

Model compression as constrained optimization, with application to neural nets. Part I: general framework.

Miguel Á. Carreira-Perpiñán
Electrical Engineering and Computer Science, University of California, Merced
<http://eecs.ucmerced.edu>

July 4, 2017

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. We give a general formulation of model compression as constrained optimization. This 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 the 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 simple, efficient algorithm is guaranteed to find the best compressed model for the task in a local sense under standard assumptions.

We present separately in several companion papers the development of this general framework into specific algorithms for model compression based on quantization, pruning and other variations, including experimental results on compressing neural nets and other models.

1 Introduction

Large neural network models have become a central component in state-of-the-art practical implementations of the solution to various machine learning and artificial intelligence problems. These include, for example, classification problems involving images, audio or text, or reinforcement learning problems involving game playing or robot manipulation and navigation. This has also resulted in an enormous increase in the interest in deep neural net models from researchers in academy and industry, and even from non-experts and the public in general, as evidenced by the amount of scientific papers, blog entries or mainstream media articles published about it.

These practical successes in difficult problems have occurred thanks to the availability of large-scale datasets and of massive computational power provided by GPUs, which are particularly well suited for the kind of linear algebra operations involved in training a neural net (such as stochastic gradient descent). One notable characteristic of deep neural nets that seems to distinguish them from other machine learning models is their ability to grow with the data. That is, as the size of the training set grows, we can continue to increase the (say) classification accuracy by increasing the size of the neural net (number of hidden units and of layers, and consequently the number of weights). This is unlike, for example, a linear model, whose classification accuracy will quickly stagnate as the data keeps growing. Hence, we can continue to improve the accuracy of a neural net by making it bigger and training on more data. This means that we can expect to see ever larger neural nets in future practical applications. Indeed, models reported in the literature of computer vision have gone from less than a million weights in the 1990s to millions in the 2000s and, in recent works, to models exceeding billions of weights (each a floating-point value).

The large size of the resulting model does cause an important practical problem when one intends to deploy it in a resource-constrained target device such as a mobile phone or other embedded systems. That

is, the large neural net is trained in a resource-rich setting, e.g. GPUs and a multicore architecture with large memory and disk, where the model designer can explore different model architectures, hyperparameter values, etc. until a model with the best accuracy on a validation set is found. This final, large model is then ready to be deployed in practice. However, the target device will typically have far more restrictive computation constraints in memory size and speed, arithmetic operations, clock rate, energy consumption, etc., which make it impossible to accommodate the large model. In other words, we can only deploy models of a certain size. Download bandwidth is also significantly limited in apps for mobile phones or software for cars.

This problem has attracted considerable attention recently. Two important facts weigh upon it which have been known for some time among researchers. The first is that the large neural nets that we currently know how to train for optimal accuracy contain significant redundancy, which makes it possible to find smaller neural nets with comparable accuracy. The second is that, for reasons not entirely understood, one typically achieves a more accurate model by training a large model first and then somehow transforming it into a smaller one (“compressing” the model), than by training a small model in the first place. This leads us to the problem of *compressing a neural net* (or other model), which is our focus.

Compressing neural nets has been recognized in recent years as an important problem and various academic and industrial research groups have shown that one can indeed significantly compress neural nets without appreciable losses in accuracy (see related work below). However, the solutions proposed so far are somewhat ad-hoc in two senses. First, they define a specific compression technique (and a specific algorithm to find the compressed model) which may work well with some types of models but not others. Second, some of these solutions are not guaranteed to be optimal in the sense of achieving the highest classification accuracy for the compression technique considered.

In this paper, we provide a general formulation of model compression and a training algorithm to solve it. The formulation can accommodate any compression technique as long as it can be put in a certain mathematical form, which includes most existing techniques. The compression mechanism appears as a black box, so that the algorithm simply iterates between learning a large model and compressing it, but is guaranteed to converge to a locally optimal compressed model under some standard assumptions. In separate papers, we develop specific algorithms for various compression forms, such as quantization (Carreira-Perpiñán and Idelbayev, 2017a) or pruning (Carreira-Perpiñán and Idelbayev, 2017b), and evaluate them experimentally. In the rest of this paper, we discuss related work (section 2), give the generic formulation (section 3) and the generic LC algorithm (section 4), give conditions for convergence (section 5) and discuss the relation of our LC algorithm with other algorithms (section 6) and the relation with generalization and model selection (section 7).

2 Related work: what does it mean to compress a model?

In a general sense, we can understand model compression as replacing a “large” model with a “small” model within the context of a given task, and this can be done in different ways. Let us discuss the setting of the problem that motivates the need for compression.

Assume we define the machine learning task as classification for object recognition from images and consider as large model $\mathbf{f}(\mathbf{x}; \mathbf{w})$: $\mathcal{X} \rightarrow \mathcal{Y}$ a deep neural net with inputs $\mathbf{x} \in \mathcal{X}$ (images), outputs $\mathbf{y} \in \mathcal{Y}$ (object class labels) and real-valued weights $\mathbf{w} \in \mathbb{R}^P$ (where the number of weights P is large). In order to train a model we use a loss $L()$, e.g. the cross-entropy on a large training set of input-output pairs $(\mathbf{x}_n, \mathbf{y}_n)$. Also assume we have trained a large, *reference* neural net $\mathbf{f}(\mathbf{x}; \bar{\mathbf{w}})$, i.e., $\bar{\mathbf{w}} = \arg \min_{\mathbf{w}} L(\mathbf{f}(\cdot; \mathbf{w}))$, and that we are happy with its performance in terms of accuracy, but its size is too large.

We now want a smaller model $\mathbf{h}(\mathbf{x}; \Theta)$: $\mathcal{X} \rightarrow \mathcal{Y}$ that we can apply to the same task (classifying input images \mathbf{x} into labels \mathbf{y}). How should we define this smaller model? One possibility is for the small model $\mathbf{h}(\mathbf{x}; \Theta)$ to be of the same type as the reference model \mathbf{f} but reparameterized in terms of a low-dimensional parameter vector. That is, we construct weights $\mathbf{w} \in \mathbb{R}^P$ from low-dimensional parameters $\Theta \in \mathbb{R}^Q$ via some transformation $\mathbf{w} = \Delta(\Theta)$ so that the size of Θ is smaller than the size of \mathbf{w} , i.e., $Q < P$ (obviously, this will constrain the possible $\mathbf{w} \in \mathbb{R}^P$ that can be constructed). In the example before, $\mathbf{h}(\mathbf{x}; \Theta)$ would be the same deep net but, say, with weight values $\mathbf{w} = \Delta(\Theta)$ quantized using a codebook with K entries (so Θ contains the codebook and the assignment of each weight w_i to a codebook entry). A second possibility

is for the small model $\mathbf{h}(\mathbf{x}; \Theta)$ to be a completely different type of model from the reference, e.g. a linear mapping with parameters Θ (so there is no relation between \mathbf{w} and Θ).

Given this, we have the following options to construct the small model $\mathbf{h}(\mathbf{x}; \Theta)$:

- *Direct learning*: $\min_{\Theta} L(\mathbf{h}(\mathbf{x}; \Theta))$: *find the small model with the best loss regardless of the reference.* That is, simply train the small model to minimize the loss directly, possibly using the chain rule to compute the gradient wrt Θ . This approximates the reference model indirectly, in that both \mathbf{f} and \mathbf{h} are trying to solve the same task. We can have the small model \mathbf{h} be a reparameterized version of \mathbf{f} or a completely different model. Direct learning is the best thing to do sometimes but not always, as noted later.
- *Direct compression (DC)*: $\min_{\Theta} \|\bar{\mathbf{w}} - \Delta(\Theta)\|^2$: *find the closest approximation to the parameters of the reference model, using a low-dimensional parameterization $\Delta(\Theta)$.* This forces \mathbf{h} and \mathbf{f} to be models of the same type. Direct compression can be simply done with (lossless or lossy) compression of $\bar{\mathbf{w}}$, but it generally will not be optimal wrt the loss since the latter is ignored in learning Θ . We discuss this later in detail.
- *Model compression as constrained optimization*: this is our proposed approach, which we describe in section 3. It forces \mathbf{h} and \mathbf{f} to be models of the same type, by constraining the weights \mathbf{w} to be constructed from a low-dimensional parameterization $\mathbf{w} = \Delta(\Theta)$, but \mathbf{h} must optimize the loss L .
- *Teacher-student*: $\min_{\Theta} \int_{\mathcal{X}} p(\mathbf{x}) \|\mathbf{f}(\mathbf{x}; \bar{\mathbf{w}}) - \mathbf{h}(\mathbf{x}; \Theta)\|^2 d\mathbf{x}$: *find the closest approximation \mathbf{h} to the reference function \mathbf{f} , in some norm.* The norm over the domain \mathcal{X} may be approximated with a sample (e.g. the training set). Here, the reference model \mathbf{f} “teaches” the student model \mathbf{h} . We can have the small model \mathbf{h} be a reparameterized version of \mathbf{f} or a completely different model.

Most existing compression approaches fall in one of these categories. In particular, traditional compression techniques have been applied using techniques most related to direct training and direct compression: using low-precision weight representations through some form of rounding (see Gupta et al., 2015; Hubara et al., 2016 and references therein), even single-bit (binary) values (Fiesler et al., 1990; Courbariaux et al., 2015; Rastegari et al., 2016; Hubara et al., 2016; Zhou et al., 2016), ternary values (Hwang and Sung, 2014; Li et al., 2016; Zhu et al., 2017) or powers of two (Marchesi et al., 1993; Tang and Kwan, 1993); quantization of weight values, soft (Nowlan and Hinton, 1992; Ullrich et al., 2017) or hard (Fiesler et al., 1990; Marchesi et al., 1993; Tang and Kwan, 1993; Gong et al., 2015; Han et al., 2015); zeroing weights to achieve a sparse model (Hanson and Pratt, 1989; Weigend et al., 1991; LeCun et al., 1990; Hassibi and Stork, 1993; Reed, 1993; Yu et al., 2012; Han et al., 2015); low-rank factorization of weight matrices (Sainath et al., 2013; Denil et al., 2013; Jaderberg et al., 2014; Denton et al., 2014; Novikov et al., 2015); hashing (Chen et al., 2015); and lossless compression, such as Huffman codes (Han et al., 2016). Some papers combine several such techniques to produce impressive results, e.g. Han et al. (2016) use pruning, trained quantization and Huffman coding. Although we comment on some of these works in this paper (particularly regarding the direct compression approach), we defer detailed discussions to our companion papers on specific compression techniques (quantization in Carreira-Perpiñán and Idelbayev, 2017a and pruning in Carreira-Perpiñán and Idelbayev, 2017b, so far).

The teacher-student approach seems to have arisen in the ensemble learning literature, inspired by the desire to replace a large ensemble model (e.g. a collection of decision trees), which requires large storage and is slow at test time, with a smaller or faster model with similar classification accuracy (Zhou, 2012, section 8.5). The smaller model can be of the same type as the ensemble members, or a different type of model altogether (e.g. a neural net). The basic idea is, having trained the large ensemble on a labeled training set, to use this ensemble (the “teacher”) to label a larger, unlabeled dataset (which may be available in the task at hand, or may be generated by some form of sampling). This larger, labeled dataset is then used to train the smaller model (the “student”). The hope is that the knowledge captured by the teacher is adequately represented in the synthetically created training set (although the teacher’s mistakes will also be represented in it), and that the student can learn it well even though it is a smaller model. More generally, the approach can be applied with any teacher, not necessarily an ensemble model (e.g. a deep net), and the teacher’s labels may be transformed to improve the student’s learning, e.g. log-outputs (Ba and Caruana, 2014) or other manipulations of the output class probabilities (Hinton et al., 2015). However, from a compression

point of view the results are unimpressive, with modest compression ratios (Hinton et al., 2015) or even failure to compress (using a single-layer net as student needs many more weights than the teacher deep net; Ba and Caruana, 2014). One practical problem with the teacher-student approach is that all it really does is construct the artificial training set, but it leaves the design of the student to the user, and this is a difficult model selection problem. This is unlike the compression approaches cited above, which use the teacher’s model architecture but compress its parameters.

3 A constrained optimization formulation of model compression

In this work we understand compression in a mathematically specific sense that involves a learning task that we want to solve and a large model that sets the reference to meet. It is related to the direct training and direct compression concepts introduced above. Assume we have a large, *reference* model $\mathbf{f}(\mathbf{x}; \bar{\mathbf{w}})$ with P parameters (e.g. a neural net with inputs \mathbf{x} and weights $\mathbf{w} \in \mathbb{R}^P$) that has been trained on a *loss* $L()$ (e.g. cross-entropy on a given training set) to solve a *task* (e.g. classification). That is, $\bar{\mathbf{w}} = \arg \min_{\mathbf{w}} L(\mathbf{w})$, where we abuse the notation to write the loss directly over the weights rather than $L(\mathbf{f}(\mathbf{x}; \mathbf{w}))$, and the minimizer $\bar{\mathbf{w}}$ may be local. We define *compression* as finding a low-dimensional parameterization $\Delta(\Theta)$ of \mathbf{w} in terms of $Q < P$ parameters Θ . This will define a *compressed* model $\mathbf{h}(\mathbf{x}; \Theta) = \mathbf{f}(\mathbf{x}; \Delta(\Theta))$. We seek a Θ such that its corresponding model has (locally) optimal loss. We denote this “optimal compressed” and write it as Θ^* and $\mathbf{w}^* \equiv \Delta(\Theta^*)$ (see fig. 1).

Ordinarily, one could then solve the problem directly over Θ : $\Theta^* = \arg \min_{\Theta} L(\Delta(\Theta))$. This is the *direct learning* option in the previous section. Instead, we equivalently write *model compression as a constrained optimization* problem:

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

The reason, which will be useful later to develop an optimization algorithm, is that we decouple the part of *learning* the task, $L(\mathbf{w})$ in the objective, from the part of *compressing* the model, $\mathbf{w} = \Delta(\Theta)$ in the constraints.

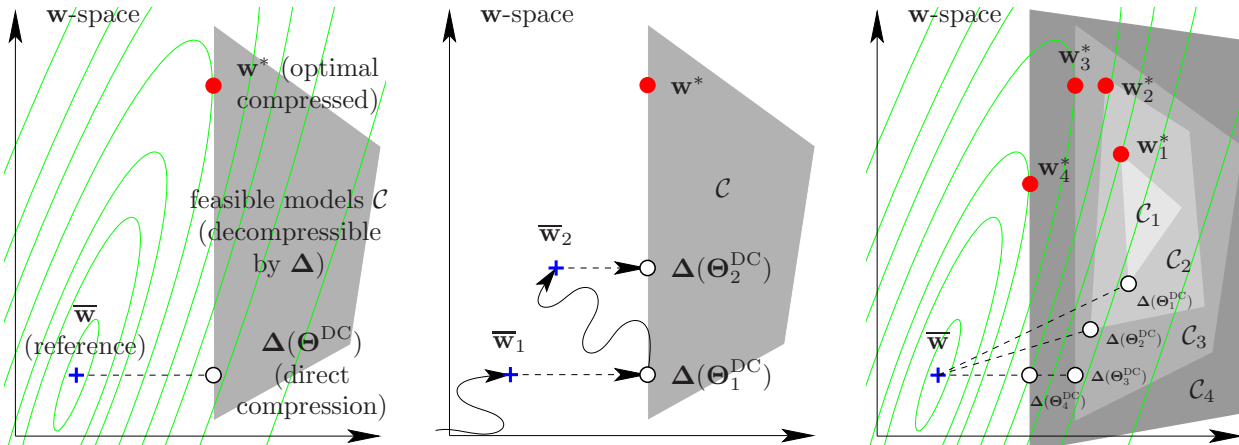


Figure 1: Schematic representation of the idea of model compression by constrained optimization. The plots illustrate the uncompressed model space (\mathbf{w} -space = \mathbb{R}^P), the contour lines of the loss $L(\mathbf{w})$ (green lines), and the set of compressed models (the feasible set $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^P: \mathbf{w} = \Delta(\Theta) \text{ for } \Theta \in \mathbb{R}^Q\}$, grayed areas), for a generic compression technique Δ . The Θ -space is not shown. $\bar{\mathbf{w}}$ optimizes $L(\mathbf{w})$ but is infeasible (no Θ can decompress into it). The direct compression $\mathbf{w}^{\text{DC}} = \Delta(\Theta^{\text{DC}})$ is feasible but not optimal compressed (not optimal in the feasible set). $\mathbf{w}^* = \Delta(\Theta^*)$ is optimal compressed. Plot 2 shows two local optima $\bar{\mathbf{w}}_1$ and $\bar{\mathbf{w}}_2$ of the loss $L(\mathbf{w})$, and their respective DC points (the contour lines are omitted to avoid clutter). Plot 3 shows several feasible sets, corresponding to different compression levels (\mathcal{C}_1 is most compression).

By eliminating \mathbf{w} , our formulation (1) is equivalent to direct learning $\min_{\Theta} L(\Delta(\Theta))$, so why not do that in the first place, rather than training a large model and then compressing it? In fact, direct learning using gradient-based methods (via the chain rule) may sometimes be a good option. But it is not always convenient or possible. Firstly, if the decompression mapping Δ is not differentiable wrt Θ (as happens with quantization), then the chain rule does not apply. Second, using gradient-based methods over Θ may lead to slow convergence or be prone to local optima compared to training a large, uncompressed model (this has been empirically reported with pruning Reed, 1993 and low-rank compression, e.g. Denil et al., 2013). Third, the direct learning does not benefit from the availability of a large, well-trained model in \mathbf{w} -space, since it operates exclusively in the low-dimensional Θ -space. Finally, in direct learning the learning task aspects (loss L and training set) are intimately linked to the compression ones (Δ and Θ), so that the design of a direct learning algorithm is specific to the combination of loss and compression technique (in our LC algorithm, both aspects separate and can be solved independently).

3.1 Compression as orthogonal projection on the feasible set

Compression and decompression are usually seen as algorithms, but here we regard them as mathematical mappings in parameter space. If

$$\mathbf{\Pi}(\mathbf{w}) = \arg \min_{\Theta} \|\mathbf{w} - \Delta(\Theta)\|^2 \quad (2)$$

is well defined, we call $\mathbf{\Pi}: \mathbf{w} \in \mathbb{R}^P \rightarrow \Theta \in \mathbb{R}^Q$ the *compression mapping*. $\mathbf{\Pi}$ behaves as the “inverse” of the *decompression mapping* $\Delta: \Theta \in \mathbb{R}^Q \rightarrow \mathbf{w} \in \mathbb{R}^P$ (although it is not a true inverse, because $\Delta \circ \mathbf{\Pi} \neq \text{identity}$). Since $P > Q$, there generally will be a unique inverse Θ for any given \mathbf{w} (but not necessarily). Compressing \mathbf{w} may need an algorithm (e.g. SVD for low-rank compression, k -means for quantization) or a simple formula (e.g. taking the sign or rounding a real value). $\mathbf{\Pi}$ will usually satisfy $\mathbf{\Pi}(\Delta(\Theta)) = \Theta$ for any $\Theta \in \mathbb{R}^Q$, i.e., decompressing Θ then compressing it gives back Θ . The decompression mapping Δ appears explicitly in our problem definition (1), while the compression mapping $\mathbf{\Pi}$ appears in our LC algorithm (in the C step), as we will see.

With lossless compression, Δ is bijective and $\mathbf{\Pi} = \Delta^{-1}$. With lossy compression, Δ need be neither surjective (since $Q < P$) nor injective (since it can have symmetries, e.g. reordering singular values in the SVD or centroids in k -means). Also, Δ need not be differentiable wrt Θ (for low-rank compression it is, for quantization it is not).

The feasible set $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^P: \mathbf{w} = \Delta(\Theta) \text{ for } \Theta \in \mathbb{R}^Q\}$ contains all high-dimensional models \mathbf{w} that can be obtained by decompressing some low-dimensional model Θ . In our framework, compression is equivalent to orthogonal projection on the feasible set. Indeed, $\mathbf{\Pi}(\mathbf{w}) = \arg \min_{\Theta} \|\mathbf{w} - \Delta(\Theta)\|^2$ is equivalent to $\mathbf{\Pi}(\mathbf{w}) = \arg \min_{\Theta, \mathbf{w}'} \|\mathbf{w} - \mathbf{w}'\|^2$ s.t. $\mathbf{w}' = \Delta(\Theta)$, which is the problem of finding the closest feasible point to \mathbf{w} in Euclidean distance, i.e., $\mathbf{\Pi}(\mathbf{w})$ is the orthogonal projection of \mathbf{w} on the feasible set.

3.2 Types of compression

Our framework includes many well-known types of compression:

Low-rank compression defines $\Delta(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}^T$, 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 $r < \min(m, n)$. If we learn both \mathbf{U} and \mathbf{V} , the compression mapping is given by the singular value decomposition (SVD) of \mathbf{W} . We can also use a fixed dictionary or basis (e.g. given by wavelets or the discrete cosine transform) and learn either \mathbf{U} or \mathbf{V} only. The compression mapping is then given by solving a linear system. We study this in a paper in preparation.

Quantization uses a discrete mapping Δ given by assigning each weight to one of K codebook values. If we learn both the assignments and the codebook, compression can be done by k -means. We can also use a fixed codebook, such as $\{-1, +1\}$ or $\{-1, 0, +1\}$. The compression mapping is then given by a form of rounding. We study this in a separate paper (Carreira-Perpiñán and Idelbayev, 2017a).

Low-precision approximation defines a constraint $w_i = \theta_i$ per weight where w_i is real (or, say, a double-precision float) and θ_i is, say, a single-precision float. The compression mapping sets θ_i to the truncation of w_i . A particular case is binarization, where $\theta_i \in \{-1, +1\}$ and the compression mapping sets $\theta_i = \text{sgn}(w_i)$. This can be seen as quantization using a fixed codebook.

Pruning defines $\mathbf{w} = \Delta(\boldsymbol{\theta}) = \boldsymbol{\theta}$ where \mathbf{w} is real and $\boldsymbol{\theta}$ is constrained to have few nonzero values. The compression mapping involves some kind of thresholding. We study this in a separate paper (Carreira-Perpiñán and Idelbayev, 2017b).

Lossless compression takes many forms, such as Huffman coding, arithmetic coding or run-length encoding (Gersho and Gray, 1992), and is special in that Δ is a bijection. Hence, the direct compression solves the problem with no need for our LC algorithm. However, lossless compression affords limited compression power.

It is also possible to combine several compression techniques.

For any lossy compression technique, the user can choose a *compression level* (e.g. the rank in low-rank approaches or the codebook size in quantization approaches). Obviously, we are interested in the highest compression level that retains acceptable accuracy in the task. Note that in accounting for the size of the compressed model, we need to add two costs: storing the weights of the compressed model, and storing the decompression algorithm data (the dictionary, the codebook, the location of the nonzero values, etc.).

We can be flexible in the definition of the constraints in problem (1). We need not compress all the weights (e.g. we usually do not compress the biases in a neural net); this simply means we have no constraint for each such weight w_i . We can use different types of compression for different parts of the model (e.g. for different layers of the net); this means we have sets of constraints $\mathbf{w}_j = \Delta_j(\boldsymbol{\Theta}_j)$, which separate and can be ran in parallel in the C step (see later). Also, there may be additional constraints in problem (1). For example, in quantization we have binary assignment vectors whose sum must equal one, or variables that must belong to a set such as $\{-1, +1\}$. The original minimization over the loss $L(\mathbf{w})$ may also be constrained, e.g. if the weights should be nonnegative.

The decompression mapping $\mathbf{w} = \Delta(\boldsymbol{\Theta})$ can take different forms. Each w_i may be a function of the entire $\boldsymbol{\Theta}$, which are then “shared” compression parameters. This happens with low-rank compression: $\mathbf{W} = \Delta(\mathbf{U}, \mathbf{V}) = \mathbf{U}\mathbf{V}^T$. Instead, each w_i may have “shared” parameters and “private” parameters ϑ_i . This happens in quantization: $w_i = c_{\vartheta_i}$, where $\{c_1, \dots, c_K\}$ is a codebook shared by all weights and $\vartheta_i \in \{1, \dots, K\}$ is the index in the codebook that w_i is assigned to¹. Here, $\boldsymbol{\Theta} = \{c_1, \dots, c_K\} \cup \{\vartheta_i\}_{i=1}^P$.

Earlier we defined the feasible set $\mathcal{C} = \{\mathbf{w} \in \mathbb{R}^P : \mathbf{w} = \Delta(\boldsymbol{\Theta}) \text{ for } \boldsymbol{\Theta} \in \mathbb{R}^Q\}$, which contains all high-dimensional models \mathbf{w} that can be obtained by decompressing some low-dimensional model $\boldsymbol{\Theta}$. Good compression schemes should satisfy two desiderata:

1. Achieve low compression error. Obviously, this depends on the compression level (which determines the “size” of the feasible set) and on the optimization, e.g. our LC algorithm (which adapts the parameters $\boldsymbol{\Theta}$ to the task at hand as best as possible). But the form of the compression mapping (low-rank, quantization, etc.) matters. This form should be such that every uncompressed model \mathbf{w} of interest is near some part of the feasible set, i.e., the decompression mapping $\Delta(\boldsymbol{\Theta})$ is “space-filling” to some extent.
2. Have simple compression and decompression algorithms: fast and ideally without local optima.

3.3 Other formulations of model compression

A penalty formulation One can define the compression problem as

$$\min_{\mathbf{w}} L(\mathbf{w}) + \lambda C(\mathbf{w}) \tag{3}$$

where the penalty or cost function $C(\mathbf{w})$ encourages \mathbf{w} to be close to a compressed model and $\lambda \geq 0$ is a user parameter. Computationally, this can also be conveniently optimized by duplicating \mathbf{w} :

$$\min_{\mathbf{w}, \boldsymbol{\Theta}} L(\mathbf{w}) + \lambda C(\boldsymbol{\Theta}) \quad \text{s.t.} \quad \mathbf{w} = \boldsymbol{\Theta} \tag{4}$$

and applying a penalty method and alternating optimization, so the learning part on L and \mathbf{w} separates from the compression part on $\boldsymbol{\Theta}$. However, this formulation is generally less preferable than the constrained

¹Actually, it is more convenient to express ϑ_i as a binary assignment vector $\mathbf{z}_i \in \{0, 1\}^K$ where $\sum_{k=1}^K z_{ik} = 1$, see Carreira-Perpiñán and Idelbayev (2017a).

formulation (1). The reason is that the penalty $\lambda C(\mathbf{w})$ does not guarantee that the optimum of (3) is exactly a compressed model, only that it is close to some compressed model, and a final, suboptimal compression step is required. For example, penalizing the deviations of the weights w_i from a given codebook (say, $\{-1, +1\}$) will encourage the weights to cluster around codebook values, but not actually to equal them, so upon termination we must round all weights, which makes the result suboptimal.

For some types of compression a penalty formulation does produce exactly compressed models. In pruning (Carreira-Perpiñán and Idelbayev, 2017b), we want the weight vector \mathbf{w} to contain many zeros (be sparse). Using a penalty $\lambda C(\mathbf{w})$ where C is a sparsity-inducing norm, say $C(\mathbf{w}) = \|\mathbf{w}\|_0$, will result in a sparse weight vector. Still, in the penalty form the number of nonzeros in \mathbf{w} is implicitly related to the value of λ , while the constraint form $\|\mathbf{w}\|_0 \leq \kappa$ allows us to set the number of nonzeros directly, which is more convenient in practice.

Another constrained formulation It is conceivable to consider the following, alternative formulation of model compression as a constrained optimization:

$$\min_{\mathbf{w}, \Theta} L(\mathbf{w}) \quad \text{s.t.} \quad \mathbf{\Pi}(\mathbf{w}) = \Theta \quad (5)$$

directly in terms of a well-defined compression mapping $\mathbf{\Pi}$, rather than in terms of a decompression mapping $\mathbf{\Delta}$ as in (1). This has the advantage that problem (5) is simply solved by setting $\mathbf{w}^* = \bar{\mathbf{w}} \equiv \arg \min_{\mathbf{w}} L(\mathbf{w})$ (the reference model) and $\Theta^* = \mathbf{\Pi}(\bar{\mathbf{w}})$, without the need for an iterative algorithm over \mathbf{w} and Θ (we call this “direct compression” later). Indeed, the constraint in (5) does not actually constrain \mathbf{w} . However, this formulation is rarely useful, because the resulting compressed model may have an arbitrarily large loss $L(\mathbf{\Pi}(\bar{\mathbf{w}}))$. An exception is lossless compression, which satisfies $\mathbf{\Pi} = \mathbf{\Delta}^{-1}$, and here the optimal compressed solution can indeed be achieved by compressing the reference model directly.

4 A “Learning-Compression” (LC) algorithm

Although the constrained problem (1) can be solved with a number of nonconvex optimization algorithms, it is key to profit from parameter separability, which we achieve with penalty methods and alternating optimization, as described next.

Handling the constraints via penalty methods Two classical penalty methods are the quadratic penalty (QP) and the augmented Lagrangian (AL) (Nocedal and Wright, 2006). In the QP, we optimize the following over the parameters (\mathbf{w}, Θ) while driving $\mu \rightarrow \infty$:

$$Q(\mathbf{w}, \Theta; \mu) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{\Delta}(\Theta)\|^2. \quad (6)$$

This has the effect of gradually enforcing the constraints, and the parameters trace a path $(\mathbf{w}(\mu), \Theta(\mu))$ for $\mu > 0$. A better method is the AL. Here we optimize the following over (\mathbf{w}, Θ) while driving $\mu \rightarrow \infty$:

$$\mathcal{L}_A(\mathbf{w}, \Theta, \boldsymbol{\lambda}; \mu) = L(\mathbf{w}) - \boldsymbol{\lambda}^T (\mathbf{w} - \mathbf{\Delta}(\Theta)) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{\Delta}(\Theta)\|^2 \quad (7)$$

$$= L(\mathbf{w}) + \frac{\mu}{2} \left\| \mathbf{w} - \mathbf{\Delta}(\Theta) - \frac{1}{\mu} \boldsymbol{\lambda} \right\|^2 - \frac{1}{2\mu} \|\boldsymbol{\lambda}\|^2 \quad (8)$$

and we update the Lagrange multiplier estimates $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \mu(\mathbf{w} - \mathbf{\Delta}(\Theta))$ after optimizing \mathcal{L}_A over (\mathbf{w}, Θ) for each μ . Optimizing \mathcal{L}_A over (\mathbf{w}, Θ) for fixed $\boldsymbol{\lambda}$ is like optimizing the QP with a shifted parameterization $\mathbf{\Delta}(\Theta) + \frac{1}{\mu} \boldsymbol{\lambda}$, as eq. (8) shows explicitly. The AL is equivalent to the QP if $\boldsymbol{\lambda} = \mathbf{0}$.

Optimizing the penalized function with alternating optimization Applying alternating optimization to the penalized function (quadratic-penalty function Q or augmented Lagrangian \mathcal{L}_A) over \mathbf{w} and Θ (for fixed $\boldsymbol{\lambda}$) gives our “learning-compression” (LC) algorithm. The steps are as follows:

- **L (learning) step:**

$$\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2} \left\| \mathbf{w} - \Delta(\Theta) - \frac{1}{\mu} \boldsymbol{\lambda} \right\|^2. \quad (9)$$

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:**

$$\min_{\Theta} \left\| \mathbf{w} - \frac{1}{\mu} \boldsymbol{\lambda} - \Delta(\Theta) \right\|^2 \iff \Theta = \Pi \left(\mathbf{w} - \frac{1}{\mu} \boldsymbol{\lambda} \right). \quad (10)$$

This means finding the best (lossy) compression of $\mathbf{w} - \frac{1}{\mu} \boldsymbol{\lambda}$ (the current uncompressed model, shifted by $\frac{1}{\mu} \boldsymbol{\lambda}$) in the ℓ_2 sense (orthogonal projection on the feasible set), and corresponds to our definition of the compression mapping Π in section 3.1. *This step is independent of the loss, training set and task.*

4.1 Practicalities

Fig. 2 gives the LC algorithm pseudocode for the augmented Lagrangian (using a single LC iteration per $\boldsymbol{\lambda}$ update). For the quadratic-penalty version, ignore or set to zero $\boldsymbol{\lambda}$ everywhere.

Reusing existing code in the L and C steps The L step gradually pulls \mathbf{w} towards a model that can be obtained by decompressing some $\Theta \in \mathbb{R}^Q$, and the C step compresses the current \mathbf{w} . Both of these steps can be done by reusing existing code rather than writing a new algorithm, which makes the LC algorithm easy to implement. The L step just needs an additive term “ $\mu(\mathbf{w} - \Delta(\Theta) - \frac{1}{\mu} \boldsymbol{\lambda})$ ” to the gradient of $L(\mathbf{w})$, e.g. in stochastic gradient descent (SGD) for neural nets. The C step depends on the compression type, but will generally correspond to a well-known compression algorithm (e.g. SVD for low-rank compression, k -means for quantization). Different types of compression can be used by simply calling a different compression routine in the C step, with no other change to the LC algorithm. This facilitates the model designer’s job of trying different types of compression to find the most suitable for the task at hand.

Runtime 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), although this depends on the type of compression and the loss. Hence, it pays to do the C step as exactly as possible. 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.

Schedule of the penalty parameter μ_k In practice, as usual with penalty methods, we use a multiplicative μ schedule: $\mu_k = a^k \mu_0$ with a slightly larger than 1 and $\mu_0 \approx 0$ (set by trial and error); see also section 5. As noted in the pseudocode, we run a single L and C step per μ value because this keeps the

input training data and model with parameters (weights) \mathbf{w}	
$\bar{\mathbf{w}} \leftarrow \arg \min_{\mathbf{w}} L(\mathbf{w})$	reference model
$\Theta \leftarrow \Theta^{\text{DC}} = \Pi(\bar{\mathbf{w}}) = \arg \min_{\Theta} \ \bar{\mathbf{w}} - \Delta(\Theta)\ ^2$	compress reference model
$\boldsymbol{\lambda} \leftarrow \mathbf{0}$	
for $\mu = \mu_0 < \mu_1 < \dots < \infty$	
$\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2} \ \mathbf{w} - \Delta(\Theta) - \frac{1}{\mu} \boldsymbol{\lambda}\ ^2$	L step: learn model
$\Theta \leftarrow \Pi(\mathbf{w} - \frac{1}{\mu} \boldsymbol{\lambda}) = \arg \min_{\Theta} \ \mathbf{w} - \frac{1}{\mu} \boldsymbol{\lambda} - \Delta(\Theta)\ ^2$	C step: compress model
$\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \mu(\mathbf{w} - \Delta(\Theta))$	Lagrange multipliers
if $\ \mathbf{w} - \Delta(\Theta)\ $ is small enough then exit the loop	
return \mathbf{w}, Θ	

Figure 2: Pseudocode for the LC algorithm, augmented-Lagrangian version.

algorithm simple (we avoid an extra loop). However, in some cases it may be advantageous to run multiple L and C steps per μ value, e.g. if it is possible to cache matrix factorizations in order to speed up the L or C step.

Initialization and termination We always initialize $\lambda = \mathbf{0}$ and $(\mathbf{w}, \Theta) = (\bar{\mathbf{w}}, \Theta^{\text{DC}})$, i.e., to the reference model and direct compression, which is the exact solution for $\mu \rightarrow 0^+$, as we show in the next section. We stop the LC algorithm when $\|\mathbf{w} - \Delta(\Theta)\|$ is smaller than a set tolerance, which will happen when μ is large enough. At this point, the final iterate (\mathbf{w}, Θ) satisfies $\Theta = \Pi(\mathbf{w} - \frac{1}{\mu}\lambda) = \arg \min_{\Theta} \|\mathbf{w} - \frac{1}{\mu}\lambda - \Delta(\Theta)\|^2$, so that $\Delta(\Theta)$ is a compressed model (hence a feasible weight vector), while \mathbf{w} is not (although it will be very close to $\Delta(\Theta)$). Hence, the solution (feasible and (near-)optimal) is $\Delta(\Theta)$.

Other comments In the derivation of the LC algorithm we used a quadratic penalty to penalize violations of the equality constraint $\mathbf{w} = \Delta(\Theta)$. This is convenient because it makes the L step easy with gradient-based optimization (it behaves like a form of weight decay on the loss), and the C step is also easy for some compression forms (quantization can be done by k -means, low-rank approximation by the SVD). However, non-quadratic penalty forms $P(\mathbf{w}, \Delta(\Theta))$ may be convenient in other situations.

Note that the C step can also be seen as trying to predict the weights \mathbf{w} from the low-dimensional parameters Θ via the mapping Δ . In this sense, compression is a machine learning problem of modeling “data” (the P weights) using a low-dimensional space (the Q parameters). This was noted by Denil et al. (2013) in the context of low-rank models, but it applies generally in our framework. See also our discussion of parametric embeddings in section 6.3. In fact, model fitting itself in machine learning can be seen as compressing the training set into the model parameters.

4.2 Direct compression (DC) and the beginning of the path

In the LC algorithm, the parameters trace a path $(\mathbf{w}(\mu), \Theta(\mu))$ for $\mu \geq 0$, and the solution is obtained for $\mu \rightarrow \infty$, when the constraints are satisfied and we achieve an optimal compressed model. The beginning of the path, for $\mu \rightarrow 0^+$, has a special meaning: it corresponds to the *direct compression* (training the reference model and then compressing its weights), as we show next. (We write $\mu \rightarrow 0^+$ rather than $\mu = 0$ because the latter defines a problem without Θ .)

Taking $\mu \rightarrow 0^+$ in (6) or (7) and assuming an initial $\lambda = \mathbf{0}$, we obtain

$$\mathbf{w}(0^+) = \arg \min_{\mathbf{w}} L(\mathbf{w}) \equiv \bar{\mathbf{w}}, \quad (11)$$

since the μ term is negligible, and

$$\Theta(0^+) = \Pi(\bar{\mathbf{w}}) = \arg \min_{\Theta} \|\bar{\mathbf{w}} - \Delta(\Theta)\|^2 \equiv \Theta^{\text{DC}}, \quad (12)$$

i.e., the orthogonal projection of $\bar{\mathbf{w}}$ on the feasible set (up to local optima in both \mathbf{w} and Θ), recalling the discussion of section 3.1. Hence, the path starts at $(\mathbf{w}(0^+), \Theta(0^+)) = (\bar{\mathbf{w}}, \Theta^{\text{DC}})$, which corresponds to the *direct compression*: training the large, reference model and then compressing its weights (note that in DC we discard $\bar{\mathbf{w}}$ and keep only $\mathbf{w}^{\text{DC}} \equiv \Delta(\Theta^{\text{DC}})$, i.e., the compressed model). This is not optimal in the sense of problem (1) because the compression ignores the learning task; the best compression of the weights need not be the best compressed model for the task.

The constrained optimization view shows that, if an optimal uncompressed model $\bar{\mathbf{w}}$ is feasible, i.e., there is a $\Theta \in \mathbb{R}^Q$ with $\Delta(\Theta) = \bar{\mathbf{w}}$, then it is optimal compressed, since the compression has zero error, and in this case $\bar{\mathbf{w}} = \mathbf{w}^{\text{DC}} = \mathbf{w}^*$ (and there is no need to optimize with the LC algorithm). But, generally, compression will increase the loss, the more so the larger the compression level (so the smaller the feasible set and the larger the distance $\|\bar{\mathbf{w}} - \Delta(\Theta^{\text{DC}})\|$ to the DC model). Therefore, we should expect that, with low compression levels, direct compression will be near-optimal, but as we compress more—which is our goal, and critical in actual deployment in mobile devices—it will become worse and worse in loss wrt the optimal compressed model \mathbf{w}^* . Hence, high compression rates require the better LC optimization. Plot 3 in figure 1 illustrates this. Indeed, the suboptimality of direct compression compared to the result of the LC algorithm becomes practically evident in experiments compressing neural nets as we push the compression

level (Carreira-Perpiñán and Idelbayev, 2017a,b). In section 6.2, we discuss existing work related to direct compression.

5 Convergence results for the LC algorithm

The quadratic penalty and augmented Lagrangian methods belong to the family of homotopy (path-following) algorithms, where the minima $(\mathbf{w}(\mu), \Theta(\mu))$ of $Q(\mathbf{w}, \Theta; \mu)$ or $\mathcal{L}_A(\mathbf{w}, \Theta, \lambda; \mu)$ define a path for $\mu \geq 0$ and the solution we want is at $\mu \rightarrow \infty$. We give a theorem based on the QP; similar results are possible for the AL. Assume the loss $L(\mathbf{w})$ and the decompression mapping $\Delta(\Theta)$ are continuously differentiable wrt their arguments, and that the loss is lower bounded.

Theorem 5.1. *Consider the constrained problem (1) and its quadratic-penalty function $Q(\mathbf{w}, \Theta; \mu)$ of (6). Given a positive increasing sequence $(\mu_k) \rightarrow \infty$, a nonnegative sequence $(\tau_k) \rightarrow 0$, and a starting point (\mathbf{w}^0, Θ^0) , suppose the QP method finds an approximate minimizer (\mathbf{w}^k, Θ^k) of $Q(\mathbf{w}^k, \Theta^k; \mu_k)$ that satisfies $\|\nabla_{\mathbf{w}, \Theta} Q(\mathbf{w}^k, \Theta^k; \mu_k)\| \leq \tau_k$ for $k = 1, 2, \dots$. Then, $\lim_{k \rightarrow \infty} ((\mathbf{w}^k, \Theta^k)) = (\mathbf{w}^*, \Theta^*)$, which is a KKT point for the problem (1), and its Lagrange multiplier vector has elements $\lambda_i^* = \lim_{k \rightarrow \infty} (-\mu_k (\mathbf{w}_i^k - \Delta(\Theta_i^k)))$, $i = 1, \dots, P$.*

Proof. It follows by applying theorem 17.2 in Nocedal and Wright (2006), quoted in appendix A, to the constrained problem (1) and by noting that 1) $\lim_{k \rightarrow \infty} ((\mathbf{w}^k, \Theta^k)) = (\mathbf{w}^*, \Theta^*)$ exists and 2) that the constraint gradients are linearly independent. We prove these two statements in turn. First, the limit of the sequence $((\mathbf{w}^k, \Theta^k))$ exists because the loss and hence the QP function $Q(\mathbf{w}, \Theta; \mu)$ are lower bounded and have continuous derivatives. Second, the constraint gradients are linearly independent at any point (\mathbf{w}, Θ) and thus, in particular, at the limit (\mathbf{w}^*, Θ^*) . To see this, note that the Jacobian of the constraint function $\mathbf{w} - \Delta(\Theta) = \mathbf{0}$ wrt (\mathbf{w}, Θ) is the $P \times (P + Q)$ matrix $(\mathbf{I}_P \quad \nabla_{\Theta} \Delta(\Theta))$, whose rank is obviously P , and so is full-rank. \square

Stated otherwise, the LC algorithm defines a continuous path $(\mathbf{w}(\mu), \Theta(\mu))$ which, under some mild assumptions (essentially, that we minimize $Q(\mathbf{w}, \Theta; \mu)$ increasingly accurately as $\mu \rightarrow \infty$), converges to a stationary point (typically a minimizer) of the constrained problem (1). With convex problems, there is a unique path leading to the solution. With nonconvex problems, there are multiple paths, each leading to a local optimum. As with any nonconvex continuous optimization problem, convergence may occur in pathological cases to a stationary point of the constrained problem that is not a minimizer, but such cases should be rare in practice.

Computationally, it is better to approach the solution by following the path from small μ , because Q (or \mathcal{L}_A) become progressively ill-conditioned as $\mu \rightarrow \infty$. While ideally we would follow the path closely, by increasing μ slowly from 0 to ∞ , in practice we follow the path loosely to reduce the runtime, typically using a multiplicative schedule for the penalty parameter, $\mu_k = \mu_0 a^k$ for $k = 0, 1, 2 \dots$ where $\mu_0 > 0$ and $a > 1$. If, after the first iteration, the iterates get stuck at the direct compression value, this is usually a sign that we increased μ too fast. The smaller μ_0 and a , the more closely we follow the path.

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. However, it does not apply to some popular compression forms, specifically quantization and pruning, which generally give rise to NP-complete problems. Indeed, consider one of the simplest models: least-squares linear regression, which defines a quadratic loss over the weights, whose solution is given by a linear system. Forcing the weights to be either -1 or $+1$ (quantization by binarization) defines a binary quadratic problem over the weights, which is NP-complete (Garey and Johnson, 1979). Forcing a given proportion of the weights to be zero (pruning) is an ℓ_0 -constrained problem, also NP-complete (Natarajan, 1995). While we cannot expect the LC algorithm to find the global optimum of these problems, we can expect reasonably good results in the following sense. 1) The LC algorithm is still guaranteed to converge to a weight vector \mathbf{w} that satisfies the constraints (having elements in $\{-1, +1\}$ or having the given proportion of elements be zero, in those examples), hence it will always converge to a validly compressed model. 2) Because the L step minimizes (partially) the loss, the convergence will likely be to a low-loss point (even if not necessarily optimal).

5.1 Choice of learning rate in the L step with large-scale optimization²

Theorem 5.1 states that for convergence to occur, the L and C steps must be solved increasingly accurately. This is generally not a problem for the C step, as it is usually solved by an existing compression algorithm. The L step needs some consideration. The objective function over \mathbf{w} in the L step has the form of the original loss L plus a very simple term, a separable quadratic function:

$$\min_{\mathbf{w}} Q(\mathbf{w}) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}'\|^2 \quad (13)$$

where $\mathbf{w}' = \Delta(\Theta)$ for the QP (6) and $\mathbf{w}' = \Delta(\Theta) - \frac{1}{\mu}\lambda$ for the AL (7). Intuitively, optimizing (13) should not be very different from optimizing the loss (which we had to do in order to obtain the reference model). Indeed, gradient-based optimization is straightforward, since the gradient of (13) simply adds $\mu(\mathbf{w} - \mathbf{w}')$ to the gradient of the loss. Many optimization algorithms can be used to solve this depending on the form of L , such as a modified Newton method with line searches or even solving a linear system if L is quadratic. However, for large-scale problems (large datasets and/or large dimension of \mathbf{w}) we are particularly interested in gradient-based optimization without line searches, such as gradient descent with a fixed step size, or stochastic gradient descent (SGD) with learning rates (step sizes) satisfying a Robbins-Monro schedule. Convergence without line searches requires certain conditions on the step size, and these must be corrected to account for the fact that the quadratic μ -term increases as μ increases, since then the gradient term $\mu(\mathbf{w} - \mathbf{w}')$ also increases, and this can cause problems (such as large, undesirable jumps in early epochs of each new L step if using SGD). We consider two cases of optimizing the L step: using gradient descent with a fixed step size, and using SGD.

5.1.1 Optimization of a convex loss using gradient descent with a fixed step size

As is well known from convex optimization arguments (see proofs in appendix B.1), if the loss $L(\mathbf{w})$ is convex differentiable with Lipschitz continuous gradient with Lipschitz constant $M > 0$, then training the reference model can be done by gradient descent with a fixed step size $\frac{1}{M}$. In the L step, the objective function (13) is strictly convex and therefore it has a unique minimizer. Gradient descent with a fixed step size $\frac{1}{M+\mu}$, $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{M+\mu}(\nabla L(\mathbf{w}_t) + \mu(\mathbf{w}_t - \mathbf{w}'))$, converges to the minimizer linearly with rate $\frac{M}{M+\mu} \in (0, 1)$ from any initial point. Hence, we simply need to adjust the step size from $\frac{1}{M}$ in the reference model to $\frac{1}{M+\mu}$ in the L step. Although the step size becomes smaller as μ increases, the convergence becomes faster. The reason is that the objective function becomes closer to a separable quadratic function, whose optimization is easier.

5.1.2 Optimization using stochastic gradient descent (SGD)

With neural nets, which involve a nonconvex loss and a large dataset, SGD is the usual optimization procedure. The loss takes the form $L(\mathbf{w}) = \sum_{n=1}^N L_n(\mathbf{w})$ where L_n is the loss for the n th training point and N is large, so it is too costly to evaluate the gradient exactly. It is practically preferable to take approximate gradient steps based on evaluating the gradient at step t on a minibatch $\mathcal{B}_t \subset \{1, \dots, N\}$, hence each step can be seen as a gradient step using a noisy gradient:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \nabla_{\mathcal{B}_t} L(\mathbf{w}_t) \quad \text{where} \quad \nabla_{\mathcal{B}_t} L(\mathbf{w}_t) = \sum_{n \in \mathcal{B}_t} \nabla L_n(\mathbf{w}_t) = \nabla L(\mathbf{w}_t) + \text{noise}_t. \quad (14)$$

The convergence theorems for this stochastic setting are very different from those assuming exact gradients, most notably in the requirement that the step sizes (learning rates) η_t must tend to zero, at a certain speed, as $t \rightarrow \infty$, which we call a *Robbins-Monro schedule*:

$$\text{Robbins-Monro schedule } \{\eta_t\}_{t=0}^{\infty}: \quad \eta_t > 0 \quad \forall t = 0, 1, 2, \dots, \quad \sum_{t=0}^{\infty} \eta_t = \infty, \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty. \quad (15)$$

²In this section, we use the subindex t to indicate iterates, such as \mathbf{w}_t , *within the L step*. These are different from the iterates of the LC algorithm, denoted as (\mathbf{w}^k, Θ^k) in theorem 5.1.

Appendix B.2 gives detailed theorems with sufficient conditions for convergence (to a stationary point of the loss) in the case where the noise is deterministic (theorem B.7), in particular for the incremental gradient algorithm (theorem B.8), and when the noise is stochastic (theorem B.9). These conditions include Lipschitz continuity of the loss gradient, a condition on the noise to be small enough, and that the learning rates satisfy a Robbins-Monro schedule. The convergence rate is sublinear and much slower than with exact gradients (discussed in the previous section). In practice, the schedule typically has the form $\eta_t = \frac{\alpha}{\beta+t}$ where $\alpha, \beta > 0$ are determined by trial-and-error on a subset of the data. Unfortunately, the convergence theory for SGD is not very practical: apart from the conditions on the loss and the noise (which can be hard to verify but are mild), all the theory tells us is that using a Robbins-Monro schedule will lead to convergence. However, the performance of SGD is very sensitive to the schedule and selecting it well is of great importance. Indeed, SGD learning of neural nets is notoriously tricky and considerable trial and error in setting its hyperparameters (learning rate, momentum rate, minibatch size, etc.) is unavoidable in practice.

Consider now the $Q(\mathbf{w})$ objective function (13) of the L step. The SGD updates take the form:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \left(\nabla_{\mathcal{B}_t} L(\mathbf{w}_t) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}'\|^2 \right) \text{ where } \nabla_{\mathcal{B}_t} L(\mathbf{w}_t) = \sum_{n \in \mathcal{B}_t} \nabla L_n(\mathbf{w}_t) = \nabla L(\mathbf{w}_t) + \text{noise}_t. \quad (16)$$

Our concern is, given a good Robbins-Monro schedule $\{\eta_t\}_{t=0}^\infty$ for the reference model (i.e., for optimizing $L(\mathbf{w})$ alone), should the schedule be modified for optimizing Q , and if so how? In theory, no change is needed, because the only condition that the convergence theorems require of the schedule is that it be Robbins-Monro. In practice, this requires some consideration. The addition of the μ -term to the loss has two effects. First, since its exact gradient $\mu(\mathbf{w} - \mathbf{w}')$ is fast to compute, the noise in the gradient of Q will (usually) be smaller, which is a good thing for convergence purposes. Second, and this is the practical problem, since μ increases towards infinity its gradient becomes progressively large³. Hence, the *early* weight updates in the L step (which use a larger learning rate) may considerably perturb \mathbf{w} , sending it away from the initial \mathbf{w} (provided by warm-start from the previous iteration's C step). While convergence to some minimizer (or stationary point in general) of Q is still guaranteed with a Robbins-Monro schedule, this may be much slower and occur to a different local minimizer. This makes the overall optimization unstable and should be avoided.

We can solve these problems by using a schedule $\{\eta'_t\}_{t=0}^\infty$ that both satisfies the Robbins-Monro conditions and would lead to convergence of the μ -term alone:

$$\text{Clipped schedule } \{\eta'_t\}_{t=0}^\infty: \quad \eta'_t = \min \left(\eta_t, \frac{1}{\mu} \right), \quad t = 0, 1, 2 \dots \quad (17)$$

That is, the first iterations will use a learning rate $\frac{1}{\mu}$, and then switch to the original η_t schedule (when $\eta_t \leq \frac{1}{\mu}$). The justification is provided by the following two theorems.

Theorem 5.2. *Consider minimizing a function $f(\mathbf{w})$ using gradient descent with a fixed step size $\eta > 0$, i.e., we iterate $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \nabla f(\mathbf{w}_t)$ for $t = 0, 1, 2 \dots$ and some \mathbf{w}_0 . If $f(\mathbf{w}) = \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}'\|^2$, then that sequence converges linearly to the minimizer \mathbf{w}' iff $\eta \in (0, \frac{2}{\mu})$. Also, $\eta = \frac{1}{\mu}$ converges to the minimizer in one iteration.*

Proof. A gradient descent step with step size $\eta > 0$ is $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \mu (\mathbf{w}_t - \mathbf{w}') = (1 - \eta \mu) \mathbf{w}_t + \eta \mu \mathbf{w}'$. So $\eta = \frac{1}{\mu}$ yields $\mathbf{w}_{t+1} = \mathbf{w}'$. And $\mathbf{w}_{t+1} - \mathbf{w}' = (1 - \eta \mu)(\mathbf{w}_t - \mathbf{w}') = (1 - \eta \mu)_{t+1}(\mathbf{w}_0 - \mathbf{w}')$, which converges linearly to zero if $|1 - \eta \mu| < 1 \Leftrightarrow \eta \in (0, \frac{2}{\mu})$. \square

Theorem 5.3. *Consider a learning rate schedule $\{\eta_t\}_{t=0}^\infty$ that satisfies the Robbins-Monro conditions (15) and let $\mu > 0$. Then the schedule $\{\eta'_t\}_{t=0}^\infty$ with $\eta'_t = \min(\eta_t, \frac{1}{\mu})$, $t = 0, 1, 2 \dots$ also satisfies (15).*

Proof. We have by assumption that a) $\eta_t > 0 \forall t$, b) $\sum_{t=0}^\infty \eta_t = \infty$ and c) $\sum_{t=0}^\infty \eta_t^2 < \infty$. Obviously, $\eta'_t > 0 \forall t$. From c) we have that $\lim_{t \rightarrow \infty} \eta_t = 0$, so there exists a $T \geq 0$ such that $\eta_t < \frac{1}{\mu} \forall t \geq T$, and $\sum_{t=T}^\infty \eta_t = \infty$. Hence $\sum_{t=T}^\infty \eta'_t = \sum_{t=T}^\infty \eta_t = \infty$, and $\sum_{t=0}^\infty \eta'_t = \infty$. Finally, since $0 < \eta'_t \leq \eta_t \forall t$, i.e., the sequence (η'_t) is majorized by the sequence (η_t) , then $\sum_{t=0}^\infty (\eta'_t)^2 \leq \sum_{t=0}^\infty \eta_t^2 < \infty$. \square

³This makes the situation different from weight decay, which uses a similar objective function, of the form $L(\mathbf{w}) + \lambda \|\mathbf{w}\|^2$, but where λ is fixed and usually very small.

Theorem 5.2 tells us we should use learning rates below $\frac{2}{\mu}$ and suggests using $\frac{1}{\mu}$ (particularly as μ increases, since then the μ -term becomes dominant). Theorem 5.3 guarantees that clipping a Robbins-Monro schedule remains Robbins-Monro. Hence, the clipped schedule makes sure that the initial, larger updates do not exceed $\frac{1}{\mu}$ (the optimal step size for the μ -term), and otherwise leaves it unchanged. This then ensures that the first steps do not unduly perturb the initial \mathbf{w} , while convergence to a minimum of $Q(\mathbf{w})$ is guaranteed since the schedule is Robbins-Monro and Q has Lipschitz continuous gradient if L does (as long as the noise condition in the convergence theorems holds).

In a nutshell, our practical recommendation is as follows: first, we determine by trial and error a good schedule for the reference model (i.e., which drives the weight vector \mathbf{w} to close to a local minimizer $\bar{\mathbf{w}}$ of the loss L as fast as possible). Then, we use the clipped schedule in the L step for $\mu > 0$. We have done this in experiments on various compression forms (Carreira-Perpiñán and Idelbayev, 2017a,b) and found it effective.

6 Relation of the LC algorithm with other algorithms

6.1 One algorithm, many compression types

We emphasize that the specific form of our LC algorithm follows necessarily from the definition of the compression technique in the constraints of problem (1). Some work on neural net compression is based on modifying the usual training procedure (typically backpropagation with SGD) by manipulating the weights on the fly in some ad-hoc way, such as binarizing or pruning them, but this usually has no guarantee that it will solve problem (1), or converge at all. In our framework, the LC algorithm (specifically, the C step) follows necessarily from the constraints that define the form of compression in (1). For example, for quantization (Carreira-Perpiñán and Idelbayev, 2017a) and low-rank compression the C step results in k -means and the SVD, respectively, because the C step optimization “ $\min_{\Theta} \|\mathbf{w} - \Delta(\Theta)\|^2$ ” takes the form of a quadratic distortion problem in both cases. For pruning using the ℓ_0 norm (Carreira-Perpiñán and Idelbayev, 2017b), the optimization in the C step results in a form of weight magnitude thresholding. There is no need for any ad-hoc decisions in the algorithm.

6.2 Direct compression and retraining approaches

Our formulation of what an optimal solution to the model compression problem is and the form of the LC algorithm allows us to put some earlier work into context.

6.2.1 Direct compression approaches

Direct compression (DC) consists of training the reference model and then compressing its weights. As shown in section 4.2, DC corresponds to the beginning of the iterates’ path in the LC algorithm, and is suboptimal, that is, it does not produce the compressed model with lowest loss. That said, direct compression is an appealing approach: it is an obvious thing to do, it is simple, and it is fast, because it does not require further training of a reference model (and hence no further access to the training set). Indeed, particular instances of DC corresponding to particular compression techniques have been recently applied to compress neural nets (although presumably much earlier attempts exist in the literature). These include quantizing the weights of the reference net with k -means (Gong et al., 2015), pruning the weights of the reference net by zeroing small values (Reed, 1993) or reducing the rank of the weight matrices of the reference net using the SVD (Sainath et al., 2013; Jaderberg et al., 2014; Denton et al., 2014). However, the LC algorithm is nearly as simple as direct compression: it can be seen as iterating the direct compression but with a crucial link between the L and C steps, the $\frac{\mu}{2} \|\mathbf{w} - \Delta(\Theta)\|^2$ term. Practically, this is not much slower, given that we have to train the reference model anyway. Since the C steps are usually negligible compared to the L steps, the LC algorithm behaves like training the reference model for a longer time.

6.2.2 Retraining after direct compression

As we mentioned in section 4.2, the result of direct compression can have a large loss, particularly for large compression levels. A way to correct for this partially is to retrain the compressed model, and this has

been a standard approach with neural net pruning (Reed, 1993). Here, we first train the reference net and then prune its weights in some way, e.g. by thresholding small-magnitude weights. This gives the direct compression model if using sparsity-inducing norms (see Carreira-Perpiñán and Idelbayev, 2017b). Finally, we optimize the loss L again but only over the remaining, unpruned weights. This reduces the loss, often considerably. However, it loses the advantage of DC of not having to retrain the net (which requires access to the training set), and it is still suboptimal, since generally the set of weights that were pruned is not the set that would give the lowest loss. The LC algorithm consistently beats retraining for pruning, particularly for higher compression levels (see Carreira-Perpiñán and Idelbayev, 2017b).

6.2.3 Iterated direct compression (iDC)

Imagine we iterate the direct compression procedure. That is, we optimize $L(\mathbf{w})$ to obtain $\bar{\mathbf{w}}$ and then compress it into Θ^{DC} . Next, we optimize $L(\mathbf{w})$ again but initializing \mathbf{w} from $\Delta(\Theta^{\text{DC}})$, and then we compress it; etc. Our argument in section 4.2 implies that nothing would change after the first DC and we would simply cycle between $\bar{\mathbf{w}}$ and $\Delta(\Theta^{\text{DC}})$ forever. In fact, several DC iterations may be needed for this to happen, for two reasons. 1) With local optima of $L(\mathbf{w})$, we might converge to a different optimum after the compression (see fig. 1 plot 2). However, sooner rather than later this will end up cycling between a local optimum of $L(\mathbf{w})$ and its compressed model. Still, this improves over the DC optimum. 2) A more likely reason in practice are inexact compression or learning steps. This implies the iterates never fully reach either $\bar{\mathbf{w}}$ or $\Delta(\Theta^{\text{DC}})$ or both, and keep oscillating forever in between. This is particularly the case if training neural nets with stochastic gradient descent (SGD), for which converging to high accuracy requires long training times.

We call the above procedure “iterated direct compression (iDC)”. A version of this for quantization has been proposed recently (“trained quantization”, Han et al., 2015), although without the context that our constrained optimization framework provides. In our experiments elsewhere (Carreira-Perpiñán and Idelbayev, 2017a), we verify that neither DC nor iDC converge to a local optimum of problem (1), while our LC algorithm does.

6.3 Other algorithms beyond model compression

6.3.1 The method of auxiliary coordinates (MAC)

Overall, we derive our LC algorithm by applying the following design pattern to solve the compression problem: 1) introducing auxiliary variables \mathbf{w} in eq. (1), 2) handling the constraints via penalty methods (QP or AL) in eqs. (6)–(7), and 3) optimizing the penalized function using alternating optimization over the original variables Θ and the auxiliary variables \mathbf{w} in eqs. (9)–(10). This design pattern is similar to that used in the *method of auxiliary coordinates (MAC)* for optimizing nested systems such as deep nets (Carreira-Perpiñán and Wang, 2012, 2014), i.e., involving functions of the form $\mathbf{f}(\mathbf{x}; \mathbf{W}) = \mathbf{f}_{K+1}(\dots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x}; \mathbf{W}_1); \mathbf{W}_2) \dots; \mathbf{W}_{K+1})$, where \mathbf{x} is an input data point and \mathbf{W} are trainable parameters (e.g. weights in a deep net). Here, one introduces auxiliary coordinates per data point \mathbf{x}_n of the form $\mathbf{z}_{k,n} = \mathbf{f}_k(\mathbf{z}_{k-1,n}; \mathbf{W}_k)$ (where $\mathbf{z}_{0,n} \equiv \mathbf{x}_n$), for $k = 1, \dots, K$ and $n = 1, \dots, N$. Then, handling these constraints with a penalty method and applying alternating optimization yields the final algorithm. This alternates a “maximization” step that optimizes single-layer functions independently of each other (over the \mathbf{W}_k) with a “coordination” step that optimizes over the auxiliary coordinates (the $\mathbf{z}_{k,n}$) independently for each data point. Hence, in MAC the auxiliary coordinates are per data point and capture intermediate function values within the nested function, while in our LC algorithm the auxiliary variables duplicate the parameters of the model in order to separate learning from compression.

6.3.2 Parametric embeddings

MAC and the LC algorithm become identical in one interesting case: *parametric embeddings*. A *nonlinear embedding* algorithm seeks to map a collection of high-dimensional data points $\mathbf{Y} = (\mathbf{y}_1, \dots, \mathbf{y}_N)$ of $D \times N$ to a collection of low-dimensional projections $\mathbf{X} = (\mathbf{x}_1, \dots, \mathbf{x}_N)$ of $L \times N$ with $L < D$ such that distances or similarities between pairs of points $(\mathbf{y}_n, \mathbf{y}_m)$ are approximately preserved between the corresponding pairs of projections $(\mathbf{x}_n, \mathbf{x}_m)$. Examples of nonlinear embeddings are spectral methods such as

multidimensional scaling (Borg and Groenen, 2005) or Laplacian eigenmaps (Belkin and Niyogi, 2003), and truly nonlinear methods such as stochastic neighbor embedding (SNE) (Hinton and Roweis, 2003), t -SNE (van der Maaten and Hinton, 2008) or the elastic embedding (Carreira-Perpiñán, 2010). For example, the elastic embedding optimizes:

$$E(\mathbf{X}) = \sum_{n,m=1}^N a_{nm} \|\mathbf{x}_n - \mathbf{x}_m\|^2 + \lambda \sum_{n,m=1}^N \exp(-\|\mathbf{x}_n - \mathbf{x}_m\|^2) \quad (18)$$

where a_{nm} defines the similarity between \mathbf{y}_n and \mathbf{y}_m (the more positive a_{nm} is, the more similar \mathbf{y}_n and \mathbf{y}_m are). Therefore, the first term attracts similar points, the second term repels all points, and the optimal embedding \mathbf{X} balances both forces (depending on the tradeoff parameter $\lambda > 0$). In a parametric embedding, we wish to learn a projection mapping \mathbf{F} : $\mathbf{y} \in \mathbb{R}^D \rightarrow \mathbf{x} \in \mathbb{R}^L$ rather than the projections $\mathbf{y}_1, \dots, \mathbf{y}_N$ themselves (so we can use \mathbf{F} to project a new point \mathbf{y} as $\mathbf{F}(\mathbf{y})$). For the elastic embedding this means optimizing the following (where \mathbf{F} is a parametric mapping with parameters Θ , e.g. a linear mapping or a neural net):

$$P(\Theta) = \sum_{n,m=1}^N a_{nm} \|\mathbf{F}(\mathbf{y}_n; \Theta) - \mathbf{F}(\mathbf{y}_m; \Theta)\|^2 + \lambda \sum_{n,m=1}^N \exp(-\|\mathbf{F}(\mathbf{y}_n; \Theta) - \mathbf{F}(\mathbf{y}_m; \Theta)\|^2). \quad (19)$$

To optimize this using MAC (Carreira-Perpiñán and Vladymyrov, 2015), we recognize the above as a nested mapping and introduce auxiliary coordinates $\mathbf{z}_n = \mathbf{F}(\mathbf{y}_n; \Theta)$ for $n = 1, \dots, N$. The QP function is

$$Q(\mathbf{Z}, \Theta; \mu) = E(\mathbf{Z}) + \frac{\mu}{2} \sum_{n=1}^N \|\mathbf{z}_n - \mathbf{F}(\mathbf{y}_n; \Theta)\|^2 = E(\mathbf{Z}) + \frac{\mu}{2} \|\mathbf{Z} - \mathbf{F}(\mathbf{Y}; \Theta)\|^2 \quad (20)$$

and alternating optimization yields the following two steps:

- Over \mathbf{Z} , it has the form of a nonlinear embedding with a quadratic regularization:

$$\min_{\mathbf{Z}} E(\mathbf{Z}) + \frac{\mu}{2} \|\mathbf{Z} - \mathbf{F}(\mathbf{Y}; \Theta)\|^2. \quad (21)$$

- Over Θ , it has the form of a regression problem with inputs \mathbf{Y} and outputs \mathbf{Z}

$$\min_{\Theta} \sum_{n=1}^N \|\mathbf{z}_n - \mathbf{F}(\mathbf{y}_n; \Theta)\|^2. \quad (22)$$

We can see this as an LC algorithm for model compression if we regard \mathbf{Z} as the uncompressed model (so $\bar{\mathbf{Z}} = \arg \min_{\mathbf{Z}} E(\mathbf{Z})$ is the reference model) and Θ (or equivalently the projection mapping \mathbf{F}) as the compressed model. MAC and the LC algorithm coincide because in a parametric embedding each data point (\mathbf{y}_n) is associated with one parameter vector (\mathbf{z}_n). The decompression mapping is $\mathbf{Z} = \Delta(\Theta) = \mathbf{F}(\mathbf{Y}; \Theta)$, which (approximately) recovers the uncompressed model by applying the projection mapping to the high-dimensional dataset. The compression step finds optimally the parameters Θ of \mathbf{F} via a regression fit. The learning step learns the regularized embedding \mathbf{Z} . “Direct compression” (called “direct fit” in Carreira-Perpiñán and Vladymyrov, 2015) fits \mathbf{F} directly to the reference embedding $\bar{\mathbf{Z}}$, which is suboptimal, and corresponds to the beginning of the path in the MAC or LC algorithm. Hence, in this view, *parametric embeddings can be seen as compressed nonlinear embeddings*.

7 Compression, generalization and model selection

In this paper we focus exclusively on compression as a mechanism to find a model having minimal loss and belonging to a set of compressed models, as precisely formulated in problem (1). However, generalization is an important aspect of compression, and we briefly discuss this.

Compression can also be seen as a way to prevent overfitting, since it aims at obtaining a smaller model with a similar loss to that of a well-trained reference model. This was noted early in the literature of neural nets, in particular pruning weights or neurons was seen as a way to explore different network architectures (see Reed, 1993 and Bishop, 1995, ch. 9.5). Soft weight-sharing (Nowlan and Hinton, 1992), a form of soft quantization of the weights of a neural net, was proposed as a regularizer to make a network generalize better. More recently, weight binarization schemes have also been seen as regularizers (Courbariaux et al., 2015).

Many recent papers, including our own work (Carreira-Perpiñán and Idelbayev, 2017a,b), report experimentally that the training and/or test error of compressed models is lower than that of the reference (as long as the compression level is not too large). Some papers interpret this as an improvement in the generalization ability of the compressed net. While this is to some extent true, there is a simpler reason for this (which we note in section 6.2.3) that surely accounts for part of this error reduction: the reference model was not trained well enough, so that the continued training that happens while compressing reduces the error. This will generally be unavoidable in practice with large neural nets, because the difficulty in training them accurately will mean the reference model is close to being optimal, but never exactly optimal.

Model selection consists of determining the model type and size that achieves best generalization for a given task. It is a difficult problem in general, but much more so with neural nets because of the many factors that determine their architecture: number of layers, number of hidden units, type of activation function (linear, sigmoidal, rectified linear, maxpool...), type of connectivity (dense, convolutional...), etc. This results in an exponential number of possible architectures. *Compression can be seen as a shortcut to model selection*, as follows. Instead of finding an approximately optimal architecture for the problem at hand by an expensive trial-and-error search, one can train a reference net that *overestimates* the necessary size of the architecture (with some care to control overfitting). This gives a good estimate of the best performance that can be achieved in the problem. Then, one compresses this reference using a suitable compression scheme and a desired compression level, say pruning $p\%$ of the weights or quantizing the weights using $\log_2 K$ bits. Then, what our LC algorithm does is automatically search a subset of models of a given size (corresponding to the compression level). For example, the ℓ_0 -based pruning mechanism of Carreira-Perpiñán and Idelbayev (2017b) uses a single κ parameter (the total number of nonzero weights in the entire net) but implicitly considers all possible pruning levels for each layer of the net. This is much easier on the network designer than having to test multiple combinations of the number of hidden units in each layer. By running the LC algorithm at different compression levels $\kappa > 0$, one can determine the smallest model that achieves a target loss that is good enough for the application at hand. In summary, a good approximate strategy for model selection in neural nets is to train a large enough reference model and compress it as much as possible.

8 Conclusion

We have described a general framework to obtain compressed models with optimal task performance based on casting compression, usually understood as a procedure, as constrained optimization in model parameter space. This accommodates many existing compression techniques and also sheds light on previous approaches that were derived procedurally and do not converge to an optimal compressed model, even if they are effective in practice. We have also given a general “learning-compression” (LC) algorithm, provably convergent under standard assumptions. The LC algorithm reuses two kinds of existing algorithms as a black-box, independently of each other: in the L step, *a learning algorithm for the task loss* (such as SGD on the cross-entropy), which requires the training set, and whose form is independent of the compression technique; and in the C step *a compression algorithm on the model parameters* (such as k -means or the SVD), whose form depends on the compression technique but is independent of the loss and training set. The L and C steps follow mathematically from the definition of the constrained problem; for example, the C step for quantization and low-rank compression results in k -means and the SVD, respectively, because the C step takes the form of a quadratic distortion problem in both cases. A model designer can try different losses or compression techniques by simply calling the appropriate routine in the L or C step.

In companion papers, we develop this framework for specific compression mechanisms and report experimental results that often exceed or are comparable with the published state of the art, but with the additional advantages of generality, simplicity and convergence guarantees. Because of this, we think our

framework may be a useful addition to neural net toolboxes. Our framework also opens further research avenues that we are actively exploring.

Acknowledgements

Work supported by NSF award IIS-1423515. I thank Yerlan Idelbayev (UC Merced) for useful discussions.

A A convergence theorem for the quadratic-penalty method

For reference, we quote a theorem from Nocedal and Wright (2006) that we use to prove convergence of the LC algorithm (using the QP) in our theorem 5.1.

Consider the equality-constrained problem

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad (\mathbf{x}) = \mathbf{0} \quad (23)$$

where $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable and $(\mathbf{x}) = (c_1(\mathbf{x}), \dots, c_m(\mathbf{x}))^T \in \mathbb{R}^m$ are m equality constraints, also continuously differentiable. Define the quadratic-penalty function

$$Q(\mathbf{x}; \mu) = f(\mathbf{x}) + \frac{\mu}{2} \sum_{i=1}^m c_i^2(\mathbf{x}) \quad (24)$$

where $\mu > 0$ is the penalty parameter. Assume we are given a sequence $0 < \mu_1 < \mu_2 < \dots < \infty$, a nonnegative sequence of tolerances (τ_k) with $(\tau_k) \rightarrow 0$ and an starting point \mathbf{x}_0 . The quadratic-penalty method works by finding, at each iterate $k = 0, 1, 2, \dots$, an approximate minimizer \mathbf{x}_k of $Q(\mathbf{x}; \mu_k)$, starting at \mathbf{x}_{k-1} and terminating when $\|\nabla_{\mathbf{x}} Q(\mathbf{x}; \mu_k)\| \leq \tau_k$.

Theorem A.1. *Suppose that the tolerances and penalty parameters satisfy $(\tau_k) \rightarrow 0$ and $(\mu_k) \rightarrow \infty$. Then, if a limit point \mathbf{x}^* of the sequence (\mathbf{x}_k) is infeasible, it is a stationary point of the function $\|(\mathbf{x})\|^2$. On the other hand, if a limit point \mathbf{x}^* is feasible and the constraint gradients $\nabla c_i(\mathbf{x}^*)$, $i = 1, \dots, m$ are linearly independent, then \mathbf{x}^* is a KKT point for the problem (23). For such points, we have for any infinite subsequence \mathcal{K} such that $\lim_{k \in \mathcal{K}} \mathbf{x}_k = \mathbf{x}^*$ that $\lim_{k \in \mathcal{K}} (-\mu_k c_i(\mathbf{x}_k)) = \lambda_i^*$, for $i = 1, \dots, m$, where $\boldsymbol{\lambda}^*$ is the multiplier vector that satisfies the KKT conditions for problem (23).*

Proof. See Nocedal and Wright (2006, theorem 17.2). □

Note that the QP method does not use the Lagrange multipliers in any way; the fact that $-\mu_k c_i(\mathbf{x}_k)$ tends to the Lagrange multiplier for constraint i is a subproduct of the fact that the iterates converge to a KKT point. The AL method improves over the QP precisely by capitalizing on the availability of those estimates of the Lagrange multipliers.

B Learning rates for the L step: theorems and proofs

B.1 Optimization of a convex loss using gradient descent with a fixed step size

First we present a few well-known results about gradient-based optimization for convex functions, with a short proof if possible, and then apply them to our L step objective function (13).

B.1.1 Convergence theorems

A function $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is convex iff $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\lambda \in [0, 1]$: $f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{y}) \leq \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y})$ (and strictly convex if the inequality is strict). Let f be convex and differentiable. We say f is strongly convex with constant $l > 0$ if $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$: $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T (\mathbf{y} - \mathbf{x}) + \frac{1}{2}l \|\mathbf{y} - \mathbf{x}\|^2$. A function $\mathbf{G}: \mathbb{R}^n \rightarrow \mathbb{R}^m$ is Lipschitz continuous with Lipschitz constant $M > 0$ if $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$: $\|\mathbf{G}(\mathbf{x}) - \mathbf{G}(\mathbf{y})\| \leq M \|\mathbf{x} - \mathbf{y}\|$. All norms are Euclidean in this section. Most of the statements apply if the convex function is defined on a convex subset of \mathbb{R}^n . For further details, see a standard reference such as Nesterov (2004).

Lemma B.1. Let $f_1, f_2: \mathbb{R}^n \rightarrow \mathbb{R}$ be Lipschitz continuous with respective Lipschitz constants $M_1, M_2 > 0$. Then $f = f_1 + f_2$ is Lipschitz continuous with Lipschitz constant $M_1 + M_2$.

Proof. $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n: \|f(\mathbf{x}) - f(\mathbf{y})\| = \|f_1(\mathbf{x}) - f_1(\mathbf{y}) + f_2(\mathbf{x}) - f_2(\mathbf{y})\| \leq \|f_1(\mathbf{x}) - f_1(\mathbf{y})\| + \|f_2(\mathbf{x}) - f_2(\mathbf{y})\| \leq (M_1 + M_2) \|\mathbf{x} - \mathbf{y}\|$, by applying the triangle inequality. \square

Lemma B.2. Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be differentiable. Then f is convex if and only if $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n: f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$.

Proof. (\Rightarrow) Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Since f is convex, we have $\forall \lambda \in [0, 1]: (1 - \lambda)f(\mathbf{x}) + \lambda f(\mathbf{y}) \geq f((1 - \lambda)\mathbf{x} + \lambda\mathbf{y}) \Rightarrow f(\mathbf{y}) \geq f(\mathbf{x}) + \frac{1}{\lambda}(f(\mathbf{x} + \lambda(\mathbf{y} - \mathbf{x})) - f(\mathbf{x}))$, which tends to $f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$ as $\lambda \rightarrow 0$.
(\Leftarrow) Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, $\lambda \in [0, 1]$ and $\mathbf{z} = \lambda\mathbf{x} + (1 - \lambda)\mathbf{y}$. Then, applying the assumption to (\mathbf{z}, \mathbf{x}) and to (\mathbf{z}, \mathbf{y}) , we get the following two inequalities: $f(\mathbf{x}) \geq f(\mathbf{z}) + \nabla f(\mathbf{z})^T(\mathbf{x} - \mathbf{z})$ and $f(\mathbf{y}) \geq f(\mathbf{z}) + \nabla f(\mathbf{z})^T(\mathbf{y} - \mathbf{z})$. Multiplying the first by λ , the second by $1 - \lambda$ and summing, we obtain $\lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{y}) \geq f(\mathbf{z}) = f(\lambda\mathbf{x} + (1 - \lambda)\mathbf{y})$. \square

Lemma B.3. Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be convex and continuously differentiable, and $\nabla f: \mathbb{R}^n \rightarrow \mathbb{R}^n$ be Lipschitz continuous with Lipschitz constant $M > 0$. Then $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n: f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}M \|\mathbf{y} - \mathbf{x}\|^2$.

Proof. See Nesterov (2004, lemma 1.2.3). \square

Lemma B.4. Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be convex. Then $F(\mathbf{x}) = f(\mathbf{x}) + \frac{\mu}{2} \|\mathbf{x} - \mathbf{a}\|^2$ where $\mathbf{a} \in \mathbb{R}^n$ and $\mu > 0$ is strongly convex with constant μ .

Proof. Call $g(\mathbf{x}) = \frac{\mu}{2} \|\mathbf{x} - \mathbf{a}\|^2$ so $F = f + g$. Then, since f is convex: $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$ $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, and one can verify by substitution that $g(\mathbf{y}) = g(\mathbf{x}) + \nabla g(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}\mu \|\mathbf{y} - \mathbf{x}\|^2$ $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. Summing both expressions we get $F(\mathbf{y}) \geq F(\mathbf{x}) + \nabla F(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}\mu \|\mathbf{y} - \mathbf{x}\|^2$ $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. \square

Theorem B.5. Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be strongly convex with constant $m > 0$ and continuously differentiable, and let its gradient ∇f be Lipschitz continuous with Lipschitz constant $M > 0$. Given any $\mathbf{x}^0 \in \mathbb{R}^n$, define the sequence $\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{1}{M} \nabla f(\mathbf{x}_t)$ for $t = 0, 1, 2, \dots$. Then f has a unique global minimizer $\mathbf{x}^* \in \mathbb{R}^n$ and $f(\mathbf{x}_t) - f(\mathbf{x}^*) \leq (1 - \frac{m}{M})_t (f(\mathbf{x}^0) - f(\mathbf{x}^*))$ for $t = 0, 1, 2, \dots$

Proof. Since f is strongly convex it has a unique minimizer $\mathbf{x}^* \in \mathbb{R}^n$. From lemma B.3:

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n: f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}M \|\mathbf{y} - \mathbf{x}\|^2.$$

By differentiating wrt \mathbf{x} , we see the RHS is minimal at $\mathbf{y} - \mathbf{x} = -\frac{1}{M} \nabla f(\mathbf{x})$ and $f(\mathbf{y}) \leq f(\mathbf{x}) - \frac{1}{2M} \|\nabla f(\mathbf{x})\|^2$. Applying this to $\mathbf{x} = \mathbf{x}_t$ and $\mathbf{y} = \mathbf{x}_{t+1}$, we get $\mathbf{x}_{t+1} = \mathbf{x}_t - \frac{1}{M} \nabla f(\mathbf{x}_t)$ (a gradient descent step at \mathbf{x}_t with step size $\frac{1}{M}$) and

$$f(\mathbf{x}_{t+1}) \leq f(\mathbf{x}_t) - \frac{1}{2M} \|\nabla f(\mathbf{x}_t)\|^2, \quad (25)$$

which gives a lower bound on how much f decreases from \mathbf{x}_t to \mathbf{x}_{t+1} . Since f is strongly convex:

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n: f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}m \|\mathbf{y} - \mathbf{x}\|^2.$$

As a function of \mathbf{y} , the RHS is minimal at $\mathbf{y} = \mathbf{x} - \frac{1}{m} \nabla f(\mathbf{x})$ and equals $f(\mathbf{x}) - \frac{1}{2m} \|\nabla f(\mathbf{x})\|^2$. Hence, $f(\mathbf{y}) \geq f(\mathbf{x}) - \frac{1}{2m} \|\nabla f(\mathbf{x})\|^2$ $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. In particular, taking $\mathbf{y} = \mathbf{x}^*$ and $\mathbf{x} = \mathbf{x}_t$:

$$f(\mathbf{x}^*) \geq f(\mathbf{x}_t) - \frac{1}{2m} \|\nabla f(\mathbf{x}_t)\|^2,$$

which gives an upper bound on how far $f(\mathbf{x}_t)$ is from the minimum. Combining this with eq. (25) we get $f(\mathbf{x}_{t+1}) - f(\mathbf{x}^*) \leq (1 - \frac{m}{M})(f(\mathbf{x}_t) - f(\mathbf{x}^*))$ and the result follows. \square

Remark B.6. Theorem B.5 shows that, if f is strongly convex with constant m and its gradient is Lipschitz continuous with constant M , gradient descent with a constant step size $\frac{1}{M}$ converges linearly with rate $0 \leq 1 - \frac{m}{M} < 1$ from any initial point.

B.1.2 Application to the L step of the LC algorithm

We now apply these results to our case. To train the reference model, we minimize the loss $L(\mathbf{w})$. If we assume L is convex differentiable with Lipschitz continuous gradient with Lipschitz constant $M > 0$, we could use gradient descent with a fixed step size $\frac{1}{M}$ and have guaranteed convergence. In the L step, we minimize an objective function $Q(\mathbf{w}) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}'\|^2$ with $\mu > 0$. We have that Q is strongly convex with constant $\mu > 0$ (lemma B.4), so it has a unique minimizer, and its gradient is Lipschitz continuous with Lipschitz constant $M + \mu$ (lemma B.1). Hence, gradient descent on Q with fixed step size $\frac{1}{M+\mu}$, $\mathbf{w}_{t+1} = \mathbf{w}_t - \frac{1}{M+\mu} \nabla Q(\mathbf{w}_t)$, converges to the minimizer linearly with rate $1 - \frac{\mu}{M+\mu} = \frac{M}{M+\mu} \in (0, 1)$ from any initial point. (If the loss is strongly convex with constant $0 < m < M$, then we have a faster rate of $\frac{M-m}{M+\mu}$.)

B.2 Optimization using stochastic gradient descent (SGD)

First we present some theorems about the convergence of SGD-type algorithms and then apply them to our L step objective function (13).

B.2.1 Convergence theorems

There is a large literature on convergence of SGD-type algorithms (Pflug, 1996; Benveniste et al., 1990; Kushner and Yin, 2003), although the basic conditions for convergence are similar (Lipschitz continuity of the gradient, approximate descent search directions, bounded error if deterministic or unbiased error with bounded variance if stochastic, and Robbins-Monro step sizes). We quote theorems from Bertsekas and Tsitsiklis (2000), which are simple and consider the cases of deterministic errors, incremental gradient algorithm and stochastic errors.

Let $f: \mathbb{R}^n \rightarrow \mathbb{R}$ be a continuously differentiable function with Lipschitz continuous gradient ∇f . Consider minimization of $f(\mathbf{x})$ using the approximate descent method $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t(\mathbf{p}_t + \mathbf{e}_t)$ for $t = 0, 1, 2, \dots$, where \mathbf{p}_t is a search direction and \mathbf{e}_t an error vector. We say that the step sizes are Robbins-Monro if they are positive and satisfy

$$\sum_{t=0}^{\infty} \eta_t = \infty, \quad \sum_{t=0}^{\infty} \eta_t^2 < \infty. \quad (26)$$

Theorem B.7 (deterministic errors). *Let \mathbf{x}_t be a sequence generated by the method $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t(\mathbf{p}_t + \mathbf{e}_t)$, where η_t is a positive step size, \mathbf{p}_t is a descent direction and \mathbf{e}_t is a error vector. We assume the following:*

1. *There exist positive scalars c_1 and c_2 such that, for all t :*

$$c_1 \|\nabla f(\mathbf{x}_t)\|^2 \leq -\nabla f(\mathbf{x}_t)^T \mathbf{p}_t \quad \|\mathbf{p}_t\| \leq c_2(1 + \|\nabla f(\mathbf{x}_t)\|). \quad (27)$$

2. *There exist positive scalars p and q such that, for all t :*

$$\|\mathbf{e}_t\| \leq \eta_t(q + p \|\nabla f(\mathbf{x}_t)\|). \quad (28)$$

3. *The step sizes are Robbins-Monro.*

Then either $f(\mathbf{x}_t) \rightarrow -\infty$ or else $f(\mathbf{x}_t)$ converges to a finite value and $\lim_{t \rightarrow \infty} \nabla f(\mathbf{x}_t) = \mathbf{0}$. Furthermore, every limit point of \mathbf{x}_t is a stationary point of f .

Proof. See proposition 1 in Bertsekas and Tsitsiklis (2000). □

This theorem can be particularized to the following important case: 1) f has the form $f(\mathbf{x}) = \sum_{i=1}^m f_i(\mathbf{x})$ and each f_i is continuously differentiable with Lipschitz continuous gradient. 2) We use the incremental gradient algorithm, where we cycle over f_1, \dots, f_m , each time updating \mathbf{x}_{t+1} using the gradient of f_i at \mathbf{x}_t . This corresponds to SGD using a minibatch of size 1 and without reshuffling the dataset at each epoch.

Theorem B.8 (incremental gradient method). *Let \mathbf{x}_t be a sequence generated by the incremental gradient method. Assume that for some positive constants C and D , and all $i = 1, \dots, m$, we have*

$$\|\nabla f_i(\mathbf{x})\| \leq C + D \|\nabla f(\mathbf{x})\| \quad \forall \mathbf{x} \in \mathbb{R}^n. \quad (29)$$

Assume that the step sizes are Robbins-Monro. Then either $f(\mathbf{x}_t) \rightarrow -\infty$ or else $f(\mathbf{x}_t)$ converges to a finite value and $\lim_{t \rightarrow \infty} \nabla f(\mathbf{x}_t) = \mathbf{0}$. Furthermore, every limit point of \mathbf{x}_t is a stationary point of f .

Proof. See proposition 2 in Bertsekas and Tsitsiklis (2000). \square

Now we let the noise term \mathbf{e}_t be stochastic. The σ -field \mathcal{F}_t should be interpreted as the history of the algorithm up to time t , just before \mathbf{e}_t is generated, so that conditioning on \mathcal{F}_t represents conditioning on $\mathbf{x}_0, \mathbf{p}_0, \mathbf{e}_0, \dots, \mathbf{x}_{t-1}, \mathbf{p}_{t-1}, \mathbf{e}_{t-1}, \mathbf{x}_t, \mathbf{p}_t$.

Theorem B.9 (stochastic errors). *Let \mathbf{x}_t be a sequence generated by the method $\mathbf{x}_{t+1} = \mathbf{x}_t + \eta_t(\mathbf{p}_t + \mathbf{e}_t)$, where η_t is a deterministic positive step size, \mathbf{p}_t is a descent direction and \mathbf{e}_t is a random noise term. Let \mathcal{F}_t be an increasing sequence of σ -fields. We assume the following:*

0. \mathbf{x}_t and \mathbf{p}_t are \mathcal{F}_t -measurable.

1. There exist positive scalars c_1 and c_2 such that, for all t :

$$c_1 \|\nabla f(\mathbf{x}_t)\|^2 \leq -\nabla f(\mathbf{x}_t)^T \mathbf{p}_t \quad \|\mathbf{p}_t\| \leq c_2(1 + \|\nabla f(\mathbf{x}_t)\|). \quad (30)$$

2. We have, for all t and with probability 1,

$$\mathbb{E}\{\mathbf{e}_t | \mathcal{F}_t\} = \mathbf{0}, \quad \mathbb{E}\{\|\mathbf{e}_t\|^2 | \mathcal{F}_t\} \leq A(1 + \|\nabla f(\mathbf{x}_t)\|^2), \quad (31)$$

where A is a positive deterministic constant.

3. The step sizes are Robbins-Monro.

Then, the following holds with probability 1: either $f(\mathbf{x}_t) \rightarrow -\infty$ or else $f(\mathbf{x}_t)$ converges to a finite value and $\lim_{t \rightarrow \infty} \nabla f(\mathbf{x}_t) = \mathbf{0}$. Furthermore, every limit point of \mathbf{x}_t is a stationary point of f .

Proof. See proposition 3 in Bertsekas and Tsitsiklis (2000). \square

B.2.2 Application to the L step of the LC algorithm

The theorems give sufficient conditions for convergence to a stationary point (usually a minimizer). Their effect in the L step if using SGD are as follows:

1. The conditions (27) and (30) on the search direction \mathbf{p}_t are always satisfied since it equals the gradient ($\mathbf{p}_t = -\nabla f(\mathbf{x}_t)$ in the theorems).

2. The conditions on the error or noise (28), (29) and (31) are hard to verify in general but should hold in many practical cases.

It is tempting to think that the condition should hold for the L step objective function $Q(\mathbf{w}) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}'\|^2$ if it holds for the loss $L(\mathbf{w})$, since the gradient for the μ -term has no error, but this is not true generally. To see this, assume the error condition holds for a function f and consider a function $f + g$, where ∇g has no error, so the error comes from ∇f only (f and g correspond to the loss and μ -term, respectively). We could pick g adversarially so that the error is small wrt ∇f but big wrt $\nabla(f + g)$ (e.g. if $g = -f$). Although this could be solved by placing some assumptions on g , we may just as well assume the error condition on $f + g$.

3. The condition on the step sizes is that they be Robbins-Monro.

Consequently, the theorems hold for SGD minimization of $Q(\mathbf{w}) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \mathbf{w}'\|^2$ if the error on ∇Q satisfies (28), (29) or (31) and the step sizes are Robbins-Monro. Hence, there is neither a simplification nor a complication of minimizing Q over minimizing L with SGD: the convergence theory leaves the choice of step sizes up to the user as long as they are Robbins-Monro.

References

- J. Ba and R. Caruana. Do deep nets really need to be deep? 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 2654–2662. MIT Press, Cambridge, MA, 2014.
- M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.
- A. Benveniste, M. Métivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations*, volume 22 of *Applications of Mathematics*. Springer-Verlag, Berlin, 1990.
- D. P. Bertsekas and J. N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM J. Optimization*, 10(3):627–642, 2000.
- C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, New York, Oxford, 1995.
- I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Application*. Springer Series in Statistics. Springer-Verlag, Berlin, second edition, 2005.
- M. Á. Carreira-Perpiñán. The elastic embedding algorithm for dimensionality reduction. In J. Fürnkranz and T. Joachims, editors, *Proc. of the 27th Int. Conf. Machine Learning (ICML 2010)*, pages 167–174, Haifa, Israel, June 21–25 2010.
- M. Á. Carreira-Perpiñán and Y. Idelbayev. Model compression as constrained optimization, with application to neural nets. Part II: Quantization. arXiv, July 2017a.
- M. Á. Carreira-Perpiñán and Y. Idelbayev. Model compression as constrained optimization, with application to neural nets. Part III: Pruning. arXiv, July 2017b.
- M. Á. Carreira-Perpiñán and M. Vladymyrov. A fast, universal algorithm to learn parametric nonlinear embeddings. 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 253–261. MIT Press, Cambridge, MA, 2015.
- M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. arXiv:1212.5921 [cs.LG], Dec. 24 2012.
- M. Á. Carreira-Perpiñán and W. Wang. Distributed optimization of deeply nested systems. In S. Kaski and J. Corander, editors, *Proc. of the 17th Int. Conf. Artificial Intelligence and Statistics (AISTATS 2014)*, pages 10–19, Reykjavik, Iceland, Apr. 22–25 2014.
- W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In F. Bach and D. Blei, editors, *Proc. of the 32nd Int. Conf. Machine Learning (ICML 2015)*, pages 2285–2294, Lille, France, July 6–11 2015.
- 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.
- M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas. Predicting parameters in deep learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 26, pages 2148–2156. MIT Press, Cambridge, MA, 2013.
- 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.

- 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.
- M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.
- A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers Group, 1992.
- 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.
- S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan. Deep learning with limited numerical precision. In F. Bach and D. Blei, editors, *Proc. of the 32nd Int. Conf. Machine Learning (ICML 2015)*, pages 1737–1746, Lille, France, July 6–11 2015.
- 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.
- S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.
- S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS)*, volume 1, pages 177–185. Morgan Kaufmann, San Mateo, CA, 1989.
- B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 5, pages 164–171. Morgan Kaufmann, San Mateo, CA, 1993.
- G. Hinton and S. T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 15, pages 857–864. MIT Press, Cambridge, MA, 2003.
- G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. arXiv:1503.02531, Mar. 9 2015.
- I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. arXiv:1609.07061, Sept. 22 2016.
- K. Hwang and W. Sung. Fixed-point feedforward deep neural network design using weights +1, 0, and -1. In *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, pages 1–6, Belfast, UK, Oct. 20–22 2014.
- M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In M. Valstar, A. French, and T. Pridmore, editors, *Proc. of the 25th British Machine Vision Conference (BMVC 2014)*, Nottingham, UK, Sept. 1–5 2014.
- H. J. Kushner and G. G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer Series in Stochastic Modelling and Applied Probability. Springer-Verlag, second edition, 2003.
- Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 598–605. Morgan Kaufmann, San Mateo, CA, 1990.
- F. Li, B. Zhang, and B. Liu. Ternary weight networks. arXiv:1605.04711, Nov. 19 2016.

- M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast neural networks without multipliers. *IEEE Trans. Neural Networks*, 4(1):53–62, Jan. 1993.
- B. K. Natarajan. Sparse approximate solutions to linear systems. *SIAM J. Comp.*, 24(2):227–234, 1995.
- Y. Nesterov. *Introductory Lectures on Convex Optimization. A Basic Course*. Number 87 in Applied Optimization. Springer-Verlag, 2004.
- J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.
- A. Novikov, D. Podoprikin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. 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 442–450. MIT Press, Cambridge, MA, 2015.
- S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, July 1992.
- G. C. Pflug. *Optimization of Stochastic Models: The Interface between Simulation and Optimization*. Kluwer Academic Publishers Group, 1996.
- M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-net: ImageNet classification using binary convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Proc. 14th European Conf. Computer Vision (ECCV’16)*, pages 525–542, Amsterdam, The Netherlands, Oct. 11–14 2016.
- R. Reed. Pruning algorithms—a survey. *IEEE Trans. Neural Networks*, 4(5):740–747, Sept. 1993.
- T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP’13)*, pages 6655–6659, Vancouver, Canada, Mar. 26–30 2013.
- C. Z. Tang and H. K. Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Trans. Signal Processing*, 41(8):2724–2727, Aug. 1993.
- K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.
- L. J. P. van der Maaten and G. E. Hinton. Visualizing data using *t*-SNE. *J. Machine Learning Research*, 9: 2579–2605, Nov. 2008.
- A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In R. P. Lippmann, J. E. Moody, and D. S. Touretzky, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 3, pages 875–882. Morgan Kaufmann, San Mateo, CA, 1991.
- D. Yu, F. Seide, G. Li, and L. Deng. Exploiting sparseness in deep neural networks for large vocabulary speech recognition. In *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP’12)*, pages 4409–4412, Kyoto, Japan, Mar. 25–30 2012.
- 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. arXiv:1606.06160, July 17 2016.
- Z.-H. Zhou. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Machine Learning and Pattern Recognition Series. CRC Publishers, 2012.
- C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.