LC: A Flexible, Extensible Open-Source Toolkit for Model Compression

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

The code is available at:
https://github.com/UCMerced-ML/LC-model-compression

Machine learning and neural networks

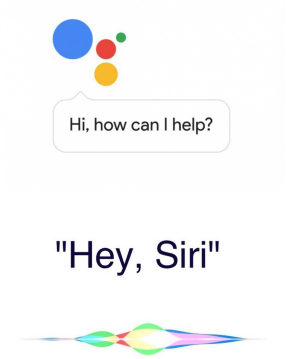
Neural networks have established state-of-the-art performance nearly in every machine learning task:

- Natural Language Processing (NLP): dialogs systems, translators...
- Computer vision: image and video recognition, classification...
- Speech processing: speech-to-text, audio synthesis...
- Various signal enhancement tasks (photo, audio, video)

The models trained on these tasks have significant practical importance

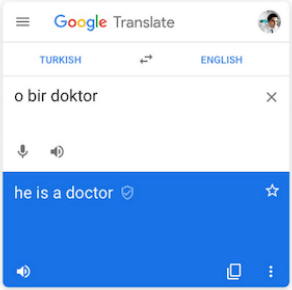
Some famous neural network use cases

Voice Assistants



Photo/Video/AR

Translators



The images are obtained from official websites or blogposts of the services.

Current mobile devices contain dozens of neural networks, some of them running non-stop!

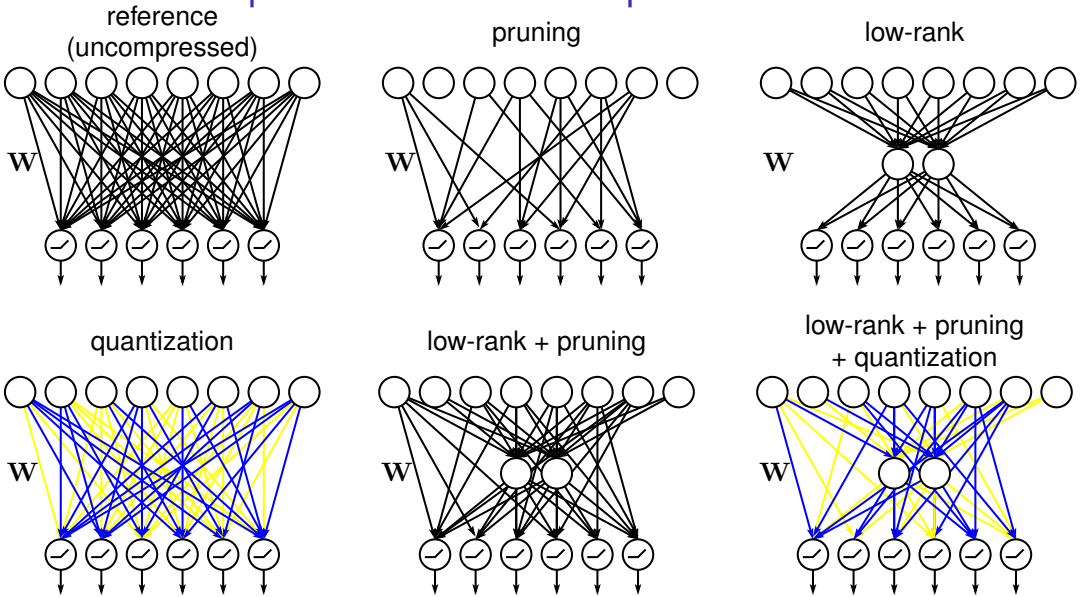
The improvement of NN performance and model compression

The main factor behind the significant improvement in NN accuracies is larger size and more computation!

- While largest neural networks 5 years ago contained millions of parameters (VGG16: 138M) current networks routinely have billions (GPT3: 175B), an exponential growth!
- Inference using such models is expensive and requires high-end hardware
- What if you want to save money or simply do not have such hardware?

This motivates the problem of model compression: reduce model's size/energy consumptions/computational requirements while maintaining the accuracy

The fundamental problem of model compression: what to choose?



Challenges

In principle, we want to explore all possible combinations, and select the best. But:

- Many compression schemes $\implies$ many algorithms
- How to maintain a library of many compressions?
- How to make it user friendly?
  - many algorithms $\implies$ many failure points
- How to make it extensible and easily maintainable?

We propose a software based on the Learning-Compression (LC) algorithm:

- single algorithm—many compressions (builtin reusability)
- extensible, modular, and fast
- impressive compression results
- based on solid optimization principles
- open source: `BSD 3-clause` license

The LC algorithm: MCCO formulation

Given a network with weights $\mathbf{w}$ and loss L:

uncompressed weights

low-dim. params

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

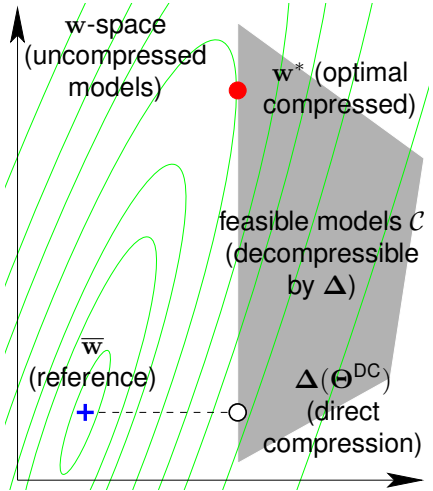
task loss

decompression mapping

$\boldsymbol{\Delta}\colon \boldsymbol{\Theta} \rightarrow \mathbf{w} \in \mathbb{R}^P$

Compression details are abstracted in $\boldsymbol{\Delta}(\boldsymbol{\Theta})$:

- low-rank: $\boldsymbol{\Delta}(\boldsymbol{\Theta}) = \mathbf{U}\mathbf{V}^T$ where $\boldsymbol{\Theta} = \{\mathbf{U}, \mathbf{V}\}$
- pruning: $\boldsymbol{\Delta}(\boldsymbol{\Theta}) = \boldsymbol{\Theta}$ s.t. $\|\boldsymbol{\Theta}\|_0 \leq \kappa$



feasible set $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^P \colon \mathbf{w} = \boldsymbol{\Delta}(\boldsymbol{\Theta}) \text{ for } \boldsymbol{\Theta} \in \mathbb{R}^Q\}$ when $\lambda = 0$

The LC algorithm (cont.)

Reformulate using penalty methods and optimize the following while driving $\mu \to \infty$:

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

Apply alternating optimization wrt $\mathbf{w}$ and $\boldsymbol{\Theta}$, which gives the (LC) algorithm:

- Learning (L) step:

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

  - 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:

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

  - C step has a form of optimal projection of the weights and independent of the dataset.
  - Many well studied cases with fast solutions

The LC algorithm: pseudocode

input training data and model with parameters $\mathbf{w}$

$\mathbf{w} \leftarrow \overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$ pretrained model

$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{\mathrm{DC}} = \boldsymbol{\Pi}(\overline{\mathbf{w}})$ init compression

$\boldsymbol{\beta} \leftarrow \mathbf{0}$

for $\mu = \mu_0 < \mu_1 < \cdots < \infty$

  $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}) - \frac{1}{\mu}\boldsymbol{\beta}\|^2$ L step

  $\boldsymbol{\Theta} \leftarrow \arg\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \frac{1}{\mu}\boldsymbol{\beta} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 + \lambda\, C(\boldsymbol{\Theta})$ C step

  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}))$ multipliers step

  if $\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|$ is small enough **then** exit the loop

return $\mathbf{w}, \boldsymbol{\Theta}$

```
class LCAlgorithm():
  # Housekeeping code skipped
  def run(self):
    self.mu = 0
    self.c_step(step_number=0)
    for i, mu in enumerate(self.mu_schedule):
      self.mu = mu
      self.l_step(i) # user defined
      self.c_step(i) # library call
      self.multipliers_step()
```

The LC software

- Written in `python` using `NumPy` and `PyTorch`
- L step We hand off the L step to the user through the lambda functions.

```
def my_l_step(model, lc_penalty, args**):
    # ...
    loss = model.loss(out_, target_) + lc_penalty()
    loss.backward()
    optimizer.step()
    # ...
```

- C step Many compression are implemented, and you can chose any. If desired, you can add your own compression too by extending the `CompressionTypeBase`: class.

```
class ScaledBinaryQuantization(CompressionTypeBase):
    def compress(self, data):
        a = np.mean(np.abs(data))
        quantized = 2 * a * (data > 0) - a
        return quantized
```

The LC software: currently implemented compressions

Type	Forms
Quantization	Adaptive Quantization into $\{c_1, c_2, \ldots c_K\}$ Binarization into $\{-1, 1\}$ and $\{-c, c\}$ Ternarization into $\{-c, 0, c\}$
Pruning	ℓ_0-constraint (s.t., $\|\mathbf{w}\|_0 \leq \kappa$) ℓ_1-constraint (s.t., $\|\mathbf{w}\|_0 \leq \kappa$) ℓ_0-penalty ($\alpha\|\mathbf{w}\|_0$) ℓ_1-penalty ($\alpha\|\mathbf{w}\|_1$)
Low-rank	Low-rank compression to a given rank Low-rank with *automatic* rank selection for FLOPs reduction Low-rank with *automatic* rank selection for storage compression
Additive Combinations	Quantization + Pruning Quantization + Low-rank Pruning + Low-rank Quantization + Pruning + Low-rank

The LC software: the ease of exploration

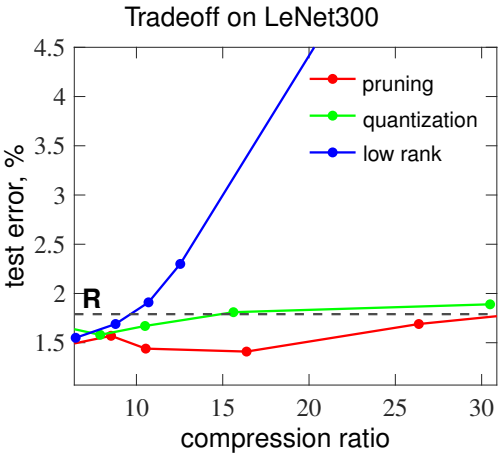
- Mix-and-match through compression tasks. We structured the software in such way that any compression can be applied to any part of the model, and you can mix them as well!
  For example, the following semantics:

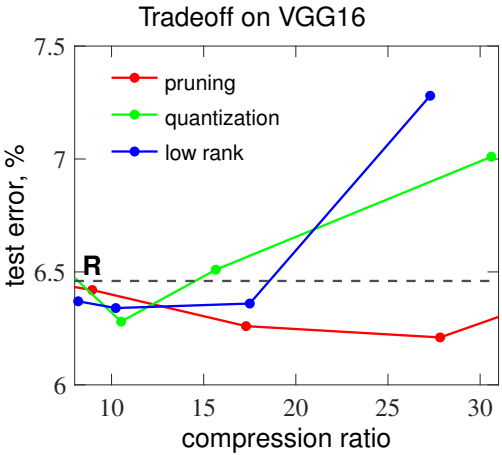
  (layer 1, layer 3) $\rightarrow$ adaptive quantization $k = 6$
  (layer 2) $\rightarrow$ low-rank with $r = 3$

  Translates into the following code:

```
from lc.torch import ParameterTorch as P, AsVector, AsIs
compression_tasks = {
  P([l1.weight, l3.weight]): (AsVector, AdaptiveQuantization(k=6)),
  P(l2.weight):              (AsIs,     LowRank(target_rank=3))
}
```

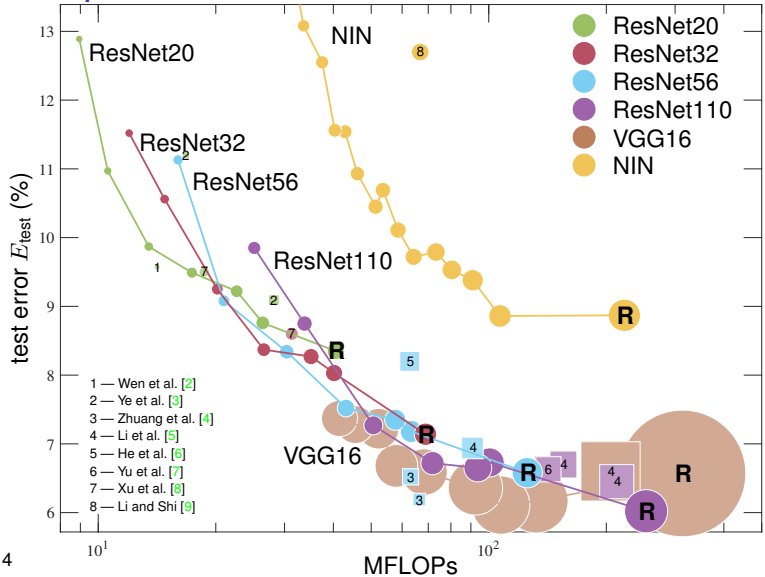
Example: Apples-to-apples comparison between compressions





See more on it in [1]

Example: Tradeoff curves for CIFAR10 networks using low-rank compression



Error-compression space of test error, inference FLOPs and number of parameters (ball size for each net). Different networks have different colors. **R** — reference network.

Results of our algorithm over different λ values for a given network span a curve, shown as connected circles ●—●

Other published results using low-rank are shown as isolated circles; filter pruning results shown as isolated squares.

Example: Low-rank AlexNet models with automatic rank selection

Our framework achieves competitive results in many compression schemes.
For example, using our code for rank-selection, we can achieve considerable speed-up on AlexNet:

	MFLOPs	top-1	top-5	ρ_{FLOPs}
Caffe-AlexNet [10]	724	42.70	19.80	1.00
our scheme 1, $\lambda = 0.20$	231	41.56	18.72	3.13
our scheme 2, $\lambda = 0.20$	**152**	**41.03**	**18.23**	**4.78**
Kim et al. [11], Tucker	272	n/a	21.67	2.66
Tai et al. [12], scheme 2	185	n/a	20.34	3.90
Wen et al. [2], scheme 1	269	n/a	20.14	2.69
Kim et al. [13], scheme 2	272	43.40	20.10	2.66
Yu et al. [7], filter prun.	232	44.13	n/a	3.12
Li et al. [14], filter prun.	334	43.17	n/a	2.16
Ding et al. [15], filter prun.	492	43.83	20.47	1.47

Compression with our algorithm vs published work using low-rank methods and structured pruning.

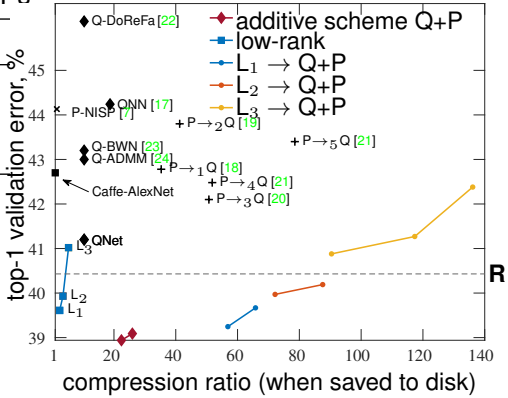
ρ_{FLOPs} — reduction in FLOPs.

See [16] for full details

Example: Additive compressions to achieve smallest AlexNets

The frameworks and software allows easy exploration of new compressions. For example, how about additive combination of quantization and pruning?

	Model	top-1	size, MB	MFLOPs
	Caffe-AlexNet Jia et al. [10]	42.70	243.5	724
ours	$L_1 \to$ Q (1-bit) + P (0.25M)	**39.67**	3.7	228
ours	$L_2 \to$ Q (1-bit) + P (0.25M)	40.19	2.8	185
ours	$L_3 \to$ Q (1-bit) + P (0.25M)	41.27	**2.1**	**152**
	AlexNet-QNN of Wu et al. [17]	44.24	13.0	175
	P$\to_1$Q of Han et al. [18]	42.78	6.9	724
	P$\to_2$Q of Choi et al. [19]	43.80	5.9	724
	P$\to_3$Q of Tung and Mori [20]	42.10	4.8	724
	P$\to_4$Q of Yang et al. [21]	42.48	4.7	724
	P$\to_5$Q of Yang et al. [21]	43.40	3.1	724
	filter pruning of Li et al. [14]	43.17	232.0	334



More on it in [25]

Source code and library 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
- requires minimal coding
- based on solid optimization principles
- single algorithm—many compressions
- time proven (since 2017), with many publications [1, 16, 26, 27, 28, 29, 30, 31]
- we welcome comments, bug reports, requests for functionality, contributions, etc.

References

[1] Y. Idelbayev and M. Á. Carreira-Perpiñán, "An empirical comparison of quantization, pruning and low-rank neural network compression using the LC toolkit."
[2] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li, "Coordinating filters for faster deep neural networks."
[3] J. Ye, X. Lu, Z. Lin, and J. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers."
[4] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks."
[5] H. Li, A. Kadav, I. Durdanovic, and H. P. Graf, "Pruning filters for efficient ConvNets."
[6] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks."
[7] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "NISP: Pruning networks using neuron importance score propagation."
[8] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks."
[9] C. Li and C. J. R. Shi, "Constrained optimization based low-rank approximation of deep neural networks."
[10] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding."
[11] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications."
[12] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization."
[13] H. Kim, M. U. K. Khan, and C.-M. Kyung, "Efficient neural network compression."
[14] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun, "OICSR: Out-in-channel sparsity regularization for compact deep neural networks."
[15] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan, "Approximated oracle filter pruning for destructive CNN width optimization."
[16] Y. Idelbayev and M. Á. Carreira-Perpiñán, "Low-rank compression of neural nets: Learning the rank of each layer."
[17] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices."
[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding."
[19] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization."
[20] F. Tung and G. Mori, "CLIP-Q: Deep network compression learning by in-parallel pruning-quantization."
[21] H. Yang, S. Gui, Y. Zhu, and J. Liu, "Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach."
[22] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients."
[23] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: ImageNet classification using binary convolutional neural networks."
[24] C. Leng, H. Li, S. Zhu, and R. Jin, "Extremely low bit neural network: Squeeze the last bit out with ADMM."
[25] Y. Idelbayev and M. Á. Carreira-Perpiñán, "More general and effective model compression via an additive combination of compressions."
[26] M. Á. Carreira-Perpiñán, "Model compression as constrained optimization, with application to neural nets. Part I: General framework."
[27] M. Á. Carreira-Perpiñán and Y. Idelbayev, "Model compression as constrained optimization, with application to neural nets. Part II: Quantization."
[28] ——, ""Learning-compression" algorithms for neural net pruning."
[29] Y. Idelbayev and M. Á. Carreira-Perpiñán, "Neural network compression via additive combination of reshaped, low-rank matrices."
[30] ——, "Optimal selection of matrix shape and decomposition scheme for neural network compression."
[31] ——, "Beyond FLOPs in low-rank compression of neural networks: Optimizing device-specific inference runtime."