ÉCOLE NORMALE SUPÉRIEURE
DÉPARTEMENT D'INFORMATIQUE
RAPPORT DE STAGE DE MASTER 1

Neural Network Compression

David A. R. Robin
david.robin@ens.fr

Feb-Aug 2019

Advisor
Swayambhoo Jain

Contents

1 Context

1.1 Neural network overparameterization

Neural networks have demonstrated great empirical success in a wide range of machine-learning tasks. Such models are typically heavily overparameterized, and trained networks exhibit high levels of redundancy. Denil et al. [2013] for instance is able to predict over 90% of weights in a convolutional neural network. Similarly, Hinton et al. [2015] transfers knowledge from a trained overparameterized network into a shallower student network, to the point where the student is able to achieve the same accuracy as its teacher, even with an order of magnitude fewer weights. Precisely how many weights are required to perform a given task accurately is not well understood yet, but recent works are starting to prove that this overparameterization is benefical during the training. Allen-Zhu et al. [2018] proved that sufficiently overparameterized networks can converge to *global* optimum with stochastic gradient descent, even when the objective is not only non-convex, but also non-smooth. Although being beneficial during the training, such an overparameterization may not be required to achieve similar accuracy. It turns out that most of the weights of a network can often be removed without inducing significant drops in accuracy [LeCun et al., 1990, Hassibi and Stork, 1993, Han et al., 2015a].

From an architecture design point of view, it is also much easier to overparameterize an architecture to make sure it will be able to complete the task at hand, without thinking about the associated complexity, and defer the memory concerns to a later design stage, as was the case for instance with *SqueezeNet* [Iandola et al., 2016], a more memory-efficient alternative to AlexNet achieving the same performance with 50 times fewer parameters, presented four years after the original AlexNet architecture.

> The goal of compression is to compute light-weight approximations of large trained neural networks to curb their storage requirements while maintaining similar prediction accuracy.

1.2 A note on retraining

Recent works in network compression have tended to split the compression pipeline into an approximation step aggressively compressing the network and incuring a severe accuracy loss, and a second retraining step meant to help the compressed network recover the initial accuracy. The accuracy is thus not retained throughout compression, but rather lost and then recovered given enough retraining. This iterative retraining is similar to the initial training step, and in some cases the retraining time needed to do so may be of the same order of magnitude of the initial training time. It is unclear whether compression methods relying on a heavy retraining step truly act as a compression of the initial model, or only use the said model to find a smaller architecture equally well suited to the task which is then trained during the "retraining" as it would have been trained from scratch.

Liu et al. [2019] shows that in the case of weight pruning, it is possible to randomly reinitialize the weights before the retraining step, and still achieve the same accuracy as the non-reinitialized network with a similar training time. This hints towards the fact that weight pruning uncovers a good architecture, more than it actually compresses a network. In order to avoid hiding these issues behind the retraining step, we focus in this work in compression without iterative retraining. Any such compression can easily be extended to the previous setting by simply appending an iterative retraining step to regain any possibly lost accuracy.

1.3 Connection pruning

The earliest works that can be identified as network compression were concerned with overfitting rather than compression, and proposed to reduce the number of free parameters by drawing information from the Hessian of the network's loss function and removing unimportant connections between neurons.

Writing $\mathbf{w} \in \mathbb{R}^n$ the weights of a neural network, and $\mathcal{L} : \mathbb{R}^n \mapsto \mathbb{R}$ its loss function, the Taylor-Young expansion of the loss to third order is written

$$\mathcal{L}(\mathbf{w} + \delta\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})^T \cdot \delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^T \cdot \nabla^2\mathcal{L}(\mathbf{w}) \cdot \delta\mathbf{w} + \mathcal{O}(\|\delta\mathbf{w}\|^3)$$

The first order term is zero for a fully trained network, since $\nabla\mathcal{L}(\mathbf{w}) = 0$. When setting a single entry of $\mathbf{w}$ to zero, say the q^{th}, we have $\delta\mathbf{w} = -w_q \cdot \mathbf{e}_q$, which gives an approximate loss increase of

$$\mathcal{L}(\mathbf{w} + \delta\mathbf{w}) - \mathcal{L}(\mathbf{w}) \approx \frac{1}{2}w_q^2 \cdot \nabla^2\mathcal{L}(\mathbf{w})_{q,q}$$

This means that the error increase caused by the deletion of a single weight is proportional to the magnitude w_q^2 of the said weight. Naturally, if one were to delete a weight, it would be wise to update the remaining weights accordingly. When deleting weight w_q, we can minimize the approximate loss increase stated above over $\delta\mathbf{w}$ under the constraint $\delta\mathbf{w}^T e_q + w_q = 0$, which gives[1]

$$\delta\mathbf{w} = -\frac{w_q}{(\nabla^2\mathcal{L}(\mathbf{w})^{-1})_{q,q}}\nabla^2\mathcal{L}(\mathbf{w})^{-1}e_q \qquad \mathcal{L}(\mathbf{w} + \delta\mathbf{w}) - \mathcal{L}(\mathbf{w}) = \frac{1}{2}\frac{w_q^2}{(\nabla^2\mathcal{L}(\mathbf{w})^{-1})_{q,q}}$$

This yields an iterative single-weight pruning algorithm presented in LeCun et al. [1990] and Hassibi and Stork [1993] (also known as "optimal brain damage" and "optimal brain surgeon").

For big networks however, the Hessian quickly becomes intractable, and inverting it on every iteration is unreasonably expensive. Inspired by the observation that the loss increase when deleting a single weight is proportional to its magnitude, Han et al. [2015b] suggest to delete all weights whose magnitude lie below a certain threshold (note that the loss increase proportionality to magnitude does not hold in this case for non-diagonal Hessians, i.e. most of the time). This works surprisingly well, and Han et al. [2015a] shows that given enough retraining, up to 90% of the weights can be discarded in this fashion without noticing an accuracy drop. Although the extensive retraining mandatory for this method to work make it questionnable [Liu et al., 2019], it at least shows that typical neural network architectures are highly compressible. Dong et al. [2017] tackles the computational issue differently, by leveraging a layer-wise approximation of the Hessian to estimate parameter's saliency without directly needing to compute the Hessian or its inverse, and thus avoid removing important yet low-magnitude weights, limiting the required retraining, but not eliminating it.

From a compression point of view, these approaches all have the drawback of preserving the size of the weight matrices despite reducing their number of non-zero entries, inducing a storage overhead compared to smaller matrices with the same number of free parameters. Indeed, storing an (n, n) matrix with a non-zero entries require $2a + n$ numbers[2] because one has to store not only the values of these non-zero entries, but also their position in the bigger matrix. Han et al. [2015a] reports that this index overhead accounts for about half of the final storage cost of heavily pruned models. This incentivizes research to reduce the size of the matrices involved to obtain truly optimal compression.

[1] see appendix for a proof of this result, section 5.1

[2] see appendix for details on this encoding, section 5.2

Magnitude-based pruning is equivalent to solving the following optimization problem, removing connections between neurons as shown in Figure 1

$$\min_{\substack{\mathbf{M} \\ \|\mathbf{M}\|_0 \leq p}} \| \mathbf{W} - \mathbf{M} \|_F^2$$

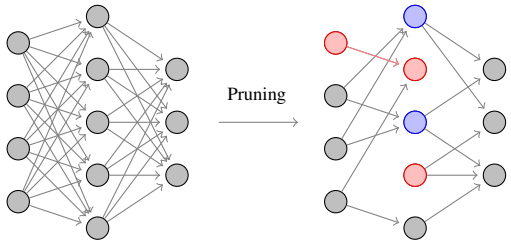

Figure 1: Magnitude pruning: dangling neurons (red) and remaining redundancies (blue)

On top of the sparsity issue, magnitude-based pruning alone as a compression method suffers from the following drawbacks. First, it may leave dangling neurons whose output is not used because they are followed by low-magnitude weights or whose input has been deleted (in red in Figure 1). Such weights could be discarded at no cost to improve the compression, but are kept because they have high magnitudes. Han et al. [2015a] argues that this is not an issue since enough retraining with weight decay will eventually bring these useless weights to zero. Another similar issue is that redundancies are not leveraged, neurons having the exact same weights perform redundant computations (in blue in Figure 1) and one of them could be discarded since the other may be used instead at no cost, yet because they have the same magnitude they are either both kept or both discarded.

1.4 Low-rank approximations

From a deep view, a "redundancy" in the network is essentially something that can be "factored away". A natural way to do so when working with linear operators thus comes to mind: low-rank factorization of weight matrices. Low-rank approximations of weight matrices can be computed through singular value decomposition, and several works have shown how to extend this to higher order convolutional kernels via canonical polyadic decompositions [Denton et al., 2014, Jaderberg et al., 2014, Tai et al., 2015, Lebedev et al., 2014].

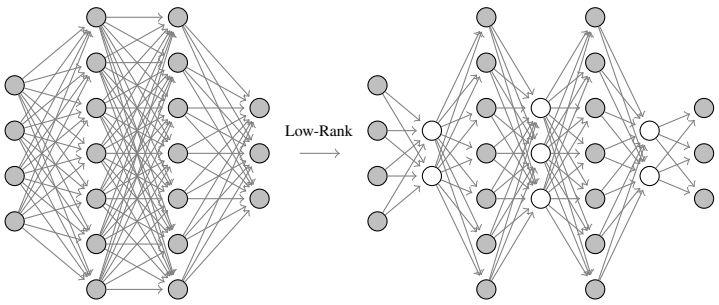

Figure 2: Low-rank approximation shape change

A rank-r approximation of an (n, m) matrix will take the storage cost from $\mathcal{O}(nm)$ to $\mathcal{O}(r\,(n+m))$. Although significantly curbing the storage cost of the network, this has the drawback of preserving the number of input and output neurons of a layer. This essentially means that the low-rank approximation is only as good as the initial architecture, in the sense that if too many hidden neurons were selected during architecture search, then the low-rank compression cannot be optimal, since the storage cost will be lower bounded by a linear function of this initial choice: the low-rank method is not able to recover from this bad architectural choice.

1.5 Neuron removal

He et al. [2014] proposes to fix this neuron count preservation issue by estimating the importance of hidden neurons and dropping the hidden neurons with the lowest scores. Alternatively, Srinivas and Babu [2015] analyzes weight matrices to merge neurons with similar weights, updating the weights of the next layer to account for this merging instead of just dropping some neurons. Mariet and Sra [2015] go further in this direction by merging neurons that, as a group, perform redundant calculations, rather than individually merging pairs of neurons. This is achieved by defining a probability measure over subsets of neurons, and sampling from it to get a "diverse" set of neurons. The weights of the following layer are then updated to only rely on the sampled neurons, effectively removing the others. We show however that splitting the selection and re-weighting is not necessary, and we can instead jointly optimize both, leading to better solutions.

1.6 Activation reconstruction

Since the goal is to preserve only the accuracy of the network, and not the network itself, it is not required to approximate the weight matrices themselves, but only their action with respect to a particular input distribution. In particular, two neurons giving similar outputs despite having very different weights, for instance because they rely on redundancies in their inputs, should be considered similar for the purpose of compression.

Consider for concreteness a feed-forward fully-connected L-layer network with weights $(\mathbf{W}_k)_{k<L}$ and non-linearities $(\sigma_k)_{k<L}$ whose input is a random variable $Z_0 \sim \mathcal{D}_0$. The activations after each layer are given by $Z_{k+1} = \sigma_k(\mathbf{W}_k \cdot Z_k)$. Approximating the weight matrices is enough to approximate the final output Z_L, since

$$\hat{Z}_k \approx Z_k \,,\; \hat{\mathbf{W}}_k \approx \mathbf{W}_k \;\Rightarrow\; \hat{Z}_{k+1} \approx Z_{k+1} \tag{1}$$

However, the property we really need in order to propagate this approximation is not control over an intrinsic distance between weight matrices, but control over a distance between the layer's actions

$$\hat{Z}_k \approx Z_k \,,\; \sigma_k(\hat{\mathbf{W}}_k Z_k) \approx \sigma_k(\mathbf{W}_k Z_k) \;\Rightarrow\; \hat{Z}_{k+1} \approx Z_{k+1} \tag{2}$$

We have the following relations between three kinds of weight approximations

$$\hat{\mathbf{W}}_k \approx \mathbf{W}_k \;\;\Rightarrow\;\; \hat{\mathbf{W}}_k Z_k \approx \mathbf{W}_k Z_k \;\;\Rightarrow\;\; \sigma_k(\hat{\mathbf{W}}_k Z_k) \approx \sigma_k(\mathbf{W}_k Z_k) \tag{3}$$

Note that no two of these approximations are equivalent in the general case. The weight approximation (first in Equation 3, propagated by Equation 1) is data-agnostic and thus potentially sub-optimal. The non-linear action approximation (third in Equation 3, propagated by 2) is obtained by solving a problem that is non-convex, in some cases non-smooth, and more generally arbitrarily complex since any mostly-differentiable function is an acceptable non-linearity for a neural network, which makes it a very hard problem to solve in general, so let us aim for the second approximation, i.e. *linear activation reconstruction*.

2 Contribution

2.1 Single layer approximation idea

Consider a fully connected layer with weight $\mathbf{W} \in \mathbb{R}^{h \times n}$. We wish to approximate its output with respect to an input distribution $\mathcal{D}$, approximated in practice by a finite number of samples consituting a training set. We consider in the following that the random variable X is drawn from $\mathcal{D}$, omitted in the indices to alleviate notations. We build on top of the low-rank approximation idea, which for fully connected layers reduces to the following optimization problem

$$\min_{\mathbf{P} \in \mathbb{R}^{h \times r},\, \mathbf{Q} \in \mathbb{R}^{n \times r}} \mathbb{E}_X \left\| \mathbf{W}X - \mathbf{P}\mathbf{Q}^T X \right\|_2^2$$

$\mathbf{Q}$ in this setting may be interpreted as a linear feature extractor, and $\mathbf{P}$ as a linear reconstruction of the output from the extracted features. To avoid manual selection of the target rank, one may relax this into a low-rank representation problem with a smooth matching constraint, and use the common relaxation of the rank into a nuclear norm to obtain instead the following problem

$$\min_{\mathbf{M}} \; \mathbb{E}_X \left\| \mathbf{W}X - \mathbf{M}X \right\|_2^2 + \lambda \cdot \left\| \mathbf{M} \right\|_*$$

where λ is a parameter that controls the tradeoff between compression and accuracy. High values of λ will favor smaller networks, at the expense of approximation quality, whereas lower values will give more accurate approximations but increase the storage cost.

As we have seen above, this approach does not as-is allow a reduction of the number of neurons. However, we may add a constraint to $\mathbf{Q}$ in order to gain a property that will allow us to do so. We thus propose to restrict the feature extractor of the low-rank approximation to be a feature selector, that is to say reduce the search space to matrices of the form $\mathbf{P}\mathbf{C}^T$, with $\mathbf{C} \in \{0,1\}^{n \times r}$ such that $\mathbf{C}^T\mathbf{1}_n = \mathbf{1}_r$ being a feature selector that selects r of the initial n input neurons.

2.2 Leveraging consecutive layers

Feature selectors have the interesting property that they can be applied indifferently before or after the non-linearity. This allows to significantly reduce the memory footprint of consecutive fully connected layers approximated in this fashion. Consider for concreteness three fully connected layers $\mathbf{W}_0$, $\mathbf{W}_1$ and $\mathbf{W}_2$ with d inputs, h_0 and h_1 hidden neurons and h_2 outputs, with layers separated by a non-linearities σ_0 and σ_1, computing the mapping

$$\mathbf{x} \in \mathbb{R}^d \;\mapsto\; \mathbf{W}_2 \cdot \sigma_1(\mathbf{W}_1 \cdot \sigma_0(\mathbf{W}_0 \cdot \mathbf{x})) \in \mathbb{R}^{h_2}$$

The approximations of the three layers as stated above allows the following rewritings

$$\begin{aligned}
& \mathbf{W}_2 \cdot \sigma_1(\quad \mathbf{W}_1 \cdot \sigma_0(\quad \mathbf{W}_0 \cdot \mathbf{x} \quad)) \\
\approx \; & \mathbf{P}_2\mathbf{C}_2^T \cdot \sigma_1(\quad \mathbf{P}_1\mathbf{C}_1^T \cdot \sigma_0(\quad \mathbf{P}_0\mathbf{C}_0^T \cdot \mathbf{x} \quad)) \\
= \; & \mathbf{P}_2 \cdot \sigma_1(\quad \mathbf{C}_2^T\mathbf{P}_1 \cdot \sigma_0(\quad \mathbf{C}_1^T\mathbf{P}_0 \cdot \mathbf{C}_0^T\mathbf{x} \quad)) \\
= \; & \hat{\mathbf{W}}_2 \cdot \sigma_1(\quad \hat{\mathbf{W}}_1 \cdot \sigma_0(\quad \hat{\mathbf{W}}_0 \cdot \mathbf{C}_0^T\mathbf{x} \quad))
\end{aligned}$$

Meaning that the output of the three-layer network $(\mathbf{W}_0, \mathbf{W}_1, \mathbf{W}_2)$ can be accurately approximated by a smaller three-layer network $(\mathbf{C}_1^T\mathbf{P}_0, \mathbf{C}_2^T\mathbf{P}_1, \mathbf{P}_2)$ of the same architecture with only a different number of hidden neurons, and an input sampler $\mathbf{C}_0^T$. If the associated ranks are respectively r_0, r_1 and r_2, the network size will be

$$\begin{aligned}
\text{original} \quad &: \quad h_2 \times h_1 \quad + \quad h_1 \times h_0 \quad + \quad h_0 \times d \\
\text{compressed} \quad &: \quad h_2 \times r_2 \quad + \quad r_2 \times r_1 \quad + \quad r_1 \times r_0 \quad + \quad \alpha \cdot \log_2 \binom{d}{r_0}
\end{aligned}$$

Note how h_1 and h_0 are not present in the compressed cost, only the number of inputs (d) and outputs (h_2) are preserved, other intermediate neurons can be removed by this approximation. The last term accounts for the storage of the first input sampler. There are (up to row reordering that can be applied to $\mathbf{P}$ instead) $\binom{d}{r_0}$ such samplers, thus a storage cost of $log_2\binom{d}{r_0}$ bits, and the α constant accounts for the fact that this cost is expressed in bits where the cost of the weight matrices were expressed in number of floating-point numbers to be stored ($\alpha = \texttt{sizeof(float)}^{-1}$). Note that this cost is negligible in practice since it cannot exceed d bits (for each input, 1 if it is kept, 0 otherwise, sufficient to encode the sampler).

The key property to ensure the low-rank approximation can be converted into a substantial storage gain is the commutation of σ and the action of $\mathbf{C}^T$. Although restricting the linear feature extractor $\mathbf{C}^T$ to be a feature selector might seem a little extreme, we argue that even for simple non-linearities like the commonly used ReLU, feature selectors are the only linear operators with this property (see Lemma 1, proof deferred to the appendix, section 5.3). This is of course to be expected since non-commutation with linear operators is a main feature of non-linearities that justified their introduction in neural networks in the first place, to prevent consecutive linear operators from collapsing into a single one.

Lemma 1 *If $\mathbf{C}$ is a linear operator whose action commutes with the pointwise ReLU non-linearity $\sigma : x \mapsto \max(0, x)$, then up to row rescaling, $\mathbf{C}$ is a feature selector.*

2.3 Comparison with low-rank

Low-Rank : $\mathbf{M} = \mathbf{P}\mathbf{Q}^T$
- $\mathbf{P} \in \mathbb{R}^{h \times r_Q}$
- $\mathbf{Q} \in \mathbb{R}^{d \times r_Q}$

Column-sparse : $\mathbf{M} = \mathbf{P}\mathbf{C}^T$
- $\mathbf{P} \in \mathbb{R}^{h \times r_C}$
- $\mathbf{C} \in \{0, 1\}^{d \times r_C}$, $\mathbf{C}^T\mathbf{1}_d = \mathbf{1}_r$

For the same ℓ_2 reconstruction error, low-rank is less constrained, hence $r_Q \leq r_C$. But it doesn't remove hidden neurons, which may end up dominating the cost of the approximation. We can thus see two regimes emerge : heavy overparameterization ($r_C \ll d$) where column-sparse approximations will be more efficient since the neuron removal is the bottleneck, and light overparameterization ($r_C \approx d$) where column-sparse approximations will have close to no effect but low-rank approximations may still be able to achieve significant gains. Interesting to note is that in the heavy overparameterization setting, one can apply a column-sparse approximation which will bring the network back to the light overparameterization case, and still apply a low-rank approximation on top of the first approximation to gain even more compression since the neuron count bottleneck has been removed.

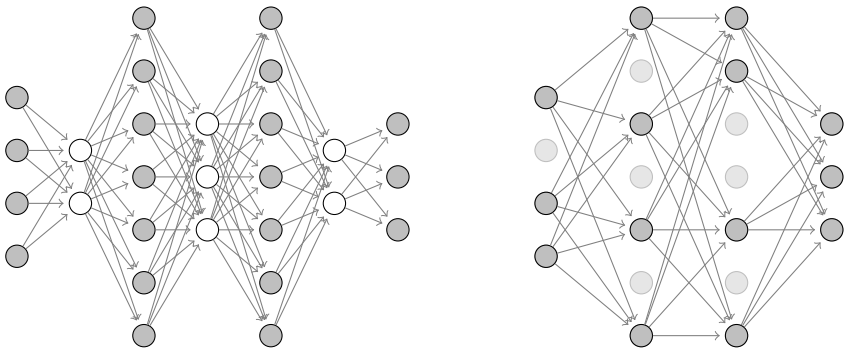

Figure 3: Low-rank vs. Column-sparse shape changes

Although similar to the pruning approximation discussed above, in that this approximation deletes unimportant neurons where pruning deleted weights, this approach does not seem to suffer so much from the pruning drawbacks previously seen. Layers are still fully connected, hence no dangling neurons appear, and redundancies can be leveraged to delete neurons since the following weight matrix is updated to minimize the number of input neurons used to reconstruct the output.

2.4 Solving the linear reconstruction problem

By observing that matrices of the form $\mathbf{P}\mathbf{C}^T$ under the above constraints are exactly column-sparse matrices with r non-zero columns out of n columns, and using the $\ell_{2,1}$ norm as a proxy for their number of non-zero columns in the same way that we had used the nuclear norm as a proxy for the rank, we can consider the following distinct relaxation

$$\min_{\mathbf{M}} \ \mathbb{E}_X \parallel \mathbf{W}X - \mathbf{M}X \parallel_2^2 + \lambda \cdot \|\mathbf{M}\|_{2,1} \tag{4}$$

where $\|\mathbf{M}\|_{2,1} = \sum_j \sqrt{\sum_i M_{i,j}^2}$ is the $\ell_{2,1}$ norm of $\mathbf{M}$, i.e. the sum of the ℓ_2-norms of its columns. The $\ell_{2,1}$ norm has already been shown useful for input subset selection when approximating linear operators in Jain and Haupt [2017].

With the empirical approximation of $\mathcal{D}$ by N samples $(\mathbf{x}_i \in \mathbb{R}^d)_{i<N}$, and rescaling the reconstruction error by a factor 2 without loss of generality to simplify gradients, Problem (4) becomes

$$\min_{\mathbf{M}} \ \frac{1}{2N} \sum_i \parallel \mathbf{W}\mathbf{x}_i - \mathbf{M}\mathbf{x}_i \parallel_2^2 + \lambda \cdot \|\mathbf{M}\|_{2,1} \tag{5}$$

The sum over all $\mathbf{x}_i$ would requires us to go through the whole dataset to evaluate this objective, which would usually be very expensive, so let us move it out of the way. Using $\mathbf{A} = \mathbf{W} - \mathbf{M}$,

$$\mathbb{E}_X \parallel \mathbf{A}X \parallel_2^2 \ = \ \mathbb{E}_X \operatorname{Tr}\left(\mathbf{A} \cdot XX^T \cdot \mathbf{A}^T\right) \ = \ \operatorname{Tr}\left(\mathbf{A} \cdot (\mathbb{E}_X XX^T) \cdot \mathbf{A}^T\right)$$

Thus, we can precompute the uncentered auto-covariance matrix (i.e. the auto-correlation matrix) $\mathbf{R} = \mathbb{E}_X[XX^T]$ (or $\mathbf{R} = \frac{1}{N}\sum_i \mathbf{x}_i\mathbf{x}_i^T$ in the empirical case) of size (d, d) and then evaluate our objective in $\mathcal{O}(hd^2)$, which does not depend on the number of samples in the dataset. This means in particular that once the said matrix has been computed, solving this problem can be done entirely in memory (or on GPU) without any need to perform expensive I/O calls.

Solving with convergence guarantees

This problem is convex, therefore relatively easy to solve. We chose to solve it in the following experiments with a Fast Iterative Shrinkage-Thresholding Algorithm (FISTA) [Beck and Teboulle, 2009], an accelerated proximal gradient method, which achieves a quadratic convergence (proof deferred to the appendix, see section 5.4) to the unique *global* optimum in our setting since the objective is strictly convex.

Algorithm 1 FISTA with fixed step size

input: $\mathbf{X} \in \mathbb{R}^{h\times N}$: input to the layer,
$\mathbf{W} \in \mathbb{R}^{o\times h}$: weight to approximate,
λ : hyperparameter
output: $\mathbf{M} \in \mathbb{R}^{o\times h}$: reconstruction
$\mathbf{R} \leftarrow \mathbf{X}\mathbf{X}^T/N$
$L \leftarrow$ largest eigenvalue of $\mathbf{R}$
$\mathbf{M} \leftarrow 0 \in \mathbb{R}^{o\times h}$, $\mathbf{P} \leftarrow 0 \in \mathbb{R}^{o\times h}$
$t \leftarrow \lambda/L$, $k \leftarrow 1$, $\theta \leftarrow 1$
repeat
$\theta \leftarrow (k-1)/(k+2)$, $k \leftarrow k+1$
$\mathbf{A} \leftarrow \mathbf{M} + \theta(\mathbf{M} - \mathbf{P})$
$d\mathbf{A} \leftarrow (\mathbf{W} - \mathbf{A})\mathbf{R}$
$\mathbf{P} \leftarrow \mathbf{M}$, $\mathbf{M} \leftarrow \mathit{prox}_{t\|\cdot\|_{2,1}}(\mathbf{A} - d\mathbf{A}/L)$
until desired convergence

Algorithm 2 Proximal operator of $\ell_{2,1}$ norm

input: t : scaling factor, $\mathbf{M} \in \mathbb{R}^{o\times h}$
output: $\mathit{prox}_{t\|\cdot\|_{2,1}}(\mathbf{M})$
$r \leftarrow 0 \in \mathbb{R}^h$
for i from 1 to h **do**
if $\|C_i\mathbf{M}\|_2 < t$ **then** $r_i \leftarrow 0$
else $r_i \leftarrow 1 - t \,/\, \|C_i\mathbf{M}\|_2$
return $\mathbf{M} \cdot \operatorname{diag}(r)$

Lemma 2 *Let* $\mathcal{L} : \mathbf{M} \mapsto \frac{1}{2}\mathbb{E}_X \| \mathbf{W}X - \mathbf{M}X \|_2^2 + \lambda \cdot \|\mathbf{M}\|_{2,1}$,
$(\mathbf{M}_k)_k$ *the iterates obtained by FISTA as described above, and* $\mathbf{M}^*$ *the global optimum. Then*

$$\mathcal{L}(\mathbf{M}_k) - \mathcal{L}(\mathbf{M}^*) \;\leq\; \frac{2L}{k^2} \|\mathbf{M}_0 - \mathbf{M}^*\|_F^2$$

where $L = \lambda_{max}(\mathbb{E}_X[XX^T])$ *is the spectral radius of the auto-correlation matrix*

Choosing $\mathbf{M}_0 = 0$, we can get a looser bound with the following observation

$$\|\mathbf{M}^*\|_F^2 \;\leq\; \|\mathbf{M}^*\|_{2,1} \cdot \min\left(\sqrt{d}, \|\mathbf{M}^*\|_{2,1}\right)$$

and by definition of $\mathbf{M}^*$, we have

$$\forall \mathbf{M}, \;\; \|\mathbf{M}^*\|_{2,1} \leq \frac{1}{\lambda}\mathcal{L}(\mathbf{M})$$

which gives a nice way to obtain a bound on the absolute error on the objective which is refined as we execute the algorithm and get lower values of $\mathcal{L}(\mathbf{M})$, usable for a stopping condition for instance.

Alternatively, since the quadratic convergence bound's constant depends on the initial distance to the optimum, we derive in the appendix an alternate minimization algorithm giving a better initial point in reasonable time, see section

Tackling Lasso bias

The Lasso regularization is known to induce a bias because of its shrinkage effect. To tackle this issue, instead of solving Problem (5) as-is to obtain $\mathbf{P}$ and $\mathbf{C}^T$, we only retain the sparse support (i.e. $\mathbf{C}$), and solve again with fixed $\mathbf{C}$ but without the $\ell_{2,1}$ regularizer to obtain the weight values $\mathbf{P}$, which is a simple linear regression. We report results obtained with this additional step under the name "debiased".

2.5 Extension to convolutional layers

The $\ell_{2,1}$ regularization is an instance of the more general Group Lasso regularization, that has been studied for its ability to induce structured sparsity when applied during the training [Alvarez and Salzmann, 2016, Wen et al., 2016, Scardapane et al., 2017]. These works use groups corresponding to the parameters of a neuron, and drop a neuron when all its parameters reach zero. Rather than grouping parameters corresponding to the same output in this fashion (rows in our fully-connected setting) and dropping output neurons whose parameters are all zero, we grouped parameters corresponding to the same *input* (i.e. columns) and dropped the input neurons whose output was not used by the next layer.

We can similarly extend our approach to all types of layers by applying a group lasso regularization by groups of inputs in place of the $\ell_{2,1}$. For convolutional layers for instance, one may group parameters corresponding to the same input channel, resulting in a channel pruning algorithm that drops entire channels that are not mandatory to accurately reconstruct the layer's output.

Note that our previous algorithm can be immediately extended to this case by interpreting the convolution as a matrix multiplication on a flattened input. For each output position (u, v) in output channel j, we write $\mathbf{X}_i^{(u,v)}$ the associated input, that will be multiplied by $\mathbf{W}_j$ to get $(\mathbf{W} * \mathbf{X}_i)_{j,u,v}$. Also note that this rewriting holds for any stride, padding or dilation values.

$$\|\mathbf{W} * \mathbf{X}_i\|_2^2 \;=\; \sum_j \sum_{u,v} \left\| \mathbf{W}_j \odot \mathbf{X}_i^{(u,v)} \right\|_2^2 \quad \text{hence} \quad \mathbf{R} \propto \sum_i \sum_{u,v} \text{vec}(\mathbf{X}_i^{(u,v)}) \cdot \text{vec}(\mathbf{X}_i^{(u,v)})^T$$

2.6 Extension to residual layers

Let us first note that the reconstruction presented above merely replaces a weight matrix with an approximated weight matrix of the same size, only with many more zeros. This enables its use in any case, regardless of the structure of the network. When writing the network weights to disk, any reasonable encoder will be able to leverage this structure to reduce the memory footprint of the encoded network.

However, the removal of unused input neurons cannot be used as-is on residual layers, because more than one layer may depend on the same input. The key to dealing with this removal is simply to view $\mathbf{C}^T$ in the described decomposition $\mathbf{M} = \mathbf{P}\mathbf{C}^T$ as an input sampler (channel sampler for convolutions) that comes before the shrunken layer. If the input is used only once, then the sampler can be applied *statically* (i.e. once, on the network weights) which will result in a smaller previous weight matrix $\mathbf{C}^T\mathbf{W}_1$ as seen previously. On the other hand, if the input is used several times by incompatible samplers, then they can still be applied *dynamically* (i.e. on every inference) which will result in a smaller speedup, since all inputs still have to be computed, but will we compatible with residual and recurrent layers for instance.

2.7 Chaining reconstructions

We presented a way to approximate a single layer with weights $\mathbf{W}$ on input X. In practice though, one may want to use this technique to approximate each layer of a whole network. When performing several reconstructions, different cascading strategies are applicable. In order to compare them, let us first extend Problem (5) to reconstructing an arbitrary output

$$\min_{\mathbf{M}} \ \frac{1}{2N} \sum_i \| \mathbf{y}_i - \mathbf{M}\mathbf{x}_i \|_2^2 + \lambda \cdot \|\mathbf{M}\|_{2,1} \tag{6}$$

FISTA is adapted to this problem by simply changing the gradient of the reconstruction error descent step, i.e. $d\mathbf{A} = \mathbf{Y}\mathbf{X}^T - \mathbf{A}\mathbf{X}\mathbf{X}^T$, where $\mathbf{Y}\mathbf{X}^T$ can again be precomputed before solving the reconstruction, as was the case with $\mathbf{R}$.

Now consider a feed-forward fully connected network (sufficient since we have shown how to extend it to other cases) with weights $(\mathbf{W}_k)_k$ and non-linearities $(\sigma_k)_k$ computing the sequence of activations $(Z_k)_k$, where Z_0 is the input to the network, and $Z_{k+1} = \sigma_k(\mathbf{W}_k Z_k)$

Approximating the weight $\mathbf{W}_k$ can be done with the "true" input $X = Z_k$ and "true" output $Y = \mathbf{W}_k Z_k$, which corresponds to the setting previously described. Since this approximation is independant from approximations of other layers, we label it *parallel*. However, the input that the approximated k-th layer will receive at inference time is not really Z_k, since the layers preceding it will have been approximated as well. We may thus wish to approximate layers sequentially, with the true output $Y = \mathbf{W}_k Z_k$ but with the approximated input $X = A_k$, defined by $A_{k+1} = \sigma_k(\mathbf{W}_k A_k)$ and $A_0 = Z_0$. We refer to this sequential forward chaining as *top-down*[3], and hope that it will limit the accumulation of reconstruction error. Alternatively, since the approximation is discarding neurons, we may take advantage of this and perform approximations sequentially in the reverse order, to avoid reconstructing neurons that will be discarded, which would uselessly waste our reconstruction budget. This corresponds to $Y = \mathbf{C}_{k+1}^T \mathbf{W}_k Z_k$ and $X = Z_k$, which we will refer to as *bottom-up*.

[3]for consistency with the naming "deep" networks, *top* refering to the input and *bottom* to the output

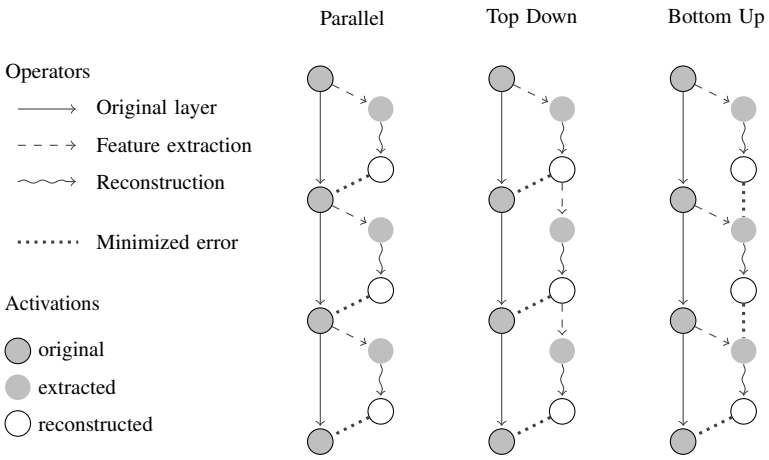

Figure 4: Strategies for chaining reconstructions

3 Experiments

3.1 Fair evaluation of linear reconstruction

We have presented our approach as a method to compress neural networks. However, it has a fundamental difference with other state-of-the-art methods : it does not introduce assumptions on the network. Weight pruning for instance, renders the weight matrices sparse, and this assumption can be leveraged at the time of encoding the network to a bitstream by using a compressed sparse row encoding for instance, which will be particularly effective on very sparse matrices. Another common approach has been quantization of weights, which enables efficient dictionnary-coding approaches to curb the memory footprint. If the assumptions introduced by two compressions don't interact too much with each other, then the two can be composed: one could for instance prune weights, and then quantize the remaining weights, as was proposed in Han et al. [2015a]. It is not possible however to compose two distinct quantization approaches.

The method described here is a little special, in that it does not introduce such assumptions. The "compressed" network obtained in this way is not different from the initial network, it just has fewer redundancies. This means in particular that any compression that can be applied on the initial network can also be applied on the compressed form. This approach is thus not really a compression in itself, but more a pre-processing that can be applied before any other compression.

While comparisons between two quantization methods or two entropy-coding methods seem fair, comparing this to a quantization for instance would appear unfair because the composition with other methods would not be taken into account. It would be more relevant to compare a compression to this pre-processing step followed by a low-rank approximation, followed itself by a weight pruning, a weight quantization, and an entropy coding before measuring the final size. Ideally, we would compare for all known compressions the result with and without this redundancy-removing pre-processing step. Such experiments would quickly become intractable, hence for the sake of this report, we will only compare to the initial network, and see how much reduction we can get *before* compression. We report sizes as simply the "raw" encoding of weight matrices, spending 32 bits per floating-point number, just to give an idea of the orders of magnitude involved. Such an encoding does not reflect real storage situations, but does match the size of the network after decoding, i.e. its size in memory when ready to use, which is still a valuable insight.

3.2 General results

We evaluate our approach on three networks : two relatively small on the MNIST handwritten digit recognition dataset (60k training samples, 10k validation samples), and one on the larger ImageNet (ILSVRC 2012) image classification dataset (1.2M training samples, 50k validation samples). LeNet-300-100 is a three-layer fully connected network with respectively 300 and 100 hidden units. LeNet-5 is a convolutional network with two convolutional layers and two fully connected layers. AlexNet is a much larger 8-layer convolutional network whose last three layers are fully connected.

We apply channel pruning as described in Section 2.5 to layers following a convolutional layer, and neuron pruning to layers following a fully connected layer.

We report the network accuracies and sizes in Table 1. The "retrained" type corresponds to one compression pass, followed by one retraining pass, the number of epochs (each training sample seen once) is reported between parentheses. We only investigate the very limited retraining scenario, but if more computational budget is available, one can naturally get even better compression rates by allocating more budget to the retraining.

Table 1: Compression results

Dataset	Network		Error		Comp. rate	Size
	Architecture	Type	Top-1	Top-5		
MNIST	LeNet-300-100	Baseline	1.68 %	-	-	1.02 MiB
		Compressed	1.71 %	-	46 %	482 KiB
		Retrained (1)	1.64 %	-	29 %	307 KiB
	LeNet-5 (Caffe)	Baseline	0.74 %	-	-	1.64 MiB
		Compressed	0.78 %	-	16 %	276 KiB
		Retrained (1)	0.78 %	-	10 %	177 KiB
ImageNet	AlexNet	Baseline	43.48 %	20.93 %	-	234 MiB
		Compressed	45.36 %	21.90 %	39 %	91 MiB

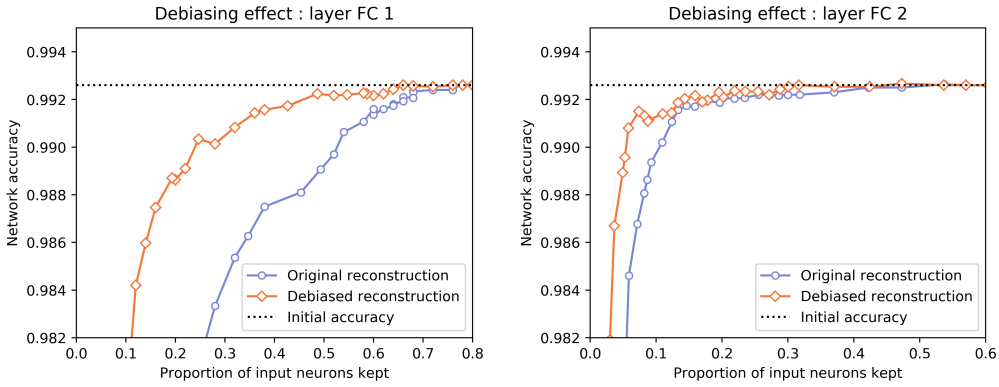

Figure 5: Influence of debiasing on reconstruction quality (LeNet-5 Caffe)

3.3 Reconstruction chaining and debiasing

We first check that the debiasing step we suggested is not performed in vain, by compressing a single layer and reporting the accuracy as a function of the number of input neurons kept for this layer (Figure 5). As expected, this step significantly improves the reconstruction quality, particularly for the lowest compression rates (high values of λ, hence heavily biased solutions).

To compare the effects of reconstruction chaining strategies, we compress a network with each set of hyperparameters from a large grid for each chaining strategy, and plot only the test accuracy as a function of the full network compress rate (higher and to the left is better).

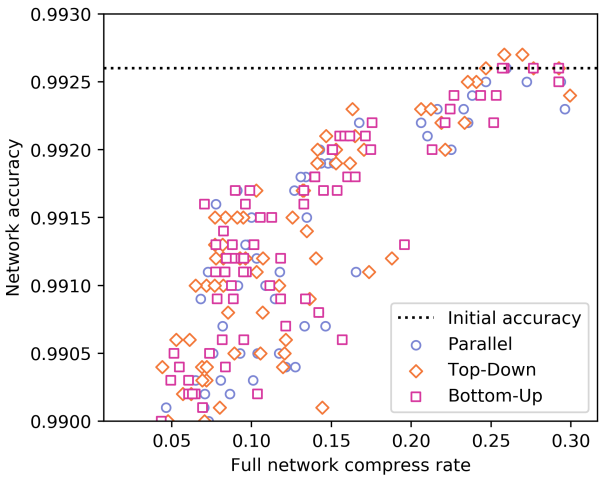

Figure 6: Performances of reconstruction chaining strategies (LeNet-5 Caffe)

Surprisingly, the effect of reconstruction chaining strategies seems rather limited. This is particularly unexpected for the top-down strategy, which we expected to perform much better, particularly on the low compress rates, since it should be able to curb the effects of error accumulation. Several interpretations are possible for this result. First, and most likely, LeNet-5 is only 4 layers deep, which may not be enough to get a significant gain from limiting error accumulation. This experiment will need to be performed on deeper networks to check this properly. Secondly, it is possible that since the accumulation of error is only minimized on the training set, we do not see a significant enough difference on the test set, i.e. the chaining only overfits the training set and does not really limit reconstruction errors in the general case. We will need to measure the accuracy on the training set as well to check whether this intuition is correct.

If further experiments confirm however that the different strategies have comparable performances, then this means that there is no need to perform approximations in a serial manner, and we may compress networks even faster by parallelizing the compression of layers, since they can be performed independently and give similar results.

4 Conclusion

We presented an efficient pre-compression procedure that is able to reduce the memory footprint of a network without significantly affecting its accuracy, in a way that remains composable with all other network compression techniques. By focusing on the reconstruction of the linear activations of layers, we were able to cast the network approximation problem to a series of convex problems that can be solved efficiently. We have demonstrated that this method can be applied in parallel and requires little use of the training data compared to the initial training process, enabling an even faster computation of the approximated models. This makes it particularly well suited to use cases requiring a large number of models of different sizes, where one can train only one model and then compress it many times to each desired size in a reasonable amount of time, instead of having to fine-tune each of the compressed models, as was required by previous inexpensive compression methods recovering from accuracy drops rather than preserving the accuracy of the networks.

References

Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*, 2018.

Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks. In *Advances in Neural Information Processing Systems*, pages 2270–2278, 2016.

Francis R. Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with sparsity-inducing penalties. *CoRR*, abs/1108.0775, 2011. URL `http://arxiv.org/abs/1108.0775`.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

A Chambolle, Ch Dossal, et al. How to make sure the iterates of fista converge. 2014.

Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015b.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 245–249. IEEE, 2014.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Swayambhoo Jain and Jarvis Haupt. Convolutional approximations to linear dimensionality reduction operators. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5885–5889. IEEE, 2017.

Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJlnB3C5Ym`.

Zelda Mariet and Suvrit Sra. Diversity networks: Neural network compression using determinantal point processes. *arXiv preprint arXiv:1511.05077*, 2015.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241:81–89, 2017.

Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in neural information processing systems*, pages 2074–2082, 2016.

5 Appendix

5.1 Proof of optimal brain surgeon step

Recall that we use the approximation of $\mathcal{L}(\mathbf{w}+\delta\mathbf{w})$ by

$$\mathcal{L}(\mathbf{w}+\delta\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \nabla\mathcal{L}(\mathbf{w})^T \cdot \delta\mathbf{w} + \frac{1}{2}\delta\mathbf{w}^T \cdot \nabla^2\mathcal{L}(\mathbf{w}) \cdot \delta\mathbf{w} + \mathcal{O}(\|\delta\mathbf{w}\|^3)$$

and assume $\nabla\mathcal{L}(\mathbf{w}) = 0$ since the network was trained. We wish to minimize $\mathcal{L}(\mathbf{w}+\delta\mathbf{w})$, under the constraint $\delta\mathbf{w}^T\mathbf{e}_q + w_q = 0$. We thus solve instead

$$\min_{\delta\mathbf{w}^T\mathbf{e}_q+w_q=0} \frac{1}{2}\delta\mathbf{w}^T \cdot \nabla^2\mathcal{L}(\mathbf{w}) \cdot \delta\mathbf{w}$$

The gradient of the Lagrangian of this problem is cancelled when

$$\nabla^2\mathcal{L}(\mathbf{w}) \cdot \delta\mathbf{w} + \lambda\mathbf{e}_q = 0$$

which yields $\delta\mathbf{w} = -\lambda\nabla^2\mathcal{L}(\mathbf{w})^{-1}\mathbf{e}_q$, that we can inject into the Lagragian to get

$$\sup_\lambda \quad \frac{1}{2}(\lambda\nabla^2\mathcal{L}(\mathbf{w})^{-1}\mathbf{e}_q)^T \cdot \nabla^2\mathcal{L}(\mathbf{w}) \cdot (\lambda\nabla^2\mathcal{L}(\mathbf{w})^{-1}\mathbf{e}_q) - \lambda\left((\lambda\nabla^2\mathcal{L}(\mathbf{w})^{-1}\mathbf{e}_q)^T\mathbf{e}_q + w_q\right)$$

which rewrites to $\sup_\lambda -\frac{1}{2}\lambda^2(\nabla^2\mathcal{L}(\mathbf{w})^{-1})_{q,q} + \lambda w_q$, attained in $\lambda = \frac{w_q}{(\nabla^2\mathcal{L}(\mathbf{w})^{-1})_{q,q}}$, hence

$$\delta\mathbf{w} = -\frac{w_q}{(\nabla^2\mathcal{L}(\mathbf{w})^{-1})_{q,q}}\nabla^2\mathcal{L}(\mathbf{w})^{-1}e_q \qquad \mathcal{L}(\mathbf{w}+\delta\mathbf{w}) - \mathcal{L}(\mathbf{w}) = \frac{1}{2}\frac{w_q^2}{(\nabla^2\mathcal{L}(\mathbf{w})^{-1})_{q,q}}$$

5.2 Compressed Sparse Row sparse storage cost

Consider a matrix of size (n, n) with a non-zero coefficients. If a is small enough, the most efficient way to encode the matrix would be as a dictionnary of keys, which for a matrix with f bits of precision, will cost $a \cdot (2\log_2(n) + f)$. This cost is only logarithmic in the size of the matrix, but does not allow efficient matrix multiplication.

Therefore, a more popular format chosen to store sparse weight matrices recently has been Compressed Sparse Row (CSR, or "Yale" format). It consists of, for each row, of the number of non-zero elements in it, and a list of their indices, along with the value. This gives a storage cost of $a \cdot (\log_2(n) + f) + n\log_2(n)$. That is to say n numbers, one for each row, plus a indices in total, and a matrix values.

5.3 Proof of Lemma 1 (commutation of $\mathbf{C}$ and σ)

Let $\mathbf{C} \in \mathbb{R}^{m \times n}$ be a linear operator whose action commutes with the pointwise ReLU non-linearity $\sigma : x \mapsto \max(0, x)$. Without loss of generality, we can assume that $\mathbf{C} = \mathbf{c}^T$ with $\mathbf{c} \in \mathbb{R}^n$ (i.e. $\mathbf{C}$ has only one row), because the non-linearity is applied pointwise. The desired property is

$$\forall \mathbf{w} \in \mathbb{R}^n, \quad \mathbf{c}^T \sigma(\mathbf{w}) = \sigma(\mathbf{c}^T \mathbf{w})$$

In particular for $\mathbf{w} = \mathbf{e}_k = (\mathbb{1}(i = k))_i$ (a vector of zeros with a one in position k),

$$c_k = \mathbf{c}^T \sigma(\mathbf{w}) = \sigma(\mathbf{c}^T \mathbf{w}) = \max(0, c_k)$$

so $\forall k,\ c_k \geq 0$. Moreover, for $\mathbf{w} = \mathbf{e}_i - \mathbf{e}_j$ with $i \neq j$,

$$c_i = \mathbf{c}^T \sigma(\mathbf{e}_i - \mathbf{e}_j) = \sigma(\mathbf{c}^T (\mathbf{e}_i - \mathbf{e}_j)) = \max(0,\ c_i - c_j) \quad \Rightarrow \quad c_i = 0 \text{ or } c_j = 0$$

Up to row rescaling (that can be interpreted as part of the reconstructor $\mathbf{P}$) and elimination of trivial rows containing only zeros, this implies that $\mathbf{C}$ is a feature selector.

5.4 Proof of Lemma 2 (convergence rate of FISTA)

The idea for this proof is borrowed from Chambolle et al. [2014].

Let $\mathcal{L} : \mathbf{M} \mapsto \frac{1}{2} \cdot \mathbb{E}_X \parallel \mathbf{W}X - \mathbf{M}X \parallel_2^2 + \lambda \cdot \|\mathbf{M}\|_{2,1}$, and $\mathbf{M}^*$ the global optimum of $\mathcal{L}$.

Let $g : \mathbf{M} \mapsto \frac{1}{2} \cdot \mathbb{E}_X \parallel \mathbf{W}X - \mathbf{M}X \parallel_2^2 = \mathrm{Tr}((\mathbf{W} - \mathbf{M})\, \mathbf{R}\, (\mathbf{W} - \mathbf{M})^T)$
and $h : \mathbf{M} \mapsto \lambda \cdot \|\mathbf{M}\|_{2,1}$, such that $\mathcal{L} = g + h$

Define $L = \lambda_{max}(\mathbf{R})$ the spectral radius of the auto-correlation matrix. We choose a fixed step-size $\gamma \leq \frac{1}{L}$ and a non-negative sequence $(t_n)_n$ of real numbers greater than 1 satisfying $t_{k+1}^2 - t_{k+1} \leq t_k^2$, to define the iterates of FISTA:

$$\begin{aligned} \mathbf{A}_k &= \mathbf{M}_k + \frac{t_k - 1}{t_{k+1}} \cdot (\mathbf{M}_k - \mathbf{M}_{k-1}) \\ \mathbf{M}_{k+1} &= \mathrm{prox}_{\gamma h}\left(\mathbf{A}_k - \gamma \cdot \nabla g(\mathbf{A}_k)\right) \end{aligned}$$

By definition of the proximal operator, $\mathbf{M}_{k+1}$ minimizes

$$\mathbf{U} \mapsto \gamma \cdot h(\mathbf{U}) + \frac{1}{2} \|\mathbf{A}_k - \gamma \cdot \nabla g(\mathbf{A}_k) - \mathbf{U}\|_F^2$$

Expanding the square norm and removing the constant, this is equivalent to minimizing

$$\mathbf{U} \mapsto \gamma \cdot h(\mathbf{U}) + \gamma \cdot \langle \nabla g(\mathbf{A}_k), \mathbf{U} - \mathbf{A}_k \rangle + \frac{1}{2} \|\mathbf{A}_k - \mathbf{U}\|_F^2$$

Dividing by the positive step-size and adding a constant $g(\mathbf{A}_k)$,
this means that $\mathbf{M}_{k+1}$ minimizes the $\frac{1}{\gamma}$-strongly convex function

$$\mathbf{U} \mapsto h(\mathbf{U}) + g(\mathbf{A}_k) + \langle \nabla g(\mathbf{A}_k), \mathbf{U} - \mathbf{A}_k \rangle + \frac{1}{2\gamma} \|\mathbf{A}_k - \mathbf{U}\|_F^2$$

Hence

$$\begin{aligned} \forall \mathbf{U},\ h(\mathbf{M_{k+1}}) + g(\mathbf{A}_k) + \langle \nabla g(\mathbf{A}_k), \mathbf{M_{k+1}} - \mathbf{A}_k \rangle + \frac{1}{2\gamma} \|\mathbf{A}_k - \mathbf{M_{k+1}}\|_F^2 + \frac{1}{2\gamma} \|\mathbf{U} - \mathbf{M_{k+1}}\|_F^2 \\ \leq h(\mathbf{U}) + g(\mathbf{A}_k) + \langle \nabla g(\mathbf{A}_k), \mathbf{U} - \mathbf{A}_k \rangle + \frac{1}{2\gamma} \|\mathbf{A}_k - \mathbf{U}\|_F^2 \end{aligned}$$

Using on the left $g(y) \leq g(x) + \langle \nabla g(x), y - x \rangle + \frac{L}{2} \|y - x\|^2$,
and on the right $g(x) + \langle \nabla g(x), y - x \rangle \leq g(y)$ by convexity, we get

$$\forall \mathbf{U},\ \mathcal{L}(\mathbf{M_{k+1}}) + \frac{1}{2\gamma} \|\mathbf{U} - \mathbf{M}_{k+1}\|_F^2 \leq \mathcal{L}(\mathbf{U}) + \frac{1}{2\gamma} \|\mathbf{A}_k - \mathbf{U}\|_F^2 \tag{7}$$

We will apply this result to $\mathbf{U} = (1 - \frac{1}{t_{k+1}})\mathbf{M}_k + \frac{1}{t_{k+1}}\mathbf{M}^*$

For clarity, let us apply this separately to each occurence of $\mathbf{U}$,
and write $\mathbf{V}_{k+1} = \mathbf{M}_k + t_{k+1}(\mathbf{M}_{k+1} - \mathbf{M}_k)$

$$\|\mathbf{U} - \mathbf{M_{k+1}}\|_F^2 = \left\|(1 - \frac{1}{t_{k+1}})\mathbf{M}_k + \frac{1}{t_{k+1}}\mathbf{M}^* - \mathbf{M_{k+1}}\right\|_F^2 = \left\|\frac{1}{t_{k+1}}\mathbf{M}^* - \frac{1}{t_{k+1}}\mathbf{V}_{k+1}\right\|_F^2$$

$$\|\mathbf{A}_k - \mathbf{U}\|_F^2 = \left\|\mathbf{M}_k + \frac{t_k - 1}{t_{k+1}}(\mathbf{M}_k - \mathbf{M}_{k-1}) - \mathbf{U}\right\|_F^2 = \left\|\frac{1}{t_{k+1}}\mathbf{V}_k - \frac{1}{t_{k+1}}\mathbf{M}^*\right\|_F^2$$

$$\mathcal{L}(\mathbf{U}) \leq (1 - \frac{1}{t_{k+1}})\mathcal{L}(\mathbf{M}_k) + \frac{1}{t_{k+1}}\mathcal{L}(\mathbf{M}^*)$$

Equation 7 with this $\mathbf{U}$ thus gives

$$\mathcal{L}(\mathbf{M_{k+1}}) + \frac{1}{2\gamma t_{k+1}^2} \|\mathbf{V}_{k+1} - \mathbf{M}^*\|_F^2 \leq (1 - \frac{1}{t_{k+1}})\mathcal{L}(\mathbf{M}_k) + \frac{1}{t_{k+1}}\mathcal{L}(\mathbf{M}^*) + \frac{1}{2\gamma t_{k+1}^2} \|\mathbf{V}_k - \mathbf{M}^*\|_F^2$$

With $\delta_k = \mathcal{L}(\mathbf{M}_k) - \mathcal{L}(\mathbf{M}^*)$, this can be written

$$t_{k+1}^2 \cdot \delta_{k+1} - (t_{k+1}^2 - t_{k+1}) \cdot \delta_k \leq \frac{1}{2\gamma} \|\mathbf{V}_k - \mathbf{M}^*\|_F^2 - \frac{1}{2\gamma} \|\mathbf{V}_{k+1} - \mathbf{M}^*\|_F^2$$

Provided we have $t_{k+1}^2 - t_{k+1} \leq t_k^2$, which is verified for $t_k = \frac{k+1}{2}$, and summing from 0 to k

$$t_{k+1}^2 \cdot \delta_{k+1} \leq \frac{1}{2\gamma} \|\mathbf{V}_0 - \mathbf{M}^*\|_F^2 - \frac{1}{2\gamma} \|\mathbf{V}_{k+1} - \mathbf{M}^*\|_F^2$$

And since $\mathbf{V}_0 = \mathbf{M}_0$, this proves the quadratic convergence of the FISTA iterates:

$$\mathcal{L}(\mathbf{M}_k) - \mathcal{L}(\mathbf{M}^*) \leq \frac{L}{2\gamma t_k^2} \|\mathbf{M}_0 - \mathbf{M}^*\|_F^2$$

5.5 Better initial points: Alternate minimization

We present here an alternative algorithm to solve the linear neural reconstruction problem. Although it does not benefit from the guaranteed quadratic convergence that we have proved for FISTA, we have found it very useful in practice since it converges very fast for relatively small values of d, which are common in recent convolutional architectures. The cost of each iteration is $\mathcal{O}(d^3 + hd^2)$, which can be more expensive than FISTA's $\mathcal{O}(hd^2)$, but not a problem in practice since d is generally less than a thousand, and the whole algorithm is very easy to implement.

Recall that we wish to solve

$$\min_{\mathbf{M}} \frac{1}{2} \operatorname{Tr}\left((\mathbf{W} - \mathbf{M})\mathbf{R}(\mathbf{W} - \mathbf{M})^T\right) + \lambda \cdot \|\mathbf{M}\|_{2,1}$$

We can use the variational formulation of the ℓ_1 norm

$$\forall w \in \mathbb{R},\ |w| = \min_{\eta>0} \frac{w^2}{2\eta} + \frac{\eta}{2}$$

to rewrite the penalty as a minimization problem

$$\|\mathbf{M}\|_{2,1} = \sum_j \left| \sqrt{\sum_i M_{i,j}^2} \right| = \min_{(\eta_j>0)_j} \sum_j \frac{\sum_i M_{i,j}^2}{2\eta_j} + \frac{\eta_j}{2}$$

and thus solve the following equivalent problem,

$$\min_{\mathbf{M},\, \eta>0} \operatorname{Tr}\left((\mathbf{W}-\mathbf{M})\mathbf{R}(\mathbf{W}-\mathbf{M})^T + \lambda \cdot \mathbf{M} D(\eta^{-1})\mathbf{M}^T\right) + \lambda \cdot \eta^T \mathbf{1} \tag{8}$$

By adding a smoothing term $\varepsilon \sum_j \eta_j^{-1}$ with any $\varepsilon > 0$, we can then make the level sets with respect to η compact on the positive quadrant, ensuring that alternate minimization algorithms are convergent [Bach et al., 2011]. Because it is strictly convex, this problem can then be solved to *global* optimum by alternate minimization. The optimum with respect to η is given in closed form by $\eta_j^2 = \sum_i M_{i,j}^2 + \varepsilon$, and the optimum with respect to $\mathbf{M}$ is obtained by canceling the gradient in $\mathbf{M}$, i.e. solving the linear system $(\mathbf{R} + \lambda \cdot D(\eta^{-1}))\mathbf{M}^T = 2\mathbf{R}\mathbf{W}^T$.

The explicit algorithm to solve the linear neural reconstruction problem in the general case is reproduced here for convenience. It is assumed that the h inputs are divided evenly into g groups of s elements for simplicity, such that $(X_{sj+k,i})_{k<s}$ constitutes the j-th group of the i-th input.

Algorithm 3 Alternate minimization for linear neural reconstruction

input: $\mathbf{X} \in \mathbb{R}^{h\times N}$: input to the layer
$\mathbf{Y} \in \mathbb{R}^{o\times N}$: output to reconstruct
$g \in \mathbb{N}$: number of input groups
output: $\mathbf{M} \in \mathbb{R}^{o\times h}$: reconstruction
$\mathbf{R} \leftarrow \mathbf{X}\mathbf{X}^T/N,\ \mathbf{B} \leftarrow 2\mathbf{X}\mathbf{Y}^T/N,\ \mathbf{M} \leftarrow 0 \in \mathbb{R}^{o\times h},\ s \leftarrow h/g$
repeat
$\eta_{sj+k} \leftarrow \sqrt{\sum_{i,u} \mathbf{M}_{i,sj+u}^2 + \varepsilon}, \quad \forall j < g,\ \forall k < s$
$\mathbf{M} \leftarrow \text{solve}(\mathbf{R} + \lambda \cdot D(\eta^{-1}),\ \mathbf{B})^T$
until desired convergence