“LEARNING-COMPRESSION” ALGORITHMS FOR NEURAL NET PRUNING

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

2018 CVPR SALT LAKE CITY • JUNE 18-22

1 Abstract

Pruning a neural net is a process of removing weights with the goal of minimally degrading its performance. Pruning has been traditionally done by ranking or penalizing weights according to some criterion (e.g. magnitude), removing low-ranked weights and retraining the remaining ones. We formulate pruning as an optimization problem of finding the weights that minimize the loss while satisfying a pruning cost condition. We give a generic algorithm to solve this, which alternates “learning” steps that optimize a regularized, data-dependent loss and “compression” steps that mark weights for pruning in a data-independent way.

Work supported by NSF award IIS–1423515, by a UC Merced Faculty Research Grant, by computing time in the MERCED cluster (NSF grant ACI–1429783), and by a Titan X Pascal GPU donated by the NVIDIA Corporation.

2 Motivation

- Deep learning is very successful in some practical applications: vision, speech, NLP
- Achieves very good classification accuracy if training large, deep neural nets on large datasets (with several GPUs over several days. . .).
- It is of interest to deploy high-accuracy deep nets on limited-computation devices (mobile phones, IoT, etc.), but this requires compressing the deep net to meet the device’s limitations in memory, runtime, energy, bandwidth, etc.
- Surprisingly high compression rates often possible because deep nets are vastly over-parameterized in practice. It seems this makes it easier to learn a high-accuracy net.
- One way of compression is pruning, which is a process of removing weights
- Pruning will degrade accuracy so we want to prune optimally.

3 Pruning as an optimization problem

Consider a following formulation for learning an optimally pruned network:

task loss — weights of neural net — user defined param. # weights to be left

$$\min_{\mathbf{w}} L(\mathbf{w}) \quad \text{s.t.} \quad \underbrace{\|\mathbf{w}\|_0 \le \kappa}_{\text{pruning constraints}} \qquad \text{pruning forms: } \begin{cases} \ell_0 - \text{constraint} \\ \ell_0 - \text{penalty} \\ \ell_1 - \text{constraint} \\ \ell_1 - \text{penalty} \end{cases}$$

Equivalently, using $\boldsymbol{\theta}$ as an auxiliary variable to duplicate weights $\mathbf{w}$:

$$\min_{\mathbf{w},\boldsymbol{\theta}} L(\mathbf{w}) \quad \text{s.t.} \quad \mathbf{w} = \boldsymbol{\theta}, \ \ \|\boldsymbol{\theta}\|_0 \le \kappa$$

This puts problem in “learning-compression” form, which admits an efficient solution using penalty methods [Ref: “Model compression as constrained optimization, with application to neural nets”, Part I: general framework, arxiv 2017 (1707.01209)]

4 The “learning-compression” (LC) algorithm

Use a penalty method (e.g. Quadratic Penalty/Augmented Lagrangian) on equality constraints: as $\mu \to \infty$:

$$\min_{\mathbf{w},\boldsymbol{\theta}} Q(\mathbf{w},\boldsymbol{\theta};\mu) = L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}-\boldsymbol{\theta}\|^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \le \kappa$$

using alternating optimization over $\mathbf{w}$ and $\boldsymbol{\theta}$:

L-step: $\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}-\boldsymbol{\theta}\|^2$ – independent of pruning

C-step: $\min_{\boldsymbol{\theta}} \|\mathbf{w}-\boldsymbol{\theta}\|^2$ s.t. $\|\boldsymbol{\theta}\|_0 \le \kappa$ – independent of the loss and dataset

L-step always takes the same form regardless of the pruning criterion:

$$\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w}-\boldsymbol{\theta}\|^2$$

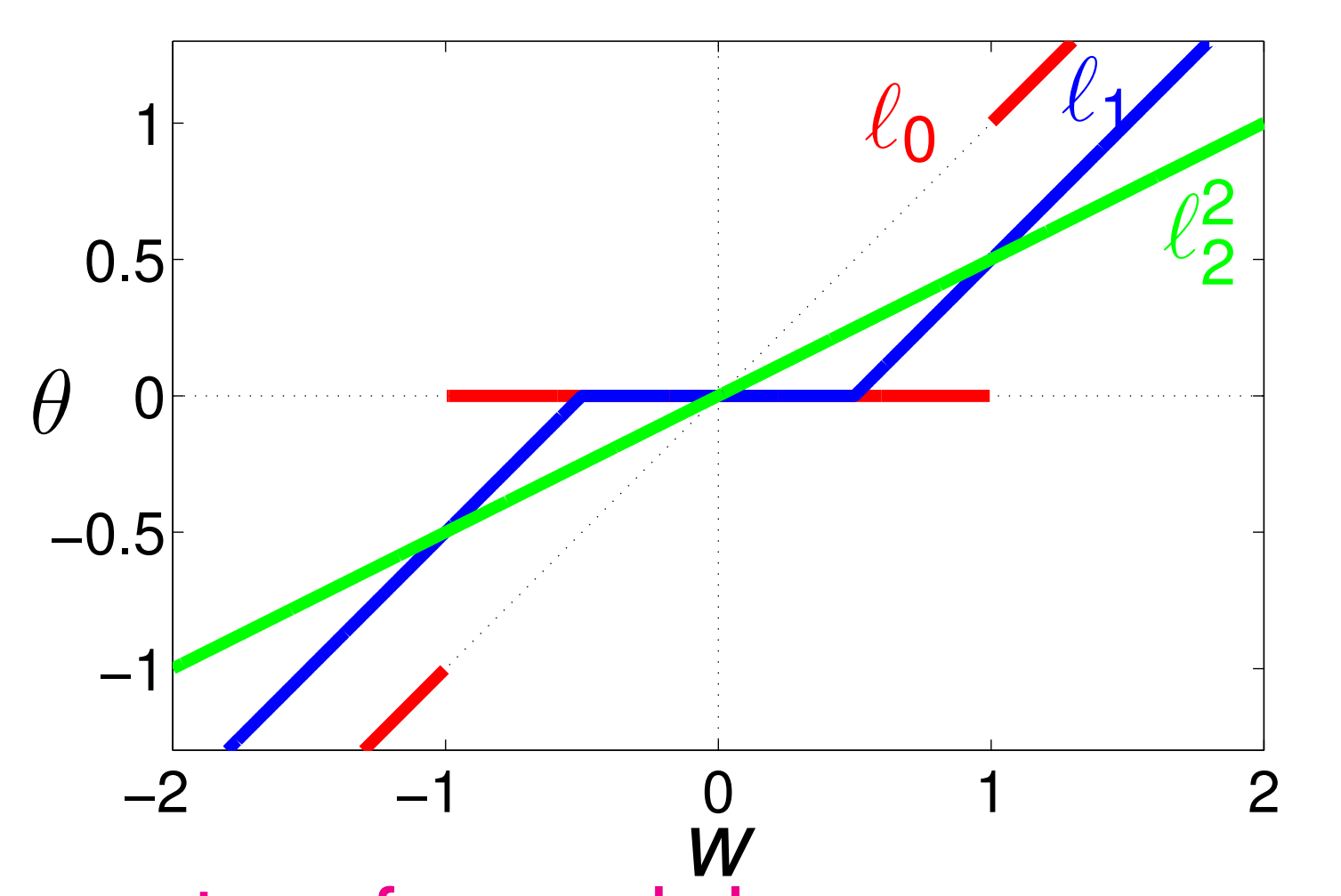
- This is the original loss on the weights $\mathbf{w}$, regularized with a quadratic term on the weights (which “decay” towards a pruned copy $\boldsymbol{\theta}$).
- It can be optimized in the same way as the reference net. Add $\mu(\mathbf{w}-\boldsymbol{\theta})$ to the gradient.
- With large datasets we typically use SGD

clip the learning rates so they never exceed $\frac{1}{\mu}$ to avoid oscillations as $\mu \to \infty$.

C-step solution means optimally pruning the current weights $\mathbf{w}$:

$$\min_{\boldsymbol{\theta}} \|\mathbf{w}-\boldsymbol{\theta}\|^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \le \kappa$$

- Solution: for ℓ_0-constraint case – leave as is the top-κ weights in magnitude, zero the rest. In general, some kind of thresholding (see figure on the right).
- Fast: it does not require the dataset, which only appears in $L(\mathbf{w})$.
- With a deep net, automatically finds the optimal number of weights to prune in each layer. No need to set pruning hyper-parameters for each layer. A single hyper parameter κ defines optimal pruning amount for each layer



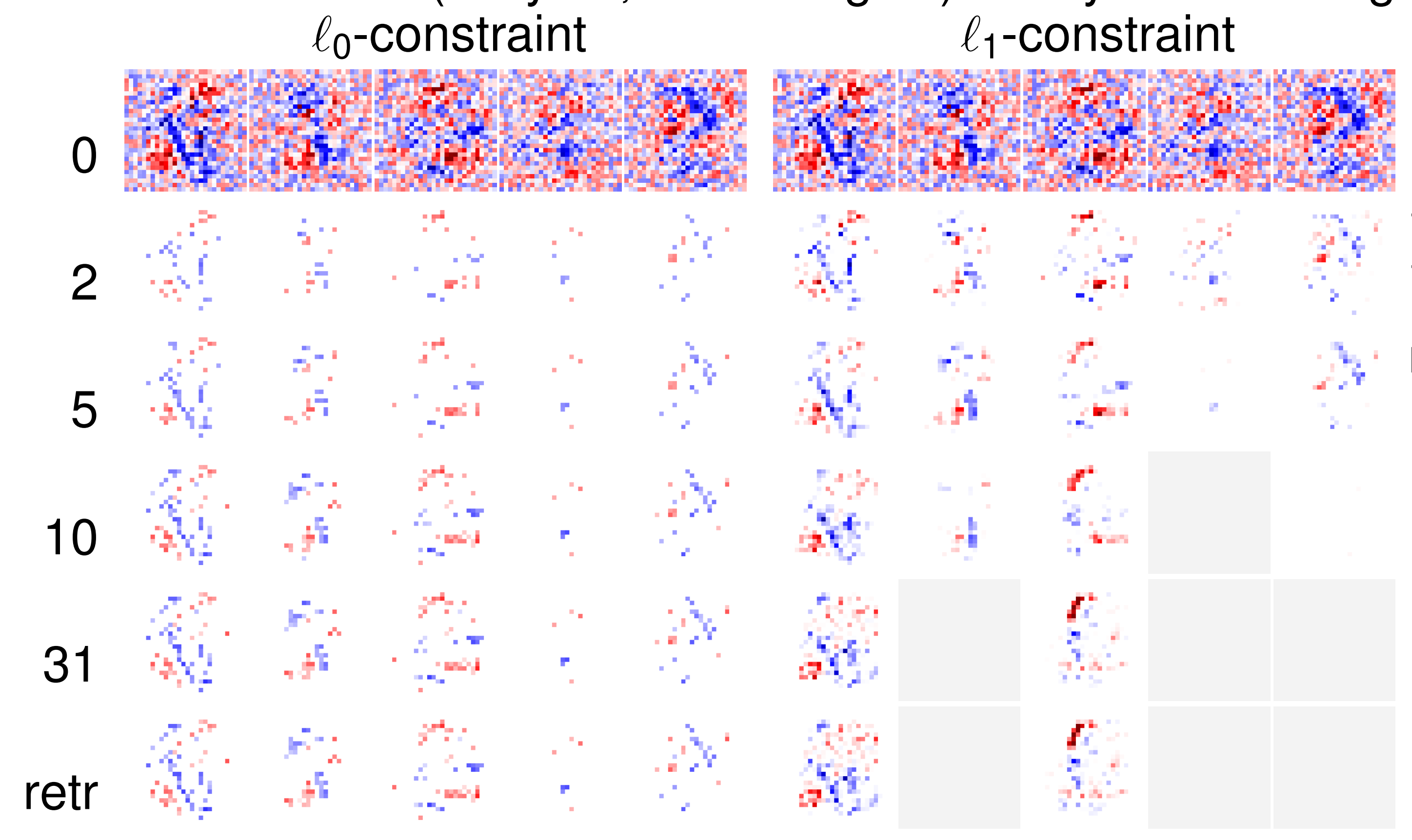
This is like magnitude pruning, but notice that weights are not irrevocably removed. They are “marked” as currently pruned in C-step via $\theta_i = 0$, while w_i values stay as nonzero. As the LC algorithm alternates both steps, it explores different sets of marked weights, eventually converging to a specific set and driving $\mathbf{w}$ towards $\boldsymbol{\theta}$, thus pruning.

Magnitude pruning is a simple approach of removing all but top-κ weights and retraining remaining.

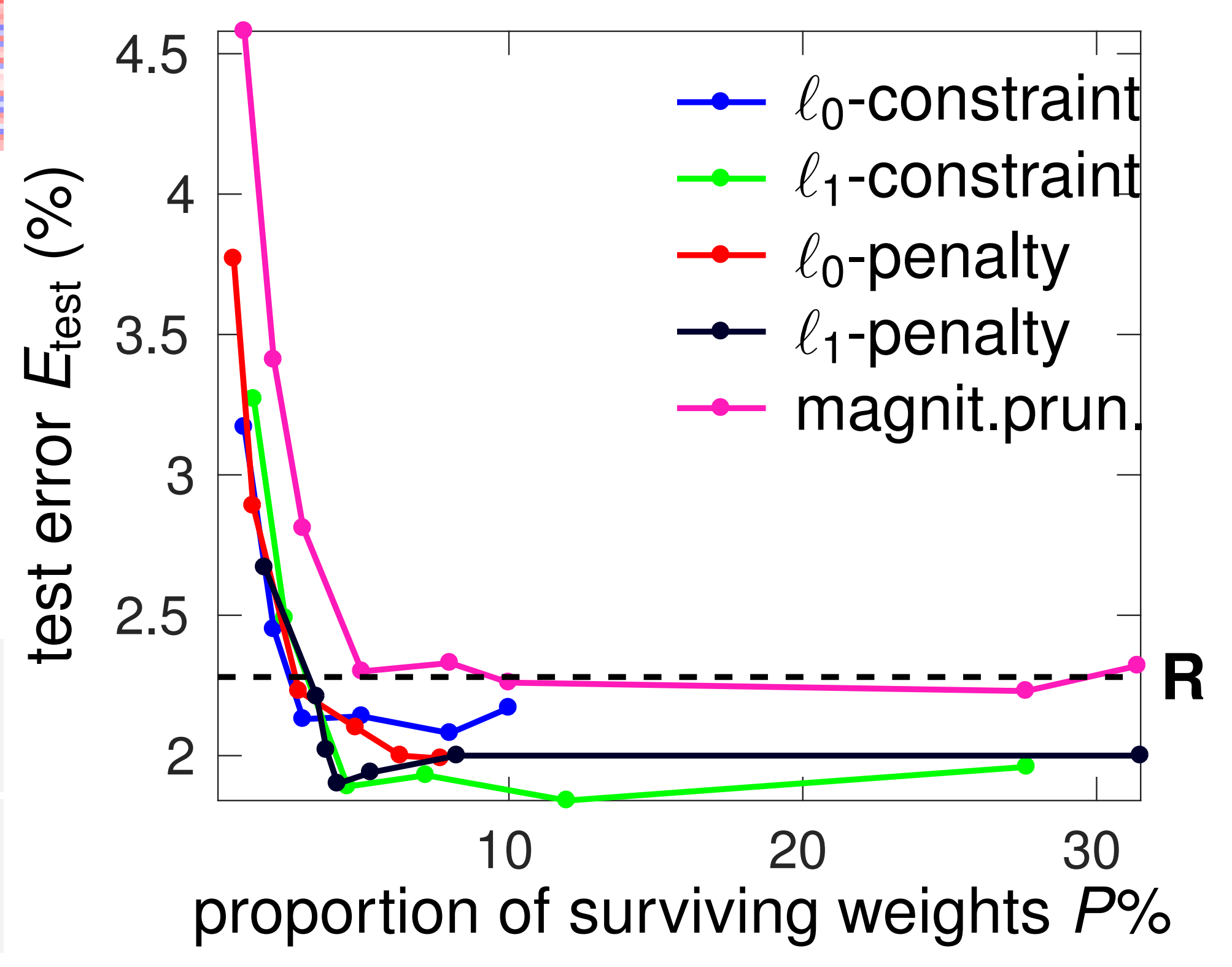
5 Experiments

MNIST/LeNet-s

For LeNet300 (3 layers, 266k weights) nearly no error degradation using P=2% weights.



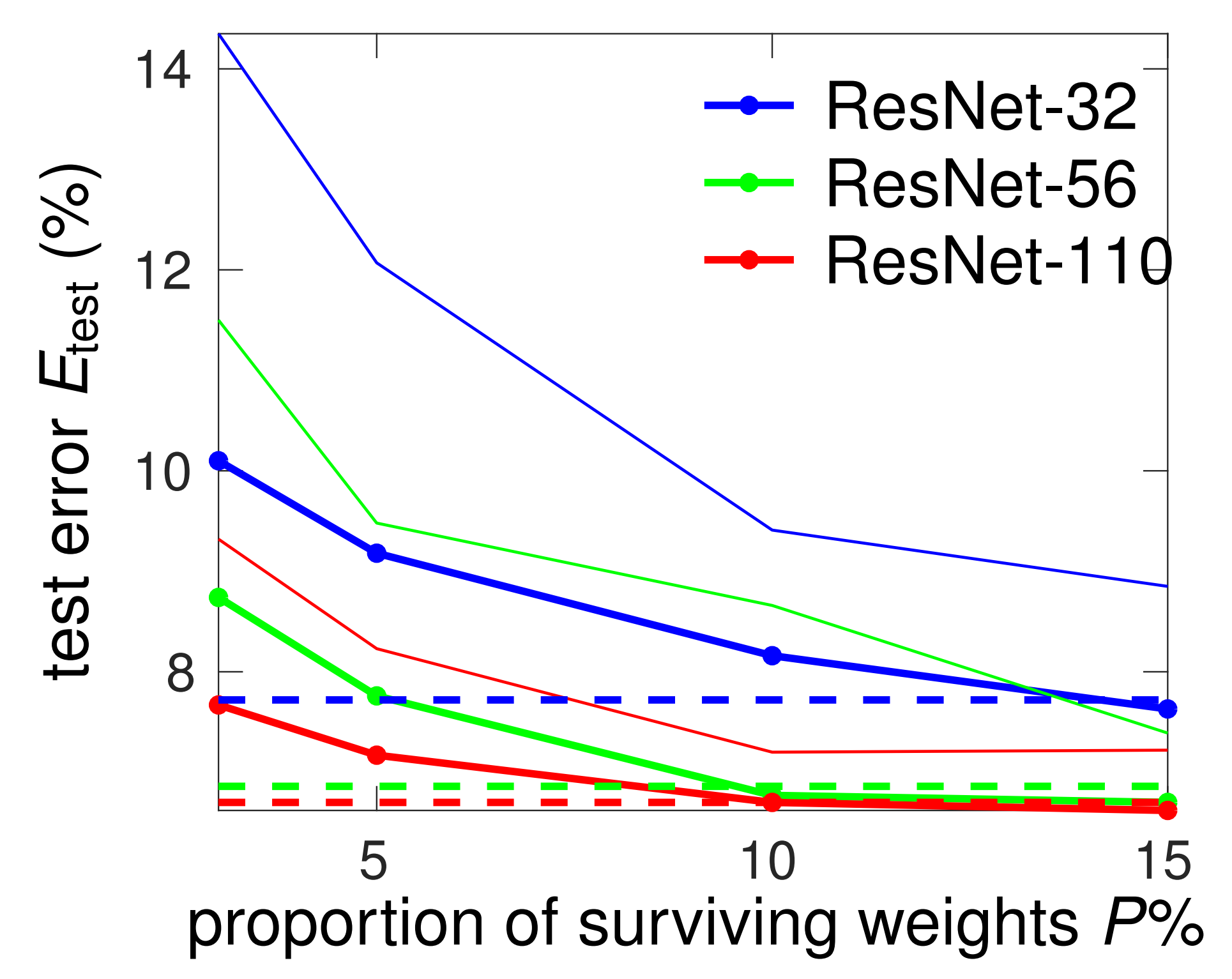
Selected 1st layer neurons for LeNet300 during LC



- The final weight vectors segment the image into negative and positive regions reminiscent of center-surround receptive fields, but these regions a sparse rather compact
- The algorithm prunes weights, not neurons, but we observe aggressive neuron pruning. The original LeNet300 architecture 784–300–100–10 becomes 400–64–99–10 with similar classification error
- The algorithm might be useful to do feature selection and determine optimal number of neurons in each layer. E.g. for input layer active neurons look like this:

CIFAR10/ResNet-s

Thick lines – LC, thin lines – magnitude based pruning, dashed horizontal lines – reference models.



Results of pruning ResNets of different depth using LC, trained on CIFAR10 (60K images, 10 classes) and having 0.4M, 0.8M and 1.7M parameters. LC achieves better errors in high pruning regions, $P < 5\%$ and always better than pruning-retraining strategy. The

ResNet models are very lean, achieving state-of-the-art results in classification tasks with a much smaller number of weights, therefore, pruning them is more harder task.