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

Yerlan Idelbayev and **Miguel Á. Carreira-Perpiñán**, CSE, UC Merced

icassp 2022 Singapore

[MLSP-10.3]

1 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. 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.

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

2 Introduction: Setup of the pruning problem

Given a network with weights $\mathbf{w}$ and task loss L, most of the NN pruning problems are formulated using a sparsifying penalty $P(\mathbf{w})$ and solve a regularized or constrained formulation given as

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

In particular, we are interested in ℓ_0 formulation of the pruning where

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

where we can precisely set the number of non-zero (unpruned) weights via parameter κ: the total budget of the remaining parameters.

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.

3 Modifed formulation

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: 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;

To make it amenable to standard optimization software we proceed by turning the problem into the learning-compression (LC) formulation [1]. 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. However, for the purposes of this poster, we only discuss

4 Optimization. LC algorithm, version 1

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 [2, ch. 17] and optimize an equivalent problem 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 [1] 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 [1].

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 $\mathbf{w} = \boldsymbol{\theta}$.

5 Solution of the C step

We 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

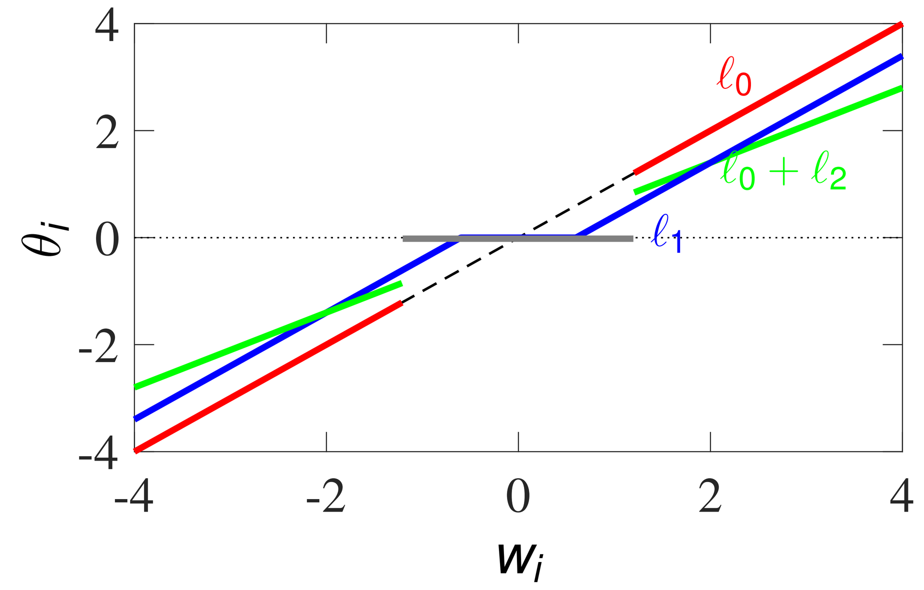
$$\boldsymbol{\theta}^*_\Omega = \frac{\mu}{\mu + 2\lambda}\mathbf{w}_\Omega, \tag{6}$$

In turn, in the paper, we show that the optimal Ω^* can be found by:

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

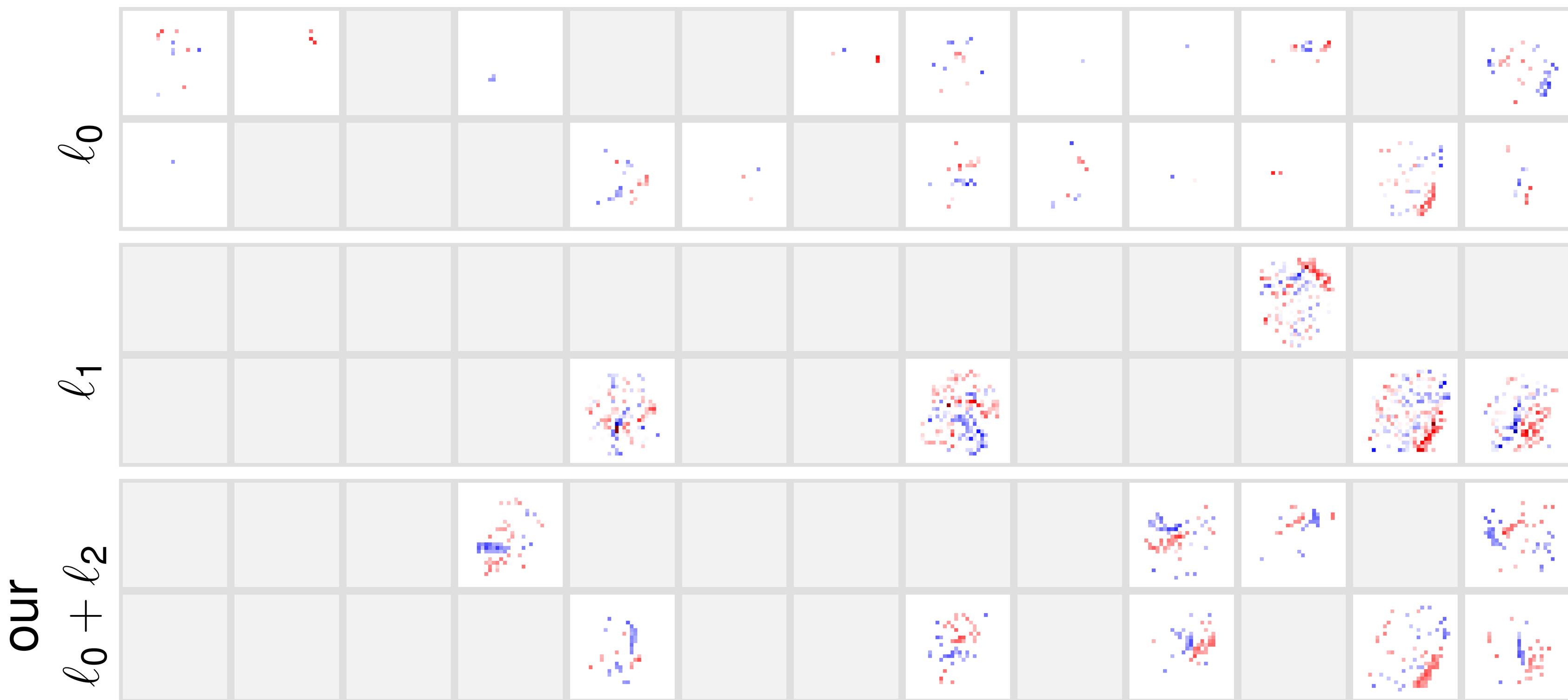
6 Pruning properties

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). Additionally, our $\ell_0 + \ell_2$ 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).



7 Experiments

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



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.

8 References

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

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