Model Compression As Constrained Optimization, with Application to Neural Nets

Miguel Á. Carreira-Perpiñán mcarreira-perpinan@ucmerced.edu
EECS, University of California, Merced
Yerlan Idelbayev yidelbayev@ucmerced.edu
EECS, University of California, Merced

Abstract

Compressing neural nets is an active research problem, given the large size of state-of-the-art nets for tasks such as object recognition, and the computational limits imposed by mobile devices. Firstly, we give a general formulation of model compression as constrained optimization. This makes the problem of model compression well defined and amenable to the use of modern numerical optimization methods. Our formulation includes many types of compression: quantization, low-rank decomposition, pruning, lossless compression and others. Then, we give a general algorithm to optimize this nonconvex problem based on a penalty function (quadratic penalty or augmented Lagrangian) and alternating optimization. This results in a "learning-compression" algorithm, which alternates a learning step of the uncompressed model, independent of the compression type, with a compression step of the model parameters, independent of the learning task. This algorithm is guaranteed to find the best compressed model for the task under standard assumptions. It is simple to implement in existing deep learning toolboxes and efficient, with a runtime comparable to that of training a reference model in the first place.

1 Motivation

Neural nets have achieved spectacular practical successes in hard problems in computer vision, speech and language in recent years. However, these neural nets are very large (upwards of many millions of weights), which makes it difficult to deploy them in mobile phones or other devices with limited computation, memory, bandwidth or battery life. This motivates the need for compressing a neural net while minimally hurting its performance.

Neural nets are overparameterized in practice, so even simple compression approaches achieve remarkably high compression—such as quantizing the net or pruning small weights and then retraining the resulting net. Many algorithms have been proposed since the 1980s for neural net compression which consider a specific type of compression (such as pruning or binarization) and usually combine some type of rounding or truncation operation with the backpropagation algorithm (e.g. [4–8, 10, 11]). Although these works achieve significant compression, they apply only to the particular type of compression they were designed for, and often have no guarantees of converging to an optimally compressed net (or of converging at all).

What distinguishes our approach? Firstly, we provide a generic formulation for the problem of optimally compressing a model, independent of the compression type. This puts the problem of compression in a sound mathematical footing, amenable to modern optimization techniques. Then, we give a generic training algorithm to find the compressed model with lowest loss. All this algorithm requires is the usual SGD training of the original model; and access to a black-box compression routine for the desired compression type (k-means for quantization, SVD for low-rank, etc.). The combination of these two elements in our learning-compression algorithm gives rise to a convergent algorithm with guarantees of finding the optimal model (under some standard assumptions).

This brings several practical advantages: a simple algorithm that can be easily integrated in a deep learning toolbox; general applicability, in being able to handle most compression techniques in a common framework; and performance, in achieving more compression for the same target loss than other, approximate algorithms. Our algorithm also opens the door for more sophisticated uses of compression along memory, time, energy and other dimensions constrained by the hardware.

OPTML 2017: 10th NIPS Workshop on *Optimization for Machine Learning* (NIPS 2017).

Next, we summarize our general approach and mention its application to quantization and pruning. For further details see papers [1–3].

2 A constrained optimization formulation of model compression

Assume we have previously trained a model with weights $\overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$. This is our *reference* model, which represents the best loss we can achieve without compression. We define *compression* as finding a low-dimensional parameterization $\boldsymbol{\Delta}(\boldsymbol{\Theta})$ of $\mathbf{w}$ in terms of $Q < P$ parameters $\boldsymbol{\Theta}$. *We seek a $\boldsymbol{\Theta}$ such that its corresponding model has (locally) optimal loss. We denote this "optimal compressed" and write it as $\boldsymbol{\Theta}^*$ and* $\mathbf{w}^* \equiv \boldsymbol{\Delta}(\boldsymbol{\Theta}^*)$ (see fig. 1 right). We define *model compression as a constrained optimization* problem:

$$\min_{\mathbf{w},\boldsymbol{\Theta}} L(\mathbf{w}) \quad \text{s.t.} \quad \mathbf{w} = \boldsymbol{\Delta}(\boldsymbol{\Theta}). \tag{1}$$

Compression and decompression are usually seen as algorithms, but here we regard them as mathematical mappings in parameter space. The *decompression mapping* $\boldsymbol{\Delta}$: $\boldsymbol{\Theta} \in \mathbb{R}^Q \rightarrow \mathbf{w} \in \mathbb{R}^P$ maps a low-dimensional parameterization to uncompressed model weights. The *compression mapping* $\boldsymbol{\Pi}(\mathbf{w}) = \arg\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$ behaves as its "inverse" and appears in the C step of our learning-compression algorithm. The C (compression) step is itself carried out by an algorithm: SVD for low-rank compression, k-means for quantization, etc. The *feasible set* $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^P \text{: } \mathbf{w} = \boldsymbol{\Delta}(\boldsymbol{\Theta}) \text{ for } \boldsymbol{\Theta} \in \mathbb{R}^Q\}$ contains all high-dimensional models $\mathbf{w}$ that can be obtained by decompressing some low-dimensional model $\boldsymbol{\Theta}$. In our framework, *compression is equivalent to orthogonal projection on the feasible set.*

Our framework includes well-known types of compression (and combinations thereof), such as:

- *Low-rank compression* defines $\boldsymbol{\Delta}(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}^T$, where we write the weights in matrix form with $\mathbf{W}$ of $m \times n$, $\mathbf{U}$ of $m \times r$ and $\mathbf{V}$ of $n \times r$, and with $r < \min(m, n)$. The compression mapping is given by the singular value decomposition (SVD) of $\mathbf{W}$.
- *Quantization* uses a discrete mapping $\boldsymbol{\Delta}$ given by assigning each weight to one of K codebook values. The compression mapping is given by k-means (or by a form of rounding if we use a fixed codebook, such as $\{-1, +1\}$ or $\{-1, 0, +1\}$).
- *Pruning* defines $\mathbf{w} = \boldsymbol{\Delta}(\boldsymbol{\theta}) = \boldsymbol{\theta}$ where $\mathbf{w}$ is real and $\boldsymbol{\theta}$ is constrained to have few nonzero values, e.g. by using a sparsifying norm such as $\|\boldsymbol{\theta}\|_0 \leq \kappa$ or $\|\boldsymbol{\theta}\|_1 \leq \kappa$. The compression mapping involves some kind of thresholding.

3 A "Learning-Compression" (LC) algorithm

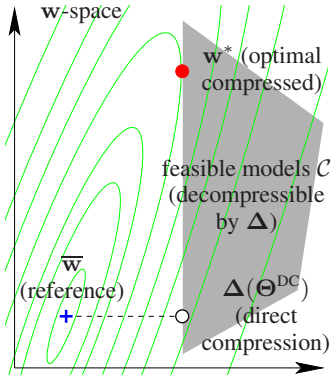
Problem (1) is nonconvex for two reasons. First, the original problem of training the reference model (i.e., minimizing $L(\mathbf{w})$) is already nonconvex for many models, such as deep nets. This makes the objective function of (1) nonconvex. Second, the decompression mapping $\boldsymbol{\Delta}(\boldsymbol{\Theta})$ typically adds and extra level of nonconvexity, often caused by an underlying combinatorial problem (e.g. the choice of best subset of weights to prune, or the choice of assignments of weight-to-codebook-entries in quantization). This makes the feasible set of (1) nonconvex.

Although this constrained problem can be solved with a number of nonconvex optimization algorithms, it is key to profit from parameter separability, which we achieve with penalty methods (quadratic penalty or augmented Lagrangian) and alternating optimization. For simplicity, here we give the quadratic penalty version. This minimizes the following function steps while slowly driving the penalty parameter $\mu \rightarrow \infty$:

$$Q(\mathbf{w}, \boldsymbol{\Theta}; \mu) = L(\mathbf{w}) + \tfrac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2. \tag{2}$$

We minimize Q over $\mathbf{w}$ and $\boldsymbol{\Theta}$ using alternating optimization. This results in an algorithm that alternates two generic steps:

- **L (learning) step:** $\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$. This is a regular training of the uncompressed model but with a quadratic regularization term. *This step is independent of the compression type.*
- **C (compression) step:** $\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 \Leftrightarrow \boldsymbol{\Theta} = \boldsymbol{\Pi}(\mathbf{w})$. This means finding the best (lossy) compression of $\mathbf{w}$ (the current uncompressed model) in the ℓ_2 sense (orthogonal projection on the feasible set), and corresponds to our definition of the compression mapping $\boldsymbol{\Pi}$. *This step is independent of the loss, training set and task.*



input training data and model with parameters $\mathbf{w}$
$\mathbf{w} \leftarrow \overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$ — reference model
$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{\mathrm{DC}} = \boldsymbol{\Pi}(\overline{\mathbf{w}}) = \arg\min_{\boldsymbol{\Theta}} \|\overline{\mathbf{w}} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$ — compress ref.
for $\mu = \mu_0 < \mu_1 < \cdots < \infty$
  $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$ — L step: learn
  $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Pi}(\mathbf{w}) = \arg\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$ — C step: compress
  if $\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|$ is small enough **then** exit the loop
return $\mathbf{w}$, $\boldsymbol{\Theta}$

Figure 1: *Left*: illustration of the idea of model compression by constrained optimization. *Right*: pseudocode for the LC algorithm (quadratic-penalty version).

Fig. 1 (right) gives the LC algorithm pseudocode for the quadratic penalty. The LC algorithm defines a continuous path $(\mathbf{w}(\mu), \boldsymbol{\Theta}(\mu))$ which, under some mild assumptions, converges to a stationary point (typically a minimizer) of the constrained problem (see theorem 3.1). The beginning of this path, for $\mu \to 0^+$, corresponds to training the reference model and then compressing it disregarding the loss (*direct compression*), a simple but suboptimal approach that is popular in practice.

Convergence results for the LC algorithm The following theorem (proven in [1]) shows convergence to a stationary point of the constrained problem (1), for the quadratic-penalty version of the algorithm. Assume the loss $L(\mathbf{w})$ and the decompression mapping $\boldsymbol{\Delta}(\boldsymbol{\Theta})$ are continuously differentiable wrt their arguments, and that the loss is lower bounded.

Theorem 3.1. *Consider the constrained problem (1) and its quadratic-penalty function $Q(\mathbf{w}, \boldsymbol{\Theta}; \mu)$ of (2). Given a positive increasing sequence $(\mu_k) \to \infty$, a nonnegative sequence $(\tau_k) \to 0$, and a starting point $(\mathbf{w}^0, \boldsymbol{\Theta}^0)$, suppose the QP method finds an approximate minimizer $(\mathbf{w}^k, \boldsymbol{\Theta}^k)$ of $Q(\mathbf{w}^k, \boldsymbol{\Theta}^k; \mu_k)$ that satisfies $\|\nabla_{\mathbf{w},\boldsymbol{\Theta}} Q(\mathbf{w}^k, \boldsymbol{\Theta}^k; \mu_k)\| \leq \tau_k$ for $k = 1, 2, \ldots$ Then, $\lim_{k\to\infty} \big((\mathbf{w}^k, \boldsymbol{\Theta}^k)\big) = (\mathbf{w}^*, \boldsymbol{\Theta}^*)$, which is a KKT point for the problem (1), and its Lagrange multiplier vector has elements $\boldsymbol{\lambda}_i^* = \lim_{k\to\infty} \big(-\mu_k (\mathbf{w}_i^k - \boldsymbol{\Delta}(\boldsymbol{\Theta}_i^k))\big)$, $i = 1, \ldots, P$.*

This theorem applies to a number of common losses used in machine learning, and to various compression techniques, such as low-rank factorization, which are continuously differentiable. It does not apply to some popular compression forms, specifically quantization and pruning, which give rise to NP-complete problems. While we cannot expect the LC algorithm to find the global optimum of these problems, we can expect reasonably good results in that it will converge to a feasible point (hence, a validly compressed model) with likely low loss.

Optimizing the L step The L step always takes the same form regardless of the type of compression (the information from the latter is contained in the numerical values of $\boldsymbol{\Delta}(\boldsymbol{\Theta})$, which are constant in the L step). For neural nets, the L step can be solved by stochastic gradient descent in the same way that the reference net was trained, except that we should clip the step sizes so they never exceed $\frac{1}{\mu}$, in order to prevent oscillations as $\mu \to \infty$ (see [1]).

Optimizing the C step The C step does not depend on the loss or training data, but it does depend on the type of compression. Its solution must be worked out for each case by solving for the compression mapping $\boldsymbol{\Pi}(\mathbf{w}) = \arg\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$. Typically, the solution is given by a well-known compression algorithm—this is an intentional consequence of our design of the LC algorithm. Hence, the C step can be solved by calling a compression routine corresponding to the desired compression type (which minimizes the decompression error in the least-squares sense). For example, for low-rank compression of a weight matrix we run the SVD and truncate all but the largest r singular values; for quantization with an adaptive codebook we run k-means with a codebook of K weights; for pruning, we prune all but the top-k weights (where k depends on the sparsifying norm used); we give details elsewhere [2, 3]. If new compression techniques are discovered, we may be able to incorporate them in the LC algorithm by solving the C step they define.

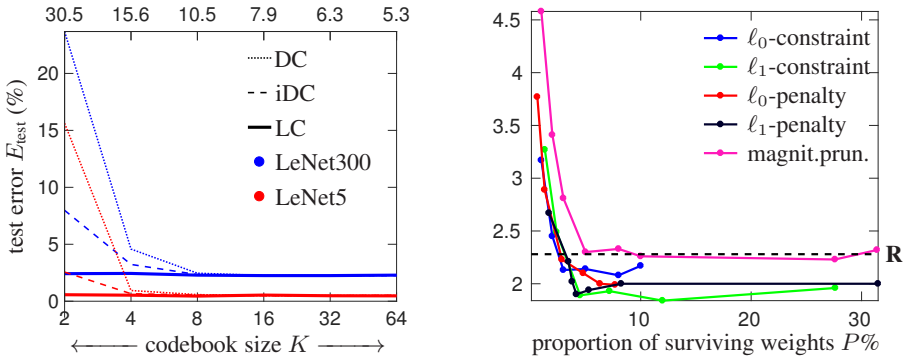


Figure 2: Error-compression curves for LeNet neural nets using quantization (left) and pruning (right).

The runtime of the C step is typically negligible compared to that of the L step (which involves the actual training set, usually much larger than the number of parameters). The overall runtime of the LC algorithm will be dominated by the L steps, as if we were training an uncompressed model for a longer time.

4 Experiments

Figure 2 shows compression results (as a tradeoff curve of error vs compression level) for a well-known benchmark, the LeNet neural nets [9]. We can use a single bit per weight (for quantization) or remove nearly 99% of the weights (for pruning) with no error degradation over the reference. See Carreira-Perpiñán and Idelbayev [2, 3] for more results and larger nets.

References

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

[2] M. Á. Carreira-Perpiñán and Y. Idelbayev. Model compression as constrained optimization, with application to neural nets. Part II: Quantization. arXiv:1707.04319 [cs.LG], July 13 2017.

[3] M. Á. Carreira-Perpiñán and Y. Idelbayev. Model compression as constrained optimization, with application to neural nets. Part III: Pruning. arXiv, Sept. 2017.

[4] M. Courbariaux, Y. Bengio, and J.-P. David. BinaryConnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 3105–3113. MIT Press, Cambridge, MA, 2015.

[5] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 27, pages 1269–1277. MIT Press, Cambridge, MA, 2014.

[6] E. Fiesler, A. Choudry, and H. J. Caulfield. Weight discretization paradigm for optical neural networks. In *Proc. SPIE 1281: Optical Interconnections and Networks*, pages 164–173, The Hague, Netherlands, Aug. 1 1990.

[7] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[8] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 1135–1143. MIT Press, Cambridge, MA, 2015.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, Nov. 1998.

[10] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4): 473–493, July 1992.

[11] R. Reed. Pruning algorithms—a survey. *IEEE Trans. Neural Networks*, 4(5):740–747, Sept. 1993.