An Empirical Comparison of Quantization, Pruning and Low-rank Neural Network Compression using the LC Toolkit

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán
Dept. CSE, University of California, Merced
http://eecs.ucmerced.edu

The code is available at:
https://github.com/UCMerced-ML/LC-model-compression
Introduction: A need for NN compressions benchmark

Compression of neural networks become an important practical problem with plethora of works and approaches in recent years.
Which of these compressions is the best for a given model?

A simple solution: try different (possibly, off-the-shelf) compression schemes and algorithms and choose the best.

Unfortunately, it is often impossible due to:

- Fairness and objectivity of comparison: Compression methods have (very) different algorithmic base
- Availability of the code: Limited availability of readily usable code
- The choice of hyperparameters: Often, important hyperparameters are undisclosed or hard to select for a new task
We want to compare pruning, low-rank, and quantization on multiple networks. To make the systematic comparison possible, we need a practical tool that is free of the aforementioned challenges.

We use recently proposed learning-compression (LC) algorithm, to lay a groundwork of such comparison work. The framework:

- seamlessly integrates number of compressions under the same roof
- is based on solid optimization principles and algorithms
- allows greater code reuse as model training is separated from compression
- combined, it allows an apples-to-apples comparison between techniques
Outline

This talk is structured in the following way:

- Overview of the LC algorithm
- Overview of the LC software
- Details of experimental setup & comparison
- Conclusion
Model compression as constrained optimization (MCCO)

General formulation:

$$\min_{w, \Theta} L(w) + \lambda C(\Theta) \quad \text{s.t.} \quad w = \Delta(\Theta)$$

We treat compression and decompression as mathematical mappings in parameter space.

The details of the compression technique are abstracted in $\Delta(\Theta)$. 

**Diagram:**
- uncompressed weights
- low-dim. params
- task loss
- compression cost
- decompression mapping $\Delta: \Theta \rightarrow w \in \mathbb{R}^P$
- feasible models $C$ (decompressible by $\Delta$)
- $w^*$ (optimal compressed)
- $\Delta(\Theta^{DC})$ (direct compression)
Optimization of MCCO, the LC algorithm

Reformulate using penalty method and optimize the following with $\mu \to \infty$:

$$\min_{\mathbf{w}, \Theta} L(\mathbf{w}) + \lambda C(\Theta) + \frac{\mu}{2} \| \mathbf{w} - \Delta(\Theta) \|^2$$

Alternation between $\mathbf{w}$ and $\Theta$ gives the learning-compression (LC) algorithm:

- **Learning (L) step:**
  $$\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2} \| \mathbf{w} - \Delta(\Theta) \|^2$$
  - This is a regular training of the model, but with a quadratic regularization term.
  - L step is independent of compression mechanism.
  - We will use SGD and standard NN software

- **Compression (C) step:**
  $$\min_{\Theta} \lambda C(\Theta) + \frac{\mu}{2} \| \mathbf{w} - \Delta(\Theta) \|^2$$
  - For $\lambda = 0$, the C step becomes an optimal projection problem
  - The C step is independent of the dataset
  - Many well studied cases with fast solutions
LC algorithm: Pseudocode

**Input** neural net with weights $w$, hyperparameter $\lambda$, cost function $C$

$w \leftarrow \text{arg min}_w L(w)$

$\Theta \leftarrow 0$

**For** $\mu = \mu_1 < \mu_2 < \cdots < \mu_T$

$w \leftarrow \text{arg min}_w L(w) + \frac{\mu}{2} \| w - \Delta(\Theta) \|_F^2$

$\Theta \leftarrow \text{arg min}_\Theta \lambda C(\Theta) + \frac{\mu}{2} \| w - \Delta(\Theta) \|$

**If** $\| w - \Delta(\Theta) \| \approx 0$

set $w \leftarrow \Delta(\Theta)$ and exit

**Return** $w, \Theta$

reference net

compressed weights

$L$ step

$C$ step

early stopping
<table>
<thead>
<tr>
<th>Type</th>
<th>Forms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quantization</strong></td>
<td>Adaptive Quantization into ${c_1, c_2, \ldots, c_K}$</td>
</tr>
<tr>
<td></td>
<td>Binarization into ${-1, 1}$ and ${-c, c}$</td>
</tr>
<tr>
<td></td>
<td>Ternarization into ${-c, 0, c}$</td>
</tr>
<tr>
<td><strong>Pruning</strong></td>
<td>$\ell_0$-constraint (s.t., $|w|_0 \leq \kappa$)</td>
</tr>
<tr>
<td></td>
<td>$\ell_1$-constraint (s.t., $|w|_0 \leq \kappa$)</td>
</tr>
<tr>
<td></td>
<td>$\ell_0$-penalty ($\alpha |w|_0$)</td>
</tr>
<tr>
<td></td>
<td>$\ell_1$-penalty ($\alpha |w|_1$)</td>
</tr>
<tr>
<td><strong>Low-rank</strong></td>
<td>Low-rank compression to a given rank</td>
</tr>
<tr>
<td></td>
<td>Low-rank with <em>automatic</em> rank selection for FLOPs reduction</td>
</tr>
<tr>
<td></td>
<td>Low-rank with <em>automatic</em> rank selection for storage compression</td>
</tr>
<tr>
<td><strong>Additive Combinations</strong></td>
<td>Quantization + Pruning</td>
</tr>
<tr>
<td></td>
<td>Quantization + Low-rank</td>
</tr>
<tr>
<td></td>
<td>Pruning + Low-rank</td>
</tr>
<tr>
<td></td>
<td>Quantization + Pruning + Low-rank</td>
</tr>
</tbody>
</table>
Having an L-step implementation (you only need one), definition of compression is very simple:

quantize each layer with separate codebooks

```python
compression_tasks = {
    Param(l1.weight): (AsVector, AdaptiveQuantization(k=2)),
    Param(l2.weight): (AsVector, AdaptiveQuantization(k=2)),
    Param(l3.weight): (AsVector, AdaptiveQuantization(k=2))
}
```

prune all but 5%

```python
compression_tasks = {
    Param([l1.weight, l2.weight, l3.weights]):
        (AsVector, ConstraintLOPruning(kappa=13310)) # 13310 = 5%
}
```

prune first layer, low-rank to second, quantize third

```python
compression_tasks = {
    Param(l1.weight): (AsVector, ConstraintLOPruning(kappa=5000)),
    Param(l2.weight): (AsIs, LowRank(target_rank=10))
    Param(l3.weight): (AsVector, AdaptiveQuantization(k=2))
}
```
LC software: Source code and other features

Our code is written in Python using PyTorch, and open source under BSD 3-clause license:

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

Using the provided code, you will be able to:

- replicate all reported experiments
- compress your own models with many available compression schemes

Our library is:

- modular and easily extensible
- only requires the L-step implementation: the regular learning of the model (using SGD)
- based on solid optimization principles
- single algorithm—many compressions
- time proven (development since 2017), with many publications [1–9]
Experiments: Setup

For our comparison we use the following forms of compressions:

- **Quantization**: adaptive quantization with a separate codebook of size $K$ per each layer
- **Pruning**: constrained $\ell_0$ based pruning, where we specify number of remaining nonzeros $\kappa$
- **Low-rank compression**: automatic rank-selection with cost function $C'$ targeting the number of parameters

Exact details of these compressions and the solution of corresponding C-step problems are in the main paper.
Experiments: Setup (cont.)

- For a given network, the L step of every compression experiment has same hyperparameters (learning rates, number of batches, etc) which allows us to make apples-to-apples comparisons across compressions.

- We vary the compression parameters (codebook size $K$, number of non-zeros $\kappa$, amount of penalty $\lambda$) to generate entire compression-error tradeoff curves.

- We report the compression ratio computed as:

  \[
  \text{compression ratio} = \frac{\text{uncompressed size}}{\text{compressed size}}
  \]

  where compressed size is computed as the actual amount of storage required to save the model to the disk.

  Exact details are in the paper.
Experiments: MNIST

Tradeoff on LeNet300

Tradeoff on LeNet5
Experiments: CIFAR10

Tradeoff on ResNet32

Tradeoff on VGG16
We argued and experimentally validated the necessity of fair and objective empirical comparison of different compression mechanisms.

The tool that allowed us to perform this comparison is the LC algorithm and its implementation — the LC Toolkit.

We have empirically observed that there is no single compression that universally outperforms other schemes.

However, for some networks pruning is a safe starting choice.
References


