the L step)

We can achieve a different version of the optimization algorithm, if we do not substitute the duplicating variable $\boldsymbol{\theta}$ into ℓ_2-penalty of (3), and leave it as is. Then, instead of reformulation (4) we get an equivalent formulation of

$$\min_{\mathbf{w},\boldsymbol{\theta}} \quad L(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \leq \kappa, \quad \mathbf{w} = \boldsymbol{\theta}. \qquad (8)$$

When we follow the similar steps as in sec. 4.1, the corresponding L and C steps for this reformulation are as follows:

- **L step** of $\min_{\mathbf{w}} L(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\theta}\|^2$
- **C step** of $\min_{\boldsymbol{\theta}} \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\theta}\|^2$ s.t. $\|\boldsymbol{\theta}\|_0 \leq \kappa$

These L and C steps can be considered as a particular case of the pruning formulation of the LC algorithm studied in [3] and, in fact, can be implemented in the LC toolkit without any additional extensions to the library. Indeed, can use a loss of $L'(\mathbf{w}) = L(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2$ instead of $L(\mathbf{w})$: this means training the neural network with a ℓ_2 weight decay by itself. *Both of the LC algorithms (versions 1 and 2) are equivalent: they solve the same $\ell_0 + \ell_2$ combination with the same resulting solution (stationary point) under mild assumptions. However, only the first version of the algorithm reveals the shrinking properties of the proposed pruning scheme, whereas the L and C steps of the this (second) algorithm are non-informative.*

5. EVALUATION

We run a set of highly controlled experiments to prune ReLU activated LeNet300, tanh activated LeNet300, and Caffe version of LeNet5 on the MNIST dataset. We train regular ℓ_0- and ℓ_1-constrained models with various sparsities using the LC toolkit, and compare it to $\ell_0 + \ell_2$ pruned networks obtained using both versions 1 and 2 of the LC algorithm. Since all evaluated algorithms are within the family of learning-compression framework [21], we use the same hyper-parameters across the experiments. We initialize our models from pre-trained neural networks achieving test errors of 2.01% for LeNet300-tanh, 1.87% for LeNet300-ReLU, and 0.82% for LeNet5. The LC algorithm is run for 30 iterations (we use augmented Lagrangian version) with $\mu_t = \mu_{\text{init}} \times b^t$ at iteration t. Each L step is trained with SGD (batch size 256) for 25 epochs with a learning rate of 0.1×0.95^t for LeNet300 and with a learning rate of 0.05×0.95^t for LeNet5 . After obtaining a pruned network we perform finetuning for 25–50 epochs with a learning rate of 0.01.

On the left of Fig. 3 we report the results of pruning LeNet300-tanh network using our $\ell_0 + \ell_2$ formulation. We prune networks to a budget of $\kappa = 1, 1.5, 2\%$ of remaining weights using several λ values, and compare to pure ℓ_0 pruning. Small amounts of ℓ_2 penalty improve the test errors of final networks across all sparsity values when compared to the ℓ_0-pruned networks. We also ask a question of which version of the algorithm (1 or 2) should be used? While we expect both algorithms to achieve same error for a given sparsity, we observe that the networks compressed with version 2 of the algorithm have slightly better error-sparsity tradeoff. For all subsequent experiments we use the second version of the algorithm.

On the right of Fig. 3, we report pruning results of LeNet300-ReLU with κ of 1.5% and 2%, and the results in the literature with similar sparsities. Our pruned models achieve comparable or better error-sparsity tradeoff, yet have an advantage: we can specify the number of remaining weights directly via κ, while other methods require trial-and-error process to find the right settings.

For LeNet5 our ℓ_0+ℓ_2 scheme with $\lambda = 10^{-5}$ achieves a pruned network with $\kappa = 1\%$ of non-zero weights and test error of 0.89%, while pure ℓ_0-pruned network of Carreira-Perpiñán and Idelbayev has an error of 1.06%. Our results are comparable with results of Yang et al. [17] ($\kappa = 0.9\%$ with test error of 0.8%), Guo et al. [24] ($\kappa = 0.93\%$ with test error of 0.91%) and Zhang et al. [15] ($\kappa = 1.4\%$ with test error of 0.8%).

We give the visualization of the pruned architectures and the neurons of the first layer of LeNet300-tanh on Fig. 1. As expected by our theoretical analysis, the $\ell_0 + \ell_2$ scheme exhibits a behavior similar to the ℓ_1-pruned network and has a fewer number of alive neurons when compared to pure ℓ_0 formulation.

6. CONCLUSION

We have studied the behavior of ℓ_0-constrained pruning of the neural networks, which is a non-differentiable and non-convex problem. We observed that by itself, ℓ_0-pruned network results in an architecture with many alive neurons having only a few non-zero weights. To remedy such a behavior we proposed a modification of the ℓ_0 pruning involving a small amount of ℓ_2 penalty and studied its properties using the common algorithmic framework of the LC toolkit. We have discovered that the proposed formulation has a strong shrinkage effect on the weights, which is similar to the effect of the ℓ_1 formulation of the pruning. Experimentally, we validated that our proposed scheme achieves a better error for a given sparsity level when compared to the regular ℓ_0-pruned networks. This suggests that adding the ℓ_2 regularization (a regular weight decay) in small amounts is beneficial to the large class of the pruning problems involving the ℓ_0 constraints and various magnitude-based criteria; and, perhaps, worth considering as a rule-of-thumb addition for such algorithms.

7. REFERENCES

[1] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "LC: A flexible, extensible open-source toolkit for model compression," in *30th ACM Int. Conf. Information & Knowledge Management (CIKM 2021)*, Online, Nov. 1–5 2021, pp. 4504–4514.

[2] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev, "Model compression as constrained optimization, with application to neural nets. Part II: Quantization," arXiv:1707.04319, 2017.

[3] Miguel Á. Carreira-Perpiñán and Yerlan Idelbayev, ""Learning-compression" algorithms for neural net pruning," in *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, Salt Lake City, UT, June 18–22 2018, pp. 8532–8541.

[4] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "Low-rank compression of neural nets: Learning the rank of each layer," in *Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, Seattle, WA, June 14–19 2020, pp. 8046–8056.

[5] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "More general and effective model compression via an additive combination of compressions," in *Proc. of the 32nd European Conf. Machine Learning (ECML–21)*, Bilbao, Spain, Sept. 13–17 2021, pp. 233–248.

[6] Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán, "Beyond FLOPs in low-rank compression of neural networks: Optimizing device-specific inference runtime," in *IEEE Int. Conf. Image Processing (ICIP 2021)*, Online, Sept. 19–22 2021, pp. 2843–2847.

[7] Ronny Krashinsky, Olivier Giroux, Stephen Jones, Nick Stam, and Sridhar Ramaswamy, "NVIDIA Ampere architecture in-depth," https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/, Accessed: 10/15/2020.

[8] Song Han, Huizi Mao, and William J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.

[9] Jonathan Frankle and Michael Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *Proc. of the 7th Int. Conf. Learning Representations (ICLR 2019)*, New Orleans, LA, May 6–9 2019.

[10] Yann LeCun, John S. Denker, and Sara A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems (NIPS)*, 1990, vol. 2, pp. 598–605, Morgan Kaufmann, San Mateo, CA.

[11] Michael C. Mozer and Paul Smolensky, "Skeletonization: A technique for trimming the fat from a network via relevance assessment," in *Advances in Neural Information Processing Systems (NIPS)*, 1989, vol. 1, pp. 107–115, Morgan Kaufmann, San Mateo, CA.

[12] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz, "Pruning convolutional neural networks for resource efficient inference," in *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.

[13] Dong Yu, Frank Seide, Gang Li, and Li Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'12)*, Kyoto, Japan, Mar. 25–30 2012, pp. 4409–4412.

[14] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems (NIPS)*, 2015, vol. 28, pp. 1135–1143, MIT Press, Cambridge, MA.

[15] Tianyun Zhang, Shaokai Ye, Kaiqi Zhang, Jian Tang, Wujie Wen, Makan Fardad, and Yanzhi Wang, "A systematic DNN weight pruning framework using alternating direction method of multipliers," in *Proc. 15th European Conf. Computer Vision (ECCV'18)*, Munich, Germany, Sept. 8–14 2018, pp. 191–207.

[16] Yuzhe Ma, Ran Chen, Wei Li, Fanhua Shang, Wenjian Yu, Minsik Cho, and Bei Yu, "A unified approximation framework for deep neural networks," arXiv:1807.10119, July 26 2018.

[17] Haichuan Yang, Shupeng Gui, Yuhao Zhu, and Ji Liu, "Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach," in *Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, Seattle, WA, June 14–19 2020, pp. 2175–2185.

[18] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, Sydney, Australia, Aug. 6–11 2017, pp. 2498–2507.

[19] Christos Louizos, Max Welling, and Diederik P. Kingma, "Learning sparse neural networks through l_0 regularization," in *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*, Vancouver, Canada, Apr. 30 – May 3 2018.

[20] Trevor Hastie, Robert Tibshirani, and Martin Wainwright, *Statistical Learning with Sparsity: The Lasso and Generalizations*, Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 2015.

[21] Miguel Á. Carreira-Perpiñán, "Model compression as constrained optimization, with application to neural nets. Part I: General framework," arXiv:1707.01209, July 5 2017.

[22] Jorge Nocedal and Stephen J. Wright, *Numerical Optimization*, Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

[23] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, third edition, 2009.

[24] Yiwen Guo, Anbang Yao, and Yurong Chen, "Dynamic network surgery for efficient DNNs," in *Advances in Neural Information Processing Systems (NIPS)*, 2016, vol. 29, pp. 1379–1387, MIT Press, Cambridge, MA.

[25] Huanrui Yang, Wei Wen, and Hai Li, "DeepHoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures," in *Proc. of the 8th Int. Conf. Learning Representations (ICLR 2020)*, Addis Ababa, Ethiopia, Apr. 26–30 2020.