SIAM J. MATH. DATA SCI.
Vol. 2, No. 1, pp. 158–188



# Fast Convex Pruning of Deep Neural Networks*

Alireza Aghasi†, Afshin Abdi‡, and Justin Romberg‡

**Abstract.** We develop a fast, tractable technique called Net-Trim for simplifying a trained neural network. The method is a convex postprocessing module, which prunes (sparsifies) a trained network layer by layer, while preserving the internal responses. We present a comprehensive analysis of Net-Trim from both the algorithmic and sample complexity standpoints, centered on a fast, scalable convex optimization program. Our analysis includes consistency results between the initial and retrained models before and after Net-Trim application and provides a sample complexity bound on the number of input samples needed to discover a network layer with sparse topology. Specifically, if there is a set of weights that uses at most $s$ terms that can recreate the layer outputs from the layer inputs, we can find these weights from $\mathcal{O}(s \log N/s)$ samples, where $N$ is the input size. These theoretical results are similar to those for sparse regression using the Lasso, and our analysis uses some of the same recently developed tools (namely recent results on the concentration of measure and convex analysis). Finally, we propose an algorithmic framework based on the alternating direction method of multipliers (ADMM), which allows a fast and simple implementation of Net-Trim for network pruning and compression.

**Key words.** network pruning, deep neural networks, compressed sensing, bowling scheme, Rademacher complexity

**AMS subject classifications.** 90C25, 15B52, 68Q25, 62M45

**DOI.** 10.1137/19M1246468

## 1. Introduction.

Deep neural networks are becoming a prominent tool to learn data structures of arbitrary complexity. This success is mainly thanks to their flexible, yet compact, nonlinear formulation, and the development of computational and architectural techniques to improve their training (cf. [8, 21] for a comprehensive review). Increasing the number of layers and the number of neurons within each layer is generally the most standard way of adding more flexibility to a neural network. While adding this flexibility can improve the fit of the model to the training data, it also adds complexity. This complexity can be mitigated by sparsifying the network, representing it using fewer weights. Sparse networks occupy less memory, are more computationally efficient, and are more stable when there is noise in the input.[1]

To simplify or stabilize neural networks, various regularizing techniques and pruning strategies have been considered. Inspired by the classic regularizers for linear models, such

*Received by the editors February 26, 2019; accepted for publication (in revised form) November 5, 2019; published electronically February 20, 2020. An initial and limited version of this work was presented as a Spotlight in Advances in Neural Information Processing Systems conference, 2017 [2].
https://doi.org/10.1137/19M1246468

†Department of Business Analytics, Georgia State University, Atlanta, GA 30303 (aaghasi@gsu.edu).
‡Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 (abdi@ece.gatech.edu, jrom@ece.gatech.edu).

[1]We include experiments on the empirical stability of sparse networks in the Supplementary Materials.

as Ridge [12] and the Lasso [24], the training of neural networks is also equipped with $\ell_2$ or $\ell_1$ penalties [7, 20] to control their variance and complexity. Adding randomness to the training process is also shown to have regularizing effects, relevant to which we may refer to Dropout [22] and DropConnect [28], which randomly remove active connections in the training phase and are likely to reduce the magnitude of the network weights, or sparsify the activations. Batch normalization [14], associated with stochastic gradient descent–type fitting techniques, can also be considered as a tool of similar nature, where in the training process the updates of the hidden units are weighted by the standard deviation of the random examples included in the mini-batch.

Another common trend to prune deep neural networks is to learn a mask for the weights [9] or consider a Bayesian framework, where inline with a sparsity inducing prior, a posterior distribution over the network weights is learned [16, 18, 19]. These methods are mainly promoted as the tools to compress large neural networks by pruning or quantizing the weights [1]. They normally require a nonconventional training process and often require introducing an augmented set of parameters to be trained along with the network weights. Moreover, the complex and nonconvex nature of the underlying objectives forbids performance guarantees for these methods.

In this paper, we advocate a different approach. We train the network using standard techniques. We then extract the internal outputs (the intermediate features) at each layer and find a sparse set of weights that reproduces these features across all the training data. The philosophy here is that the most important product of training the network is the features that it extracts, not the weights that it settles on to produce those features. For large networks, there will be many sets of weights that produce exactly the same internal features; of those weights, we choose the simplest.

Our method for finding sparse sets of weights, presented in detail in section 3, is related to well-known techniques for sparse regression, e.g., the Lasso [24] in statistics and compressed sensing [5] in signal processing. The main difference is the nonlinearity in the mapping of internal features from one layer to another. If this nonlinearity is piecewise linear and convex (as is the rectified linear unit, $\texttt{ReLU}(\boldsymbol{x}) = \max(\boldsymbol{x}, \boldsymbol{0})$, that we use in all of our analysis below), then there is a natural way to recast the condition that the outputs and inputs of a layer match as a set of linear inequality constraints. There is a similar way to recast an approximate matching as inclusion in a convex set. Using the $\ell_1$ norm as a proxy for sparsity, the entire program becomes convex. This opens the door for a thorough analysis of how well and under what conditions we can expect Net-Trim to perform well, and allows us to leverage decades of research in convex optimization to find a scalable algorithm with predictable convergence behavior.

The theory in section 4 presents an upper bound on the number of training samples needed to discover a weight matrix that is sparse. Given a set of layer input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$ and output vectors $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_P$, we solve the program

$$\underset{\boldsymbol{W}}{\text{minimize}} \ \|\boldsymbol{W}\|_1 \quad \text{subject to} \quad \texttt{ReLU}(\boldsymbol{W}^\top \boldsymbol{x}_p) = \boldsymbol{y}_p \quad (p = 1, \ldots, P), \tag{1.1}$$

where $\|\boldsymbol{W}\|_1 = \sum_{n,m} |w_{n,m}|$ is the sum of the absolute values of the entries in a matrix $\boldsymbol{W} \in \mathbb{R}^{N \times M}$. As the $\ell_1$ norm is convex and the $\texttt{ReLU}(\cdot)$ function is piecewise linear, meaning

that constraints in the program above can be broken into a series of linear equality and inequality constraints, the program above is convex. We show that if the $\boldsymbol{x}_p$ are independent samples of a subgaussian random vector that is nondegenerate (meaning that the correlation matrix is full-rank) and there exists a $\boldsymbol{W}_\star$ with maximally $s$-sparse columns that does indeed satisfy $\boldsymbol{y}_p = \texttt{ReLU}(\boldsymbol{W}_\star^\top \boldsymbol{x}_p)$ for all $p$, then the solution to (1.1) is exactly $\boldsymbol{W}_\star$ when the number of training samples $P$ is (almost) proportional to the sparsity $s$: we require $P \gtrsim s \log(N/s)$.

We also show that if the $\boldsymbol{x}_p$ are subgaussian, then so are the $\boldsymbol{y}_p$. This means that our theoretical result can be applied layer by layer. The equality constraints in (1.1) can also be relaxed, allowing us to trade off sparsity and accuracy (this is discussed in detail in section 3.1). Along with these theoretical guarantees, Net-Trim offers state-of-the-art performance on realistic networks. In section 6, we present some numerical experiments that show that compression factors between 10x and 50x (removing 90% to 98% of the connections) are possible with very little loss in test accuracy.

*Contributions and relations to previous work.* This paper provides a full description of the Net-Trim method from both a theoretical and an algorithmic perspective. In section 3, we present our convex formulation for sparsifying the weights in the linear layers of a network; we describe how the procedure can be applied layer by layer in a deep network either in parallel or serially (cascading the results), and present consistency bounds for both approaches. Section 4 presents our main theoretical result, stated precisely in Theorem 4.4. This result derives an upper bound on the number of data samples we need to reliably discover a layer that has at most $s$ connections in its linear layer—we show that if the data samples are random, then these weights can be learned from $\mathcal{O}(s \log N/s)$ samples. Mathematically, this result is comparable to the sample complexity bounds for the Lasso in performing sparse regression on a linear model (also known as the compressed sensing problem). As presented in section 4.2, our analysis is based on the bowling scheme [17, 25]; the main technical challenges are adapting this technique to the piecewise linear constraints in the program (1.1), and the fact that the input vectors $\{\boldsymbol{x}_p\}$ into each layer are noncentered in a way that cannot be accounted for easily.

There are several other examples of techniques for simplifying networks by retraining in the recent literature. These techniques are typically presented as model compression tools (e.g., [6, 10, 11]) for removing the inherent model redundancies. In what is perhaps the most closely related work to what we present here, [11] proposes a pruning scheme that simply truncates small weights of an already trained network, and then readjusts the remaining active weights using another round of training. In contrast, our optimization scheme ensures that the layer inputs and outputs stay consistent as the network is pruned.

The Net-Trim framework was first presented in [2]. This paper provides a more rigorous analysis under a more general assumption on the input, and shows how results from multiple layers can be combined. The theory presented in this work characterizes the pruning procedure in both $P < N$ and $P > N$ regimes (i.e., underdetermined and overdetermined regimes), while the earlier work merely focused on an underdetermined case. In addition, we present a scalable (yet relatively simple) implementation of Net-Trim using the alternating direction method of multipliers (ADMM). This is an iterative method with each iteration requiring a small number of matrix-vector multiplies. The code, along with all of the examples presented in the paper,

is available online.[2]

*Notation.* We use lowercase and uppercase boldface for vectors and matrices, respectively. Specifically, the notation $\boldsymbol{I}$ is reserved for the identity matrix. For a matrix $\boldsymbol{A}$, $\boldsymbol{A}_{\Gamma_1,:}$ denotes the submatrix formed by restricting the rows of $\boldsymbol{A}$ to the index set $\Gamma_1$. Similarly, $\boldsymbol{A}_{:,\Gamma_2}$ restricts the columns of $\boldsymbol{A}$ to $\Gamma_2$, and $\boldsymbol{A}_{\Gamma_1,\Gamma_2}$ is formed by extracting both rows and columns. Given a vector $\boldsymbol{x}$ (or matrix $\boldsymbol{X}$), supp $\boldsymbol{x}$ (or supp $\boldsymbol{X}$) is the set of indices with nonzero entries, and supp$^c$ $\boldsymbol{x}$ (or supp$^c$ $\boldsymbol{X}$) is the complement set.

For $\boldsymbol{X} = [x_{m,n}] \in \mathbb{R}^{M\times N}$, the matrix trace is denoted by $\mathrm{tr}(\boldsymbol{X})$. Furthermore, we use $\|\boldsymbol{X}\|_1 \triangleq \sum_{m=1}^{M}\sum_{n=1}^{N}|x_{m,n}|$ as a notation for the sum of absolute entries,[3] and $\|\boldsymbol{X}\|_F$ as the Frobenius norm. The neural network activation used throughout the paper is the rectified linear unit (ReLU), which is applied componentwise to vectors and matrices,

$$\left(\mathtt{ReLU}\left(\boldsymbol{X}\right)\right)_{m,n} = \max\left(x_{m,n}, 0\right).$$

We will sometimes use the notation $\boldsymbol{X}^+$ as shorthand for $\mathtt{ReLU}(\boldsymbol{X})$. For an index set $\Omega \subseteq \{1,\dots,M\}\times\{1,\dots,N\}$, $\boldsymbol{W}_\Omega$ represents a matrix of identical size as $\boldsymbol{W} = [w_{m,n}]$ with entries

$$\left(\boldsymbol{W}_\Omega\right)_{m,n} = \begin{cases} w_{m,n} & (m,n)\in\Omega, \\ 0 & (m,n)\notin\Omega. \end{cases}$$

We use $\mathbb{S}^N$ to denote the unit sphere in $\mathbb{R}^{N+1}$, and $\mathbb{B}^N$ to denote the unit Euclidean ball in $\mathbb{R}^N$. Finally, the notation $f \gtrsim \hat{f}$ (or $f \lesssim \hat{f}$) is used when there exists an absolute constant $C$ such that $f \geq C\hat{f}$ (or $f \leq C\hat{f}$).

*Outline.* The remainder of the paper is structured as follows. In section 2, we briefly overview the neural network architecture considered. Section 3 presents the pruning idea and the consistency results between the initial and retrained networks. The statistical architecture of the network and the general sample complexity results are presented in section 4. To implement the Net-Trim underlying convex program, in section 5 we present an ADMM scheme applicable to the original Net-Trim formulation. Finally, section 6 presents some experiments, along with concluding remarks. Other than Theorem 4.4, which is the main result of the paper and is proved in section 4.2, all other technical proofs, along with additional experimental and implementation discussions, are presented in the Supplementary Materials.

## 2. Feedforward network model.

In this section, we briefly overview the topology of the feedforward network model considered. The training of the network is performed via $P$ samples $\boldsymbol{x}_p$, $p = 1,\dots,P$, where $\boldsymbol{x}_p \in \mathbb{R}^N$ is the network input. To compactly represent the training samples, we form a matrix $\boldsymbol{X} \in \mathbb{R}^{N\times P}$, structured as $\boldsymbol{X} = [\boldsymbol{x}_1,\dots,\boldsymbol{x}_P]$. Considering $L$ layers in the network, the output of the network at the final layer is denoted by $\boldsymbol{X}^{(L)} \in \mathbb{R}^{N_L\times P}$, where each column in $\boldsymbol{X}^{(L)}$ is a response to the corresponding training column in $\boldsymbol{X}$.

In a ReLU network, the output of the $\ell$th layer is $\boldsymbol{X}^{(\ell)} \in \mathbb{R}^{N_\ell\times P}$, generated by applying the affine transformation $\boldsymbol{W}_\ell^\top(\cdot) + \boldsymbol{b}^{(\ell)}$ to each column of the previous layer $\boldsymbol{X}^{(\ell-1)}$, followed by a ReLU activation:

$$\boldsymbol{X}^{(\ell)} = \mathtt{ReLU}\left(\boldsymbol{W}_\ell^\top \boldsymbol{X}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\boldsymbol{1}^\top\right), \qquad \ell = 1,\dots,L. \tag{2.1}$$

[2]The link to the code and related material is https://dnntoolbox.github.io/Net-Trim/.

[3]The notation $\|\boldsymbol{X}\|_1$ should not be confused with the matrix induced $\ell_1$ norm.

Here $\boldsymbol{W}_\ell \in \mathbb{R}^{N_{\ell-1}\times N_\ell}$, $\boldsymbol{X}^{(0)} = \boldsymbol{X}$, and $N_0 = N$. By adding an additional row to $\boldsymbol{W}_\ell$ and $\boldsymbol{X}^{(\ell-1)}$, one can absorb the intercept term and compactly rewrite (2.1) as

$$\boldsymbol{X}^{(\ell)} = \texttt{ReLU}\left(\boldsymbol{W}_\ell^\top \boldsymbol{X}^{(\ell-1)}\right), \qquad \ell = 1, \dots, L. \tag{2.2}$$

Often the last layer of a neural network skips an activation by merely going through the affine transformation. As a matter of fact, the results presented in this paper also apply to such architecture (see analysis examples in [2]). A neural network that follows the model in (2.2) can be fully identified by $\boldsymbol{X}$ and $\hat{\boldsymbol{W}}_\ell$, $\ell = 1, \dots, L$. Throughout the paper, such a network will be denoted by $\mathcal{N}et(\{\boldsymbol{W}_\ell\}_{\ell=1}^L; \boldsymbol{X})$.

## 3. The Net-Trim pruning algorithm.

Net-Trim is a postprocessing scheme which prunes a neural network after the training phase. After the training phase and learning $\boldsymbol{W}_\ell$, Net-Trim retrains the network so that for the same training data the layer outcomes stay more or less close to the initial model, while the redesigned network is sparser, i.e.,

$$\ell = 1, \dots, L: \ \operatorname{nnz}\left(\hat{\boldsymbol{W}}_\ell\right) \ll \operatorname{nnz}\left(\boldsymbol{W}_\ell\right), \quad \text{while} \ \ \hat{\boldsymbol{X}}^{(\ell)} \approx \boldsymbol{X}^{(\ell)}.$$

Here, nnz(.) denotes the number of nonzero entries, and $\hat{\boldsymbol{W}}_\ell$ and $\hat{\boldsymbol{X}}^{(\ell)}$ are, respectively, the redesigned layer matrices and the corresponding layer outcomes.

Aside from the postprocessing nature and some differences in the convex formulations, Net-Trim shares many similarities with the Lasso (least absolute shrinkage and selection operator [24]), as they both use an $\ell_1$ proxy to promote model sparsity. In the remainder of this section we overview the Net-Trim formulation and the corresponding pruning schemes.

### 3.1. Pruning a single layer.

Consider $\boldsymbol{X}^{in} \in \mathbb{R}^{N\times P}$ and $\boldsymbol{X}^{out} \in \mathbb{R}^{M\times P}$ to be layer input and output matrices after the training, which based on the model in (2.1) (or (2.2)) are connected via

$$\boldsymbol{X}^{out} = \texttt{ReLU}\left(\boldsymbol{W}^\top \boldsymbol{X}^{in}\right).$$

To explore a sparser coefficient matrix, we may consider the minimization

$$\underset{\boldsymbol{U}}{\text{minimize}} \quad \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \left\|\texttt{ReLU}\left(\boldsymbol{U}^\top \boldsymbol{X}^{in}\right) - \boldsymbol{X}^{out}\right\|_F \le \epsilon, \tag{3.1}$$

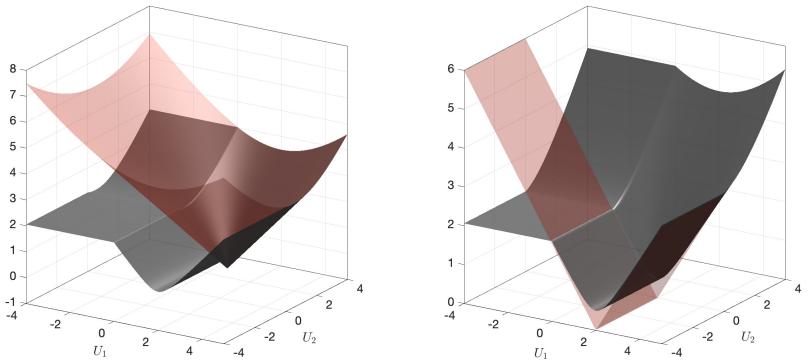
which may potentially generate a sparser $\boldsymbol{W}$-matrix relating $\boldsymbol{X}^{in}$ and $\boldsymbol{X}^{out}$, at the expense of a (controllable) discrepancy between the layer outcomes before and after the retraining.

Despite the convex objective, the constraint set in (3.1) is nonconvex (see Figure 1). Using the fact that the entries of $\boldsymbol{X}^{out}$ are either zero or strictly positive quantities, [2] proposes the following convex proxy to (3.1):

$$\hat{\boldsymbol{W}} = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \begin{cases} \left\|\left(\boldsymbol{U}^\top \boldsymbol{X}^{in} - \boldsymbol{X}^{out}\right)_\Omega\right\|_F \le \epsilon, \\ \left(\boldsymbol{U}^\top \boldsymbol{X}^{in}\right)_{\Omega^c} \le \boldsymbol{0}, \end{cases} \tag{3.2}$$

where

$$\Omega = \operatorname{supp} \boldsymbol{X}^{out} = \left\{(m,p) : \left[\boldsymbol{X}^{out}\right]_{m,p} > 0\right\}.$$




(a) (b)

**Figure 1.** *Plots of the constraint function* $f(\boldsymbol{U}) = \left\|\texttt{ReLU}\left(\boldsymbol{U}^\top \boldsymbol{X}^{in}\right) - \boldsymbol{X}^{out}\right\|_F$ *in solid color, and the convex surrogate* $f_c(\boldsymbol{U}) = \|(\boldsymbol{U}^\top \boldsymbol{X}^{in} - \boldsymbol{X}^{out})_\Omega\|_F$ *in transparent color restricted to the set* $(\boldsymbol{U}^\top \boldsymbol{X}^{in})_{\Omega^c} \leq \boldsymbol{0}$. *For these plots* $\boldsymbol{U} \in \mathbb{R}^2$, $\boldsymbol{X}^{in} = \boldsymbol{I}$, (a) $\boldsymbol{X}^{out} = (2, 0.5)$; (b) $\boldsymbol{X}^{out} = (2, -0.5)$.

The main idea behind this convex surrogate is imposing similar activation patterns before and after the retraining via the second inequality in (3.2), i.e.,

$$\left(\boldsymbol{W}^\top \boldsymbol{X}^{in}\right)_{\Omega^c}^+ = \left(\hat{\boldsymbol{W}}^\top \boldsymbol{X}^{in}\right)_{\Omega^c}^+ = \boldsymbol{0},$$

and allowing the $\epsilon$-discrepancy only on the set $\Omega$. For a more compact presentation of the convex constraint set, for given matrices $\boldsymbol{X}, \boldsymbol{Y}$, and $\boldsymbol{V}$ we use the notation

$$\boldsymbol{U} \in \mathcal{C}_\epsilon\left(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{V}\right) \iff \begin{cases} \left\|\left(\boldsymbol{U}^\top \boldsymbol{X} - \boldsymbol{Y}\right)_\Omega\right\|_F \leq \epsilon, \\ \left(\boldsymbol{U}^\top \boldsymbol{X}\right)_{\Omega^c} \leq \boldsymbol{V}_{\Omega^c} \end{cases} \quad \text{for } \Omega = \operatorname{supp} \boldsymbol{Y}. \tag{3.3}$$

Using this notation, the convex program in (3.2) may be cast as

$$\hat{\boldsymbol{W}} = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_\epsilon\left(\boldsymbol{X}^{in}, \boldsymbol{X}^{out}, \boldsymbol{0}\right). \tag{3.4}$$

**3.2. Pruning the network.** Having access to the tools to retrain any layer within the network, exclusively based on the input and the output, we may consider *parallel* or *cascade* frameworks to retrain the entire network.

The parallel Net-Trim is a straightforward application of the convex program (3.4) to each layer in the network. Basically, each layer is processed independently based on the initial model input and output, without taking into account the retraining result from the previous layers. Specifically, denoting $\boldsymbol{X}^{(\ell-1)}$ and $\boldsymbol{X}^{(\ell)}$ as the input and output of the $\ell$th layer of the initial trained network, we propose retraining the coefficient matrix $\boldsymbol{W}_\ell$ via the convex program

$$\hat{\boldsymbol{W}}_\ell = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_\ell}\left(\boldsymbol{X}^{(\ell-1)}, \boldsymbol{X}^{(\ell)}, \boldsymbol{0}\right), \quad \ell = 1, \ldots, L. \tag{3.5}$$

An immediate question would be, If each layer of a network is retrained via (3.5) and one replaces $\mathcal{N}et(\{\boldsymbol{W}_\ell\}_{\ell=1}^L; \boldsymbol{X})$ with the retrained network $\mathcal{N}et(\{\hat{\boldsymbol{W}}_\ell\}_{\ell=1}^L; \boldsymbol{X})$, how do the discrepancies $\epsilon_\ell$ propagate across the network, and how far apart would be the final responses of the two networks to $\boldsymbol{X}$? The following result addresses this question.

Theorem 3.1 (parallel Net-Trim consistency). *Consider a network* $\mathcal{N}et(\{\boldsymbol{W}_\ell\}_{\ell=1}^L;\boldsymbol{X})$*, which is normalized, i.e.,* $\|\boldsymbol{W}_\ell\|_1 = 1$ *for* $\ell = 1,\dots,L$*. Solve* (3.5) *for each layer and form the retrained network* $\mathcal{N}et(\{\hat{\boldsymbol{W}}_\ell\}_{\ell=1}^L;\boldsymbol{X})$*. Denoting by* $\hat{\boldsymbol{X}}^{(\ell)} = \texttt{ReLU}(\hat{\boldsymbol{W}}_\ell{}^\top \hat{\boldsymbol{X}}^{(\ell-1)})$ *the outcomes of the retrained network, where* $\hat{\boldsymbol{X}}^{(0)} = \boldsymbol{X}^{(0)} = \boldsymbol{X}$*, the layer outcomes of the original and retrained networks obey*

$$\left\|\hat{\boldsymbol{X}}^{(\ell)} - \boldsymbol{X}^{(\ell)}\right\|_F \le \sum_{j=1}^{\ell} \epsilon_j, \quad \ell = 1,\dots,L. \tag{3.6}$$

*Proof.* See section SM1.1 of the Supplementary Materials. ∎

It is noteworthy that the normalization assumption $\|\boldsymbol{W}_\ell\|_1 = 1$ in Theorem 3.1 is made with no loss in generality, and is only a way of presenting the result in a standard form. This is simply because $\texttt{ReLU}(|\alpha|x) = |\alpha|\texttt{ReLU}(x)$, and a scaling of any of the weight matrices $\boldsymbol{W}_\ell$ would scale $\boldsymbol{X}^{(L)}$ (or $\boldsymbol{X}^{(\ell')}$ where $\ell' \ge \ell$) by the same amount. Specifically, the outcomes of the network before and after the process obey

$$\left\|\hat{\boldsymbol{X}}^{(L)} - \boldsymbol{X}^{(L)}\right\|_F \le \sum_{j=1}^{L} \epsilon_j,$$

which makes parallel Net-Trim a stable process, producing a controllable overall discrepancy.

A more adaptive way of retraining a network, which we would refer to as the cascade Net-Trim, incorporates the outcome of the previously pruned layers to retrain a target layer. Basically, in a cascade Net-Trim, retraining $\boldsymbol{W}_\ell$ takes place by exploring a path between the input/output pairs $(\hat{\boldsymbol{X}}^{(\ell-1)}, \boldsymbol{X}^{(\ell)})$ instead of $(\boldsymbol{X}^{(\ell-1)}, \boldsymbol{X}^{(\ell)})$. Due to some feasibility concerns, which will be detailed in what follows, a cascade formulation does not simply happen by replacing $\boldsymbol{X}^{(\ell-1)}$ with $\hat{\boldsymbol{X}}^{(\ell-1)}$ in (3.5), and the formulation requires some modifications.

To derive the cascade formulation, consider starting the process by retraining the first layer via

$$\hat{\boldsymbol{W}}_1 = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_1}\left(\boldsymbol{X}, \boldsymbol{X}^{(1)}, \boldsymbol{0}\right). \tag{3.7}$$

Setting $\hat{\boldsymbol{X}}^{(1)} = \texttt{ReLU}(\hat{\boldsymbol{W}}_1{}^\top \boldsymbol{X})$, to adaptively prune the second layer, one would ideally consider the program

$$\underset{\boldsymbol{U}}{\text{minimize}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_2}\left(\hat{\boldsymbol{X}}^{(1)}, \boldsymbol{X}^{(2)}, \boldsymbol{0}\right). \tag{3.8}$$

It is not hard to see that the simple generalization in (3.8) is not guaranteed to be feasible, that is, there might not exist a matrix $\boldsymbol{W}$ such that for $\Omega = \text{supp}\boldsymbol{X}^{(2)}$,

$$\begin{cases} \left\|\left(\boldsymbol{W}^\top \hat{\boldsymbol{X}}^{(1)} - \boldsymbol{X}^{(2)}\right)_\Omega\right\|_F \le \epsilon_2, \\ \left(\boldsymbol{W}^\top \hat{\boldsymbol{X}}^{(1)}\right)_{\Omega^c} \le \boldsymbol{0}. \end{cases} \tag{3.9}$$

If instead of $\hat{\boldsymbol{X}}^{(1)}$ the constraint set (3.9) was parameterized by $\boldsymbol{X}^{(1)}$, a natural feasible point would have been $\boldsymbol{W} = \boldsymbol{W}_2$. Now that $\hat{\boldsymbol{X}}^{(1)}$ is a perturbed version of $\boldsymbol{X}^{(1)}$, the constraint set

needs to be properly slacked to maintain the feasibility of $\boldsymbol{W}_2$. In this context, one may easily verify that $\boldsymbol{W}_2$ is feasible for the slacked program

$$\underset{\boldsymbol{U}}{\text{minimize}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_2}\left(\hat{\boldsymbol{X}}^{(1)}, \boldsymbol{X}^{(2)}, \boldsymbol{W}_2^\top \hat{\boldsymbol{X}}^{(1)}\right), \tag{3.10}$$

as long as for some $\gamma \geq 1$,

$$\epsilon_2 = \gamma \left\| \left( \boldsymbol{W}_2^\top \hat{\boldsymbol{X}}^{(1)} - \boldsymbol{X}^{(2)} \right)_\Omega \right\|_F.$$

The $\gamma$-coefficient is a free parameter, which we refer to as the *slackness rate*. When $\gamma = 1$, the matrix $\boldsymbol{W}_2$ is only tightly feasible for (3.10) and the feasible set can, at the very least, become a singleton. However, increasing the slackness rate would expand the set of permissible matrices and make (3.10) capable of producing sparser solutions.

The process applied to the second layer may be generalized to the subsequent layers and form a cascade paradigm to prune the network layer by layer. The pseudocode in Algorithm 3.1 summarizes the Net-Trim cascade scheme, where we set $\epsilon_1 = \epsilon$ for the first layer, and consider the slackness rates $\gamma_\ell$, $\ell = 2, \ldots, L$, for the subsequent layers.

**Algorithm 3.1.** Cascade Net-Trim.

$\hat{\boldsymbol{W}}_1 \leftarrow \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_\epsilon\left(\boldsymbol{X}, \boldsymbol{X}^{(1)}, \boldsymbol{0}\right)$

$\hat{\boldsymbol{X}}^{(1)} \leftarrow \texttt{ReLU}\left(\hat{\boldsymbol{W}}_1{}^\top \boldsymbol{X}\right)$

**for** $\ell = 2, \ldots, L$ **do**

$\Omega \leftarrow \text{supp}\boldsymbol{X}^{(\ell)}; \quad \epsilon_\ell \leftarrow \gamma_\ell \left\| \left( \boldsymbol{W}_\ell^\top \hat{\boldsymbol{X}}^{(\ell-1)} - \boldsymbol{X}^{(\ell)} \right)_\Omega \right\|_F$

$\hat{\boldsymbol{W}}_\ell \leftarrow \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad \text{subject to} \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_\ell}\left(\hat{\boldsymbol{X}}^{(\ell-1)}, \boldsymbol{X}^{(\ell)}, \boldsymbol{W}_\ell^\top \hat{\boldsymbol{X}}^{(\ell-1)}\right)$

$\hat{\boldsymbol{X}}^{(\ell)} \leftarrow \texttt{ReLU}\left(\hat{\boldsymbol{W}}_\ell{}^\top \hat{\boldsymbol{X}}^{(\ell-1)}\right)$

**end for**

Similar to the parallel scheme, we can show a bounded discrepancy between the outcomes of the initial network $\mathcal{N}et(\{\boldsymbol{W}_\ell\}_{\ell=1}^L; \boldsymbol{X})$ and the retrained network $\mathcal{N}et(\{\hat{\boldsymbol{W}}_\ell\}_{\ell=1}^L; \boldsymbol{X})$, as follows.

Theorem 3.2 (cascade Net-Trim consistency). *Consider a network $\mathcal{N}et(\{\boldsymbol{W}_\ell\}_{\ell=1}^L; \boldsymbol{X})$, which is normalized, i.e., $\|\boldsymbol{W}_\ell\|_1 = 1$ for $\ell = 1, \ldots, L$. If the network is retrained according to Algorithm 3.1, the layer outcomes of the original and retrained networks will obey*

$$\left\| \hat{\boldsymbol{X}}^{(\ell)} - \boldsymbol{X}^{(\ell)} \right\|_F \leq \epsilon \prod_{j=2}^{\ell} \gamma_j. \tag{3.11}$$

*Proof.* See section SM1.2 of the Supplementary Materials. ∎

Specifically, when an identical slackness rate is used across all the layers, one would have $\|\hat{\boldsymbol{X}}^{(L)} - \boldsymbol{X}^{(L)}\|_F \leq \gamma^{(L-1)}\epsilon$, which is a controllably small quantity, given that $\gamma$ can be selected arbitrarily close to 1. For instance, when $\gamma = 1.01$ and $L = 10$, the total network discrepancy

would still be less than $1.1\epsilon$. As will be demonstrated in the experiments section, for the same level of total network discrepancy, the cascade Net-Trim is capable of producing sparser networks. However, such reduction is achieved at the expense of the loss in distributability, which makes the parallel scheme computationally more attractive for big data problems.

**4. Sample complexity bounds using subgaussian random flow.** In the previous section we discussed and analyzed the convex retraining scheme and its consistency with the reference model. In this section we analyze the sample complexity of the proposed retraining framework. Basically, the goal of this section is addressing the following question: *If there exists a sparse transformation matrix relating the input and output of a layer, how many random samples are sufficient to recover it via the proposed retraining scheme?*

As will be detailed in what follows, we will show that retraining each neuron within the network is possible with fewer samples than the neuron degrees of freedom. More specifically, for a trained neuron with $N$ input ports, if generating an identical response is possible with $s \ll N$ nonzero weights, Net-Trim is able to recover such a model with only $\mathcal{O}(s\log(N/s))$ random samples. This result may apply to the neurons of any layer within the network, as long as some standard statistical properties can be established for the input samples.

To present the results, we first start with a brief overview of subgaussian random variables. For a more comprehensive overview, the reader is referred to [27] and section 2.2 of [26].

Definition 4.1 (subgaussian random variable). *A random variable $\varphi$ is subgaussian*[4] *if there exists a constant $\kappa$, such that for all $t \geq 0$,*

$$\mathbb{P}\left\{|\varphi| > t\right\} \leq \exp\left(1 - \frac{t^2}{\kappa^2}\right). \tag{4.1}$$

*Equivalently, $\varphi$ is subgaussian if there exists a constant $\hat{\kappa}$ such that*

$$\mathbb{E}\exp\left(\frac{\varphi^2}{\hat{\kappa}^2}\right) \leq e. \tag{4.2}$$

The subgaussian norm of $\varphi$, also referred to as the Orlicz norm, is denoted by $\|\varphi\|_{\psi_2}$, and defined as

$$\|\varphi\|_{\psi_2} \triangleq \sup_{p\geq 1} p^{-\frac{1}{2}} \left(\mathbb{E}|\varphi|^p\right)^{\frac{1}{p}}.$$

While calculating the exact Orlicz norm can be challenging, if either one of the properties (4.1) or (4.2) holds, $\|\varphi\|_{\psi_2}$ is the smallest possible number ($\kappa$ or $\hat{\kappa}$) in either one of these inequalities, up to an absolute constant.

Definition 4.2 (subgaussian random vector). *A random vector $\boldsymbol{\varphi} \in \mathbb{R}^N$ is subgaussian if for all $\boldsymbol{\alpha} \in \mathbb{R}^N$ (or, equivalently, all $\boldsymbol{\alpha} \in \mathbb{S}^{N-1}$), the one-dimensional marginals $\boldsymbol{\alpha}^\top\boldsymbol{\varphi}$ are subgaussian.*

The notion of Orlicz norm also generalizes to the vector case as

$$\|\boldsymbol{\varphi}\|_{\psi_2} \triangleq \sup_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \|\boldsymbol{\alpha}^\top\boldsymbol{\varphi}\|_{\psi_2} = \sup_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \sup_{p\geq 1} p^{-\frac{1}{2}} \left(\mathbb{E}\left|\boldsymbol{\alpha}^\top\boldsymbol{\varphi}\right|^p\right)^{\frac{1}{p}}. \tag{4.3}$$

[4] In general, the right-hand expression in (4.1) can be replaced with $c\exp\left(-t^2/\kappa^2\right)$ using two absolute constants $c$ and $\kappa$.

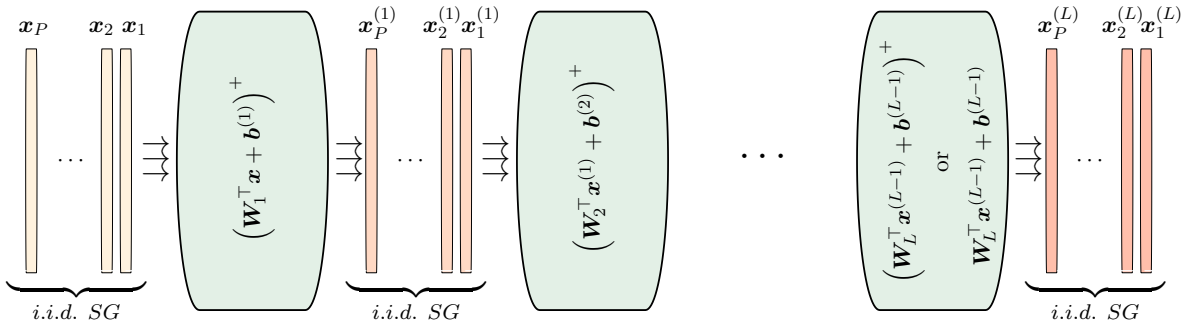



**Figure 2.** *When $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_P$, the input samples to the proposed neural network, are independently drawn from a subgaussian distribution, the input/output vectors of every subsequent layer remain i.i.d. and subgaussian.*

We are now ready to state the first result, which warrants a subgaussian random flow across the network, as long as the network input samples are independently drawn from a standard Gaussian (or subgaussian) distribution.

Theorem 4.3 (subgaussian flow). *Consider a network with fixed parameters $\boldsymbol{W}_\ell$, $\boldsymbol{b}^{(\ell)}$, where the input and output to each layer are related via*

$$\boldsymbol{x}^{(\ell)} = \texttt{ReLU}\left(\boldsymbol{W}_\ell^\top \boldsymbol{x}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}\right), \qquad \ell = 1, \ldots, L. \tag{4.4}$$

*If the network is fed with i.i.d. sample vectors $\boldsymbol{x}_1^{(0)}, \ldots, \boldsymbol{x}_P^{(0)} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, the response samples at each layer output remain i.i.d. subgaussian.*

*Proof outline.* A detailed proof is presented in section SM1.3 of the Supplementary Materials. Here, we briefly outline the steps. Establishing (4.1) to show that an affine transformation preserves subgaussianity is straightforward. Further, if $\boldsymbol{x}$ is subgaussian, then the ReLU version, $\boldsymbol{x}^+$, is also subgaussian. This follows simply since for a unit norm $\boldsymbol{\alpha}$,

$$|\boldsymbol{\alpha}^\top \boldsymbol{x}^+| \le \|\boldsymbol{\alpha}\| \|\boldsymbol{x}^+\| \le \|\boldsymbol{x}\|.$$

We finally make use of a result from [13], which warrants the subgaussianity of the $\ell_2$ norm of a subgaussian vector. ∎

As detailed in the proof, the result of Theorem 4.3 still holds when the network input samples are independently drawn from a subgaussian distribution instead of a standard normal, and/or when the last layer skips a ReLU activation. Specifically, when the network is fed with $\boldsymbol{x}_1^{(0)}, \boldsymbol{x}_2^{(0)}, \ldots, \boldsymbol{x}_P^{(0)}$, independently drawn from a subgaussian distribution, the resulting responses $\boldsymbol{x}_1^{(\ell)}, \boldsymbol{x}_2^{(\ell)}, \ldots, \boldsymbol{x}_P^{(\ell)}$ at any layer $\ell$ remain independent and subgaussian. The diagram in Figure 2 demonstrates such statistical structure among the layer inputs across the network.

Having independent subgaussian samples at any layer input port allows us to relate the number of samples to the recovery of a reduced model. Exchanging the layer index with the general input/output notation, when $\boldsymbol{X}^{in} \in \mathbb{R}^{N \times P}$ and $\boldsymbol{X}^{out} \in \mathbb{R}^{M \times P}$ are, respectively, the input and output to a layer, related via $\boldsymbol{X}^{out} = \texttt{ReLU}(\boldsymbol{W}^\top \boldsymbol{X}^{in})$, obtaining the pruned layer matrix $\hat{\boldsymbol{W}} \in \mathbb{R}^{N \times M}$ is performed via

$$\hat{\boldsymbol{W}} = \underset{\boldsymbol{W}}{\arg\min} \ \|\boldsymbol{W}\|_1 \quad \text{subject to} \quad \boldsymbol{W} \in \mathcal{C}_\epsilon\left(\boldsymbol{X}^{in}, \boldsymbol{X}^{out}, \boldsymbol{0}\right). \tag{4.5}$$

When $\epsilon = 0$, the program in (4.5) decouples into $M$ individual convex programs each retraining a column in $\boldsymbol{W}$. Basically, instead of solving (4.5) for $\hat{\boldsymbol{W}}$, if $\boldsymbol{x}^{out\top} \in \mathbb{R}^P$ is a row in $\boldsymbol{X}^{out}$, the corresponding column in $\hat{\boldsymbol{W}}$ can be calculated via

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \|\boldsymbol{w}\|_1 \quad \text{subject to} \quad \boldsymbol{w} \in \mathcal{C}_0\left(\boldsymbol{X}^{in}, \boldsymbol{x}^{out\top}, \boldsymbol{0}\right), \tag{4.6}$$

reducing (4.5) to retraining each of the $M$ output neurons, individually.[5] Even when $\epsilon \neq 0$, one may solve $M$ programs, each corresponding to a column of $\hat{\boldsymbol{W}}$, as

$$\underset{\boldsymbol{w}}{\text{minimize}} \quad \|\boldsymbol{w}\|_1 \quad \text{subject to} \quad \boldsymbol{w} \in \mathcal{C}_{\epsilon'}\left(\boldsymbol{X}^{in}, \boldsymbol{x}^{out\top}, \boldsymbol{0}\right), \tag{4.7}$$

where setting $\epsilon' = \frac{\epsilon}{\sqrt{M}}$ yields a reasonable approximation to (4.5). Basically, program (4.7) is a *neuronwise* application of the Net-Trim to approximate a layerwise scheme.

In the remainder of this section we will focus on the program (4.7) as a columnwise and simplified surrogate to (4.5). Upon the existence of a sparse set of weights, which (approximately) relates the inputs of a neuron to its outputs, our analysis relates the recovery of those sparse weights to $P$, the number of retraining samples. Specifically, the case $\epsilon = 0$ (which not only makes (4.7) an exact alternative to (4.5), but also makes the cascade and parallel schemes equivalent) is applicable to an underdetermined regime, where the relationship between $\boldsymbol{X}^{in}$ and $\boldsymbol{X}^{out}$ can be established via infinitely many $\boldsymbol{W}$ matrices and one seeks a unique sparse solution via (4.5). A corollary of our result reveals that for this special case, the required number of samples may be much less than the layer (neuron) degrees of freedom.

Before stating the main technical result, we would like to introduce some notions used in the presentation. When a neuron is initially trained via a vector $\boldsymbol{w}_0 \in \mathbb{R}^N$ and fed with i.i.d. instances of $\boldsymbol{x}$, the activation pattern of the neuron is fully controlled by the sign of $\boldsymbol{w}_0^\top \boldsymbol{x}$. In this case, one expects to gain the main retraining information from the cases when ReLU is in the linear mode (i.e., $\boldsymbol{w}_0^\top \boldsymbol{x} > 0$). In this regard, corresponding to the random input $\boldsymbol{x}$, we define the random *virtual input* as

$$\boldsymbol{v} = \boldsymbol{x} 1_{\boldsymbol{w}_0^\top \boldsymbol{x} > 0} = \begin{cases} \boldsymbol{x} & \boldsymbol{w}_0^\top \boldsymbol{x} > 0, \\ \boldsymbol{0} & \boldsymbol{w}_0^\top \boldsymbol{x} \leq 0. \end{cases}$$

The virtual random vector plays a key role in our presentation. Our presentation also depends on the smallest eigenvalue of the virtual covariance matrix, which follows the standard definition:

$$\lambda_{\min}\left(\operatorname{cov}\left(\boldsymbol{v}\right)\right) = \inf_{\boldsymbol{\alpha} \in \mathbb{S}^{N-1}} \boldsymbol{\alpha}^\top \operatorname{cov}\left(\boldsymbol{v}\right) \boldsymbol{\alpha}, \quad \text{where} \quad \operatorname{cov}\left(\boldsymbol{v}\right) = \mathbb{E}\left(\boldsymbol{v}\boldsymbol{v}^\top\right) - \mathbb{E}\left(\boldsymbol{v}\right) \mathbb{E}\left(\boldsymbol{v}\right)^\top.$$

Theorem 4.4 (main sample complexity result). *For the model (2.2), consider a trained neuron obeying* $\boldsymbol{x}^{out} = \texttt{ReLU}(\boldsymbol{X}^{in\top} \boldsymbol{w}_0)$, *where* $\boldsymbol{X}^{in} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_P] \in \mathbb{R}^{N \times P}$ *and* $\boldsymbol{x}, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$ *are independent samples of a subgaussian distribution. Assume that an* $s$*-sparse vector* $\boldsymbol{w}^* \in \mathbb{R}^N$

[5] Technically, $\boldsymbol{w}$ and $\boldsymbol{x}^{out\top}$ in (4.6) should be indexed by $i \in \{1, \ldots, M\}$, which we skip to keep the notation simple.

*is feasible for the convex program* (4.7) *and* $\hat{\boldsymbol{w}}$ *is any solution to* (4.7). *Fix* $\beta \geq 1/2$ *and* $t \geq 0$, *and define the virtual input* $\boldsymbol{v} = \boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}$. *Then with probability exceeding* $1 - e^{-ct}$,

$$\|\hat{\boldsymbol{w}} - \boldsymbol{w}^*\| \leq \frac{2\epsilon'}{\|\boldsymbol{v} - \mathbb{E}\boldsymbol{v}\|_{\psi_2}\left(\sqrt{\frac{P}{C_{\beta,v}}} - C\sqrt{s\log\left(\frac{N}{s}\right) + s + 1 + t}\right)_+}. \tag{4.8}$$

*The absolute constants* $c$ *and* $C$ *are universal and the coefficient* $C_{\beta,v}$ *depends on the statistics of the virtual input via*

$$C_{\beta,v} = (1+\beta)^2\left(\frac{\|\boldsymbol{v} - \mathbb{E}\boldsymbol{v}\|_{\psi_2}^2}{\lambda_{\min}(\mathrm{cov}(\boldsymbol{v}))}\right)^{3+\frac{1}{\beta}}. \tag{4.9}$$

*Proof outline.* As part of the technical contribution of this paper, a formal proof of the theorem is presented at the end of this section (section 4.2). To establish the result we use the *bowling scheme* proposed by [25], which discusses the recovery of a structured (e.g., sparse) signal from independent linear measurements. We make a connection between our problem with nonlinear constraints to the problem with linear constraints described there. While we used the compact model (2.2) for a more concise presentation, the model in (2.1) is still covered by Theorem 4.4, treating the intercept as a constant feature appended to the neuron input. ∎

In an underdetermined setting, where the number of samples used to apply the Net-Trim is less than the neuron's input size (i.e., $P < N$), one may use (4.6) to retrain the layer. In this case, upon the existence of a sparse solution, a corollary of Theorem 4.4 reveals that the required number of samples may be much less than the layer's input dimension.

Corollary 4.5. *Consider a similar neuron and setting as in Theorem* 4.4. *Assume that an* $s$*-sparse vector* $\boldsymbol{w}^* \in \mathbb{R}^N$ *is capable of generating an identical response to* $\boldsymbol{X}^{in}$ *as* $\boldsymbol{x}^{out}$. *If*

$$P \gtrsim C_{\beta,v}\left(s\log\left(\frac{N}{s}\right) + s + 1 + t\right), \tag{4.10}$$

*retraining the neuron via* (4.6) *accurately identifies* $\boldsymbol{w}^*$ *with probability exceeding* $1 - e^{-ct}$.

In the presentation of Theorem 4.4, the coefficient $C_{\beta,v}$ is related to the statistics of the centered virtual input, regardless of the mean shift that the previous activation units have caused to the input. This is important because the Orlicz norm of a noncentered random vector can easily become dimension-dependent. For instance, if the components of $\boldsymbol{x} \in \mathbb{R}^N$ are i.i.d. standard Gaussians, one can easily verify that $\|\boldsymbol{x}^+\|_{\psi_2} = \mathcal{O}(\sqrt{N})$, while $\|\boldsymbol{x}^+ - \mathbb{E}\boldsymbol{x}^+\|_{\psi_2} = \mathcal{O}(1)$.

Finally, Theorem 4.4 can be used as a general and powerful tool to estimate the retraining sample complexity for any layer within the network. To establish the $\mathcal{O}(s\log(N/s))$ rate for a given layer, we need only show that for the corresponding input $\boldsymbol{x}$ and initially trained weights $\boldsymbol{w}_0$, the virtual input $\boldsymbol{v} = \boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}$ satisfies the following two conditions:

$$\lambda_{\min}\left(\mathrm{cov}\left(\boldsymbol{v}\right)\right) \gtrsim 1 \qquad \text{and} \qquad \|\boldsymbol{v} - \mathbb{E}\boldsymbol{v}\|_{\psi_2} \lesssim 1. \tag{4.11}$$

As an insightful example, we go through the exercise of establishing the bounds in (4.11) for a layer fed with i.i.d. Gaussian samples; this is not an unreasonable scenario for the first layer of a neural network. As will be detailed in section 4.1 below, using standard tools to verify the conditions in (4.11) conveniently proves the $\mathcal{O}(s\log(N/s))$ rate for such layer.

For a network fed with i.i.d. Gaussian samples, going through a similar exercise for the subsequent layers ($\ell > 1$), with independent copies of the random input $\boldsymbol{x}^{(\ell)}$), requires tracing the statistics of $\boldsymbol{\upsilon}^{(\ell)} = \boldsymbol{x}^{(\ell)} 1_{\boldsymbol{w}_0^\top \boldsymbol{x}^{(\ell)} > 0}$ down to the Gaussian input $\boldsymbol{x}^{(0)}$. In such case, warranting the conditions in (4.11) would require stating realistic conditions on the initially trained $\boldsymbol{W}_j$ for $j = 1, \dots, \ell$. Such a generalization could be application specific and beyond the current scope of this paper; thus it is left as a potential future work.

**4.1. Feeding a neuron with i.i.d. Gaussian samples.** In this section we go through the exercise of establishing the conditions in (4.11) for a neuron fed with independent copies of $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, $\boldsymbol{x} \in \mathbb{R}^N$. Below, we go through each bound in (4.11), separately. In all of the calculations, $\boldsymbol{w}_0 \neq \boldsymbol{0}$ is a fixed vector that corresponds to the initially trained model. In [2], the authors go through a chain of techniques to prove an $\mathcal{O}(s \log N)$ sample complexity by carefully constructing a dual certificate for the convex program. Here we will see that thanks to Theorem 4.4, such a process is markedly reduced to establishing the conditions in (4.11), which is conveniently fulfilled using standard tools.

*Step* 1: *Bounding the covariance matrix.* To evaluate the virtual input covariance matrix we have

$$\begin{aligned}
\lambda_{\min}\left(\operatorname{cov}\left(\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right)\right) &= \lambda_{\min}\left(\mathbb{E}\boldsymbol{x}\boldsymbol{x}^\top 1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0} - \left(\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right)\left(\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right)^\top\right) \\
&\geq \lambda_{\min}\left(\mathbb{E}\boldsymbol{x}\boldsymbol{x}^\top 1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right) + \lambda_{\min}\left(-\left(\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right)\left(\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right)^\top\right) \\
&= \lambda_{\min}\left(\mathbb{E}\boldsymbol{x}\boldsymbol{x}^\top 1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right) - \left\|\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\right\|^2, \qquad (4.12)
\end{aligned}$$

where the second line follows from Weyl's inequality. To conveniently calculate the required moments, we can make use of the following lemma, which reduces the calculations to the bivariate case.

Lemma 4.6 (bivariate moment reduction). *Consider* $\boldsymbol{x} = (x_1, \dots, x_N)^\top \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ *and let* $g(.) : \mathbb{R} \to \mathbb{R}$ *be a real-valued function. Then, for any fixed vectors* $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \mathbb{S}^{N-1}$,

$$\mathbb{E}_{\boldsymbol{x}}\; g\left(\boldsymbol{\alpha}^\top \boldsymbol{x}\right) 1_{\boldsymbol{\beta}^\top \boldsymbol{x}>0} = \mathbb{E}_{x_1,x_2}\; g\left(\left(\boldsymbol{\alpha}^\top \boldsymbol{\beta}\right) x_1 + \sqrt{1-\left(\boldsymbol{\alpha}^\top \boldsymbol{\beta}\right)^2} x_2\right) 1_{x_1>0}. \qquad (4.13)$$

*Proof.* See section SM1.4 of the Supplementary Materials. ∎

Without loss of generality we can assume $\boldsymbol{w}_0 \in \mathbb{S}^{N-1}$, and apply (4.13) to the first

right-hand side term in (4.12) to get

$$\begin{aligned}\lambda_{\min}\left(\mathbb{E}\boldsymbol{x}\boldsymbol{x}^\top 1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right) &= \inf_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \mathbb{E}\left(\boldsymbol{\alpha}^\top\boldsymbol{x}\right)^2 1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\\ &= \inf_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \mathbb{E}_{x_1,x_2}\left(\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)x_1+\sqrt{1-\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)^2}x_2\right)^2 1_{x_1>0}\\ &= \inf_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \frac{1}{2}\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)^2+\frac{1}{2}\left(1-\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)^2\right)\\ &= \frac{1}{2}.\end{aligned}$$

For the second term in (4.12) we have

$$\begin{aligned}\left\|\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right\| &= \sup_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \mathbb{E}\left(\boldsymbol{\alpha}^\top\boldsymbol{x}\right)1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\\ &= \sup_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \mathbb{E}_{x_1,x_2}\left(\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)x_1+\sqrt{1-\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)^2}x_2\right)1_{x_1>0}\\ &= \sup_{\boldsymbol{\alpha}\in\mathbb{S}^{N-1}} \frac{1}{\sqrt{2\pi}}\left(\boldsymbol{\alpha}^\top\boldsymbol{w}_0\right)\\ &= \frac{1}{\sqrt{2\pi}},\end{aligned}$$

as a result of which one has $\lambda_{\min}(\mathrm{cov}(\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0})) \geq 1/2 - 1/(2\pi)$.

*Step 2: Bounding the Orlicz norm.* To bound the Orlicz norm of the centered virtual input by a constant, we need only introduce a constant $\kappa$ such that for all $\boldsymbol{\alpha}\in\mathbb{S}^{N-1}$ the marginals $\boldsymbol{\alpha}^\top(\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0} - \mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0})$ obey (4.1). To this end, one has

$$\begin{aligned}\forall\boldsymbol{\alpha}\in\mathbb{S}^{N-1}: \left|\boldsymbol{\alpha}^\top\left(\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}-\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right)\right| &\leq \left|\boldsymbol{\alpha}^\top\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right| + \left\|\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right\|\\ &\leq \left|\boldsymbol{\alpha}^\top\boldsymbol{x}\right| + \frac{1}{\sqrt{2\pi}}.\end{aligned}$$

As a result, for $\boldsymbol{x}\sim\mathcal{N}(\boldsymbol{0},\boldsymbol{I})$ and any fixed $\boldsymbol{\alpha}\in\mathbb{S}^{N-1}$,

$$\begin{aligned}\forall t\geq 0: \quad \mathbb{P}\left\{\left|\boldsymbol{\alpha}^\top\left(\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}-\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right)\right|>t\right\} &\leq \mathbb{P}\left\{\left|\boldsymbol{\alpha}^\top\boldsymbol{x}\right|+\frac{1}{\sqrt{2\pi}}>t\right\}\\ &= \mathbb{P}\left\{\left|\boldsymbol{\alpha}^\top\boldsymbol{x}\right|>\max\left(t-\frac{1}{\sqrt{2\pi}},0\right)\right\}\\ &\leq \exp\left(-\frac{1}{2}\max\left(t-\frac{1}{\sqrt{2\pi}},0\right)^2\right),\end{aligned}$$

where in the last inequality we used the fact that $\boldsymbol{\alpha}^\top\boldsymbol{x}\sim\mathcal{N}(0,1)$ and for a standard normal variable $z$, $\mathbb{P}\{|z|\geq t\}\leq\exp(-t^2/2)$ for all $t\geq 0$. Finally, we can use the basic inequality stated in Lemma SM1.1 of the proofs section to get

$$\forall t\geq 0: \quad \mathbb{P}\left\{\left|\boldsymbol{\alpha}^\top\left(\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}-\mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top\boldsymbol{x}>0}\right)\right|>t\right\}\leq\exp\left(1-\frac{t^2}{2+\frac{1}{2\pi}}\right),$$

which implies that $\|\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0} - \mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}\|_{\psi_2} \lesssim 1$.

**4.2. Proof of the main sample complexity result (Theorem 4.4).** After presenting the application of Theorem 4.4 in establishing an $\mathcal{O}(s\log(N/s))$ sample complexity rate for a neuron fed with Gaussian samples, in this section we go through a step by step proof of this theorem.

With reference to (4.6), for $\boldsymbol{X}^{in} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_P] \in \mathbb{R}^{N\times P}$ and

$$\Omega = \left\{p : x_p^{out} > 0\right\} = \left\{p : \boldsymbol{x}_p^\top \boldsymbol{w}_0 > 0\right\},$$

we bound the distance between the prescribed vector $\boldsymbol{w}^*$ and any solution $\hat{\boldsymbol{w}}$ of the convex program

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{w}\|_1 \qquad \text{subject to} \qquad \begin{cases} \left\| {\boldsymbol{X}^{in}_{:,\Omega}}^\top \boldsymbol{w} - \boldsymbol{x}^{out}_\Omega \right\| \le \epsilon, \\ {\boldsymbol{X}^{in}_{:,\Omega^c}}^\top \boldsymbol{w} \preceq \boldsymbol{0}. \end{cases} \tag{4.14}$$

For general $\boldsymbol{X}, \boldsymbol{X}_0 \in \mathbb{R}^{N\times P}$ and $\Omega \subseteq \{1, \dots, P\}$, consider the operator

$$\mathcal{T}_\Omega^{\boldsymbol{X}_0} \boldsymbol{X} \triangleq \boldsymbol{X}\,\mathrm{diag}\,(\mathbb{1}_\Omega) + \boldsymbol{X}_0,$$

where $\mathbb{1}_\Omega \in \mathbb{R}^P$ is the indicator of the set $\Omega$. Simply, $\mathcal{T}_\Omega^{\boldsymbol{0}} \boldsymbol{X}$ replaces columns of $\boldsymbol{X}$ indexed by $\Omega^c$ with zero vectors. Exploiting the notion of minimum conic singular value, we first state a characterization of the solution to (4.14), which generally holds regardless of the specific structure of $\boldsymbol{X}^{in}$.

Lemma 4.7 (extended conic singular value). *Fix $\boldsymbol{\mu} \in \mathbb{R}^N$ and $\sigma \in \mathbb{R} - \{0\}$. Consider $\boldsymbol{w}^* \in \mathbb{R}^N$ to be a (sparse) feasible vector for (4.14). Define the descent cone*

$$\mathcal{D} = \bigcup_{\tau>0} \left\{ \begin{pmatrix} \boldsymbol{y} \\ z \end{pmatrix} \in \mathbb{R}^{N+1} : \|\boldsymbol{w}^* + \tau \boldsymbol{y}\|_1 \le \|\boldsymbol{w}^*\|_1 \right\},$$

*and for $\boldsymbol{\Phi} = \mathcal{T}_\Omega^{-\boldsymbol{\mu}\boldsymbol{1}^\top} \boldsymbol{X}^{in}$ set*

$$\lambda_{\boldsymbol{w}^*}(\boldsymbol{\Phi}, \sigma) = \inf \ \left\{ \left\| \begin{pmatrix} \boldsymbol{\Phi}^\top & \sigma \boldsymbol{1} \end{pmatrix} \boldsymbol{v} \right\| : \ \boldsymbol{v} \in \mathcal{D} \cap \mathbb{S}^N \right\}. \tag{4.15}$$

*Then for any solution $\hat{\boldsymbol{w}}$ to (4.14),*

$$\|\hat{\boldsymbol{w}} - \boldsymbol{w}^*\| \le \frac{2\epsilon}{\lambda_{\boldsymbol{w}^*}(\boldsymbol{\Phi}, \sigma)}. \tag{4.16}$$

*Proof.* See section SM1.5 of the Supplementary Materials. ∎

Using Lemma 4.7 and the bowling scheme sketched in [25], we continue with lower-bounding the minimum conic singular value away from zero, and relating the conditions to the number of samples, $P$.

To this end, we may look into the structure of the matrix $\boldsymbol{\Phi}$ in Lemma 4.7 as being populated with independent copies of $\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0} - \boldsymbol{\mu}$ as the columns, and exploit the independence required

for the bowling scheme. To assure centered columns, we choose $\boldsymbol{\mu} = \mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}$, making columns of $\boldsymbol{\Phi} = [\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_P]$ independent copies of the centered subgaussian[6] random vector

$$\boldsymbol{\varphi} \triangleq \boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0} - \mathbb{E}\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}.$$

For reasons that become apparent later in the proof, our arbitrary choice of $\sigma$ in Lemma 4.7 is narrowed to

$$\sigma_0 \triangleq \sqrt{2}\|\boldsymbol{\varphi}\|_{\psi_2}.$$

In a random setting, to lower-bound the minimum conic singular value, we adapt the following result from [25, Prop. 5.1] (or see Theorem 5.4 of [17] for the original statement).

Theorem 4.8. *Fix a set $E \subset \mathbb{R}^d$. Let $\boldsymbol{\phi}$ be a random vector on $\mathbb{R}^d$, and let $\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_P$ be independent copies of $\boldsymbol{\phi}$. For $\xi \geq 0$, suppose the marginal tail relation below holds:*

$$\inf_{\boldsymbol{v}\in E} \mathbb{P}\left\{\left|\boldsymbol{\phi}^\top \boldsymbol{v}\right| \geq \xi\right\} \geq C_\xi > 0.$$

*Let $\varepsilon_1, \ldots, \varepsilon_P$ be independent Rademacher random variables, independent from everything else, and define the mean empirical width of the set $E$:*

$$\mathcal{W}_P(E;\boldsymbol{\phi}) \triangleq \mathbb{E}\sup_{\boldsymbol{v}\in E}\langle \boldsymbol{h}, \boldsymbol{v}\rangle, \quad \text{where} \quad \boldsymbol{h} = \frac{1}{\sqrt{P}}\sum_{p=1}^{P}\varepsilon_p\boldsymbol{\phi}_p. \tag{4.17}$$

*Then, for any $\xi > 0$ and $t > 0$, with probability at least $1 - \exp(-t^2/2)$,*

$$\inf_{\boldsymbol{v}\in E}\left(\sum_{p=1}^{P}\left(\boldsymbol{\phi}_p^\top \boldsymbol{v}\right)^2\right)^{\frac{1}{2}} \geq \frac{\xi}{2}C_\xi\sqrt{P} - 2\mathcal{W}_P(E;\boldsymbol{\phi}) - \frac{\xi}{2}t. \tag{4.18}$$

For a more compact (and inline) notation, we use the following notation for the concatenation of a vector $\boldsymbol{w}$ and a scalar $u$:

$$\boldsymbol{w}^\frown u \triangleq \begin{pmatrix}\boldsymbol{w}\\ u\end{pmatrix}.$$

Also, for a given objective and point $\boldsymbol{v}_0 \in \mathbb{R}^d$, we denote the descent cone by

$$\mathcal{D}_{\boldsymbol{v}}(f(\boldsymbol{v});\boldsymbol{v}_0) = \bigcup_{\tau>0}\left\{\boldsymbol{y}\in\mathbb{R}^d : f(\boldsymbol{v}_0+\tau\boldsymbol{y}) \leq f(\boldsymbol{v}_0)\right\}.$$

To show that condition (4.15) holds for the prescribed $s$-sparse vector $\boldsymbol{w}^*$, we will show that for sufficiently large $P$, the right-hand side expression in (4.18) can be bounded away from zero. To apply Theorem 4.8 to our problem in (4.15), the random vector $\boldsymbol{\phi}$ and the set $E$ to consider are

$$\boldsymbol{\phi} = \boldsymbol{\varphi}^\frown\sigma_0 \quad \text{and} \quad E = \mathcal{D}_{\boldsymbol{w}^\frown u}(\|\boldsymbol{w}\|_1; \boldsymbol{w}^{*\frown}u^*) \cap \mathbb{S}^N,$$

[6] Since $|\boldsymbol{\alpha}^\top \boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}| \leq |\boldsymbol{\alpha}^\top \boldsymbol{x}|$ and for $t \geq 0$, $\mathbb{P}\{|\boldsymbol{\alpha}^\top \boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}| > t\} \leq \mathbb{P}\{|\boldsymbol{\alpha}^\top \boldsymbol{x}| > t\}$, (4.1) confirms that $\boldsymbol{x}$ being subgaussian implies that $\boldsymbol{x}1_{\boldsymbol{w}_0^\top \boldsymbol{x}>0}$ is subgaussian.

where

$$\begin{aligned}\mathcal{D}_{\boldsymbol{w}^\frown u}\left(\|\boldsymbol{w}\|_1;\boldsymbol{w}^{*\frown}u^*\right) &= \bigcup_{\tau>0}\left\{\boldsymbol{y}^\frown z\in\mathbb{R}^{N+1}:\|\boldsymbol{w}^*+\tau\boldsymbol{y}\|_1\le\|\boldsymbol{w}^*\|_1\right\}\\ &= \mathcal{D}_{\boldsymbol{w}}\left(\|\boldsymbol{w}\|_1;\boldsymbol{w}^*\right)\times\mathbb{R},\end{aligned}$$

and $u^*=\sigma_0^{-1}(\mathbb{E}\boldsymbol{x}\mathbf{1}_{\boldsymbol{w}_0^\top\boldsymbol{x}>0})^\top\boldsymbol{w}^*$. Note that in the formulation above, $\mathcal{D}_{\boldsymbol{w}^\frown u}(.;.)\subset\mathbb{R}^{N+1}$, while $\mathcal{D}_{\boldsymbol{w}}(.;.)\subset\mathbb{R}^N$. The remainder of the proof focuses on bounding the contributing terms on the right-hand side expression of (4.18).

*Bounding the mean empirical width.* In this section of the proof, we aim to upper-bound

$$\mathcal{W}_P\left(\mathcal{D}_{\boldsymbol{w}^\frown u}\left(\|\boldsymbol{w}\|_1;\boldsymbol{w}^{*\frown}u^*\right)\cap\mathbb{S}^N;\boldsymbol{\varphi}^\frown\sigma_0\right),$$

where, following the formulation in (4.17), we have

$$\boldsymbol{h}=\frac{1}{\sqrt{P}}\sum_{p=1}^P\varepsilon_p\boldsymbol{\varphi}_p^\frown\sigma_0=\underbrace{\begin{pmatrix}\mathbf{0}\\ \frac{\sigma_0}{\sqrt{P}}\sum_{p=1}^P\varepsilon_p\end{pmatrix}}_{\boldsymbol{h}_u}+\underbrace{\begin{pmatrix}\frac{1}{\sqrt{P}}\sum_{p=1}^P\varepsilon_p\boldsymbol{\varphi}_p\\ 0\end{pmatrix}}_{\boldsymbol{h}_{\boldsymbol{w}}^\frown 0}.$$

Using the compact notation

$$\mathcal{K}_{\boldsymbol{w}^*,u^*}=\mathcal{D}_{\boldsymbol{w}^\frown u}\left(\|\boldsymbol{w}\|_1;\boldsymbol{w}^{*\frown}u^*\right)\quad\text{and}\quad\mathcal{K}_{\boldsymbol{w}^*}=\mathcal{D}_{\boldsymbol{w}}\left(\|\boldsymbol{w}\|_1;\boldsymbol{w}^*\right),$$

one has

$$\begin{aligned}\mathcal{W}_P\left(\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N;\boldsymbol{\varphi}^\frown\sigma_0\right)&=\mathbb{E}\sup_{\boldsymbol{v}\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N}\langle\boldsymbol{h},\boldsymbol{v}\rangle\\ &\le\mathbb{E}\sup_{\boldsymbol{v}\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N}\langle\boldsymbol{h}_u,\boldsymbol{v}\rangle+\mathbb{E}\sup_{\boldsymbol{v}\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N}\langle\boldsymbol{h}_{\boldsymbol{w}}^\frown 0,\boldsymbol{v}\rangle.\end{aligned}\tag{4.19}$$

For the first term in (4.19) note that

$$\begin{aligned}\mathbb{E}\sup_{\boldsymbol{w}^\frown u\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^n}\langle\boldsymbol{h}_u,\boldsymbol{w}^\frown u\rangle&=\mathbb{E}\sup_{\boldsymbol{w}^\frown u\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N}\left(\frac{\sigma_0}{\sqrt{P}}\sum_{p=1}^P\varepsilon_p\right)u\\ &=\mathbb{E}\left|\frac{\sigma_0}{\sqrt{P}}\sum_{p=1}^P\varepsilon_p\right|\\ &\le\frac{\sigma_0}{\sqrt{P}}\left(\mathbb{E}\left(\sum_{p=1}^P\varepsilon_p\right)^2\right)^{\frac{1}{2}}\\ &=\sigma_0.\end{aligned}\tag{4.20}$$

To bound the second term in (4.19), we proceed by first showing that for any fixed $\boldsymbol{h}_{\boldsymbol{w}}\in\mathbb{R}^N$,

$$\sup_{\boldsymbol{w}^\frown u\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N}\langle\boldsymbol{h}_{\boldsymbol{w}}^\frown 0,\boldsymbol{w}^\frown u\rangle=\sup_{\boldsymbol{w}\in\mathcal{K}_{\boldsymbol{w}^*}\cap\mathbb{B}^N}\langle\boldsymbol{h}_{\boldsymbol{w}},\boldsymbol{w}\rangle,\tag{4.21}$$

where $\mathbb{B}^N$ denotes the Euclidean unit ball of dimension $N$. For this purpose only the following two cases need to be considered:

- **Case 1:** $\langle \boldsymbol{h}_{\boldsymbol{w}}, \boldsymbol{w} \rangle \leq 0 \, \forall \boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*}$.

In this case the supremum value for both sides of (4.21) is zero, which may be attained by picking $\boldsymbol{w} = \boldsymbol{0}$ and $u = 1$.

- **Case 2:** $\exists \boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*}$, such that $\langle \boldsymbol{h}_{\boldsymbol{w}}, \boldsymbol{w} \rangle > 0$.

To establish the equality in this case, we show that if $\hat{\boldsymbol{v}} = \hat{\boldsymbol{w}} \frown \hat{u}$ is a point at which the (positive) supremum is attained, i.e.,

$$\sup_{\boldsymbol{v} \in \mathcal{K}_{\boldsymbol{w}^*, u^*} \cap \mathbb{S}^N} \langle \boldsymbol{h}_{\widehat{\boldsymbol{w}}} 0, \boldsymbol{v} \rangle = \langle \boldsymbol{h}_{\widehat{\boldsymbol{w}}} 0, \hat{\boldsymbol{v}} \rangle = \langle \boldsymbol{h}_{\boldsymbol{w}}, \hat{\boldsymbol{w}} \rangle ,$$

then we must have $\hat{u} = 0$. If $\hat{u} \neq 0$, then the condition $\hat{\boldsymbol{v}} \in \mathbb{S}^N$ requires that $\|\hat{\boldsymbol{w}}\| < 1$. In this case the alternative feasible point $\tilde{\boldsymbol{v}} = \|\hat{\boldsymbol{w}}\|^{-1} \hat{\boldsymbol{w}} \frown 0$ produces a greater inner product:

$$\langle \boldsymbol{h}_{\widehat{\boldsymbol{w}}} 0, \tilde{\boldsymbol{v}} \rangle = \frac{1}{\|\hat{\boldsymbol{w}}\|} \langle \boldsymbol{h}_{\boldsymbol{w}}, \hat{\boldsymbol{w}} \rangle > \langle \boldsymbol{h}_{\boldsymbol{w}}, \hat{\boldsymbol{w}} \rangle ,$$

which cannot be possible. Therefore $\hat{u} = 0$, and for both sides of (4.21) the supremum value is $\langle \boldsymbol{h}_{\boldsymbol{w}}, \hat{\boldsymbol{w}} \rangle$. Combining Cases 1 and 2 establishes the claim in (4.21).

Now, employing (4.21) and (4.20) in (4.19) certifies that for

$$\boldsymbol{h}_{\boldsymbol{w}} = \frac{1}{\sqrt{P}} \sum_{p=1}^{P} \varepsilon_p \boldsymbol{\varphi}_p,$$

and some absolute constant $C$, one has

$$\begin{aligned} \mathcal{W}_P \left( \mathcal{K}_{\boldsymbol{w}^*, u^*} \cap \mathbb{S}^N ; \boldsymbol{\varphi} \frown \sigma_0 \right) &\leq \sigma_0 + \mathbb{E} \sup_{\boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*} \cap \mathbb{B}^N} \langle \boldsymbol{h}_{\boldsymbol{w}}, \boldsymbol{w} \rangle \\ &\leq \sqrt{2} \|\boldsymbol{\varphi}\|_{\psi_2} + C \|\boldsymbol{\varphi}\|_{\psi_2} \mathbb{E}_{\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})} \sup_{\boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*} \cap \mathbb{B}^N} \langle \boldsymbol{g}, \boldsymbol{w} \rangle. \end{aligned} \tag{4.22}$$

The last line in (4.22) is thanks to the generic chaining and majorizing measure theorems (as detailed in section 6.6 of [25], employing Theorems 1.2.6 and 2.1.1 of [23]). The second term in (4.22) can be related to the statistical dimension of $\mathcal{K}_{\boldsymbol{w}^*}$, and then the conic Gaussian width of $\mathcal{K}_{\boldsymbol{w}^*}$ as follows:

$$\begin{aligned} \left( \mathbb{E}_{\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})} \sup_{\boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*} \cap \mathbb{B}^N} \langle \boldsymbol{g}, \boldsymbol{w} \rangle \right)^2 &\leq \mathbb{E}_{\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})} \left( \sup_{\boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*} \cap \mathbb{B}^N} \langle \boldsymbol{g}, \boldsymbol{w} \rangle \right)^2 \\ &\leq \left( \mathbb{E}_{\boldsymbol{g} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})} \sup_{\boldsymbol{w} \in \mathcal{K}_{\boldsymbol{w}^*} \cap \mathbb{S}^{N-1}} \langle \boldsymbol{g}, \boldsymbol{w} \rangle \right)^2 + 1 \\ &\leq 2s \log \left( \frac{N}{s} \right) + 2s + 1. \end{aligned} \tag{4.23}$$

In the chain of inequalities leading to (4.23), the first inequality is the Jensen's, the second inequality is thanks to Proposition 10.2 of [3], and the last inequality uses the standard

Gaussian width calculation (see Example 4.3 in [25]). Combining (4.23) with (4.22) would yield

$$\mathcal{W}_P\left(\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N;\boldsymbol{\varphi}\frown\sigma_0\right)\lesssim\|\boldsymbol{\varphi}\|_{\psi_2}\left(\sqrt{s\log\left(\frac{N}{s}\right)+s}+1\right). \tag{4.24}$$

*Relating the marginal tail bound and the virtual covariance.* As the next step in lower-bounding the right-hand side expression in (4.18), noting that

$$\inf_{\boldsymbol{v}\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N}\mathbb{P}\left\{\left|\boldsymbol{v}^\top\boldsymbol{\varphi}\frown\sigma_0\right|\geq\xi\right\}\geq\inf_{\boldsymbol{v}\in\mathbb{S}^N}\mathbb{P}\left\{\left|\boldsymbol{v}^\top\boldsymbol{\varphi}\frown\sigma_0\right|\geq\xi\right\}, \tag{4.25}$$

in this section we focus on lower-bounding the right-hand side expression in (4.25) in terms of $\|\boldsymbol{\varphi}\|_{\psi_2}$ and the minimum eigenvalue of the virtual covariance matrix. To this end, using the notation

$$\tilde{\lambda}_{\min}\triangleq\lambda_{\min}\left(\operatorname{cov}\left(\boldsymbol{v}\right)\right)=\lambda_{\min}\left(\mathbb{E}\boldsymbol{\varphi}\boldsymbol{\varphi}^\top\right),$$

one has

$$\mathbb{E}\boldsymbol{\varphi}\frown\sigma_0\boldsymbol{\varphi}\frown\sigma_0{}^\top=\begin{pmatrix}\mathbb{E}\boldsymbol{\varphi}\boldsymbol{\varphi}^\top & \mathbf{0}\\ \mathbf{0}^\top & \sigma_0^2\end{pmatrix}\succeq\min\left(\tilde{\lambda}_{\min},\sigma_0^2\right)\boldsymbol{I}. \tag{4.26}$$

On the other hand, from the subgaussian properties of $\boldsymbol{\varphi}$ we have

$$\|\boldsymbol{\varphi}\|_{\psi_2}\geq\sup_{\boldsymbol{w}\in\mathbb{S}^{N-1}}2^{-\frac{1}{2}}\left(\mathbb{E}\left|\boldsymbol{w}^\top\boldsymbol{\varphi}\right|^2\right)^{\frac{1}{2}}\geq\inf_{\boldsymbol{w}\in\mathbb{S}^{N-1}}2^{-\frac{1}{2}}\left(\mathbb{E}\left|\boldsymbol{w}^\top\boldsymbol{\varphi}\right|^2\right)^{\frac{1}{2}}=\sqrt{\frac{\tilde{\lambda}_{\min}}{2}},$$

which simply implies that $\sigma_0^2\geq\tilde{\lambda}_{\min}$ and combining with (4.26) yields

$$\inf_{\boldsymbol{v}\in\mathbb{S}^N}\mathbb{E}\left|\boldsymbol{v}^\top\boldsymbol{\varphi}\frown\sigma_0\right|^2\geq\min\left(\tilde{\lambda}_{\min},\sigma_0^2\right)=\tilde{\lambda}_{\min}. \tag{4.27}$$

Considering a positive random variable $\chi$ and a fixed $\xi\geq0$, we can derive a variant of the Paley–Zygmund inequality by writing $\chi^2=\chi^2 1_{\{\chi^2<\xi^2\}}+\chi^2 1_{\{\chi^2\geq\xi^2\}}$, which, using the Hölder's inequality naturally, yields

$$\mathbb{E}\chi^2\leq\xi^2+\left(\mathbb{P}\left\{\chi\geq\xi\right\}\right)^{\frac{\beta}{1+\beta}}\left(\mathbb{E}\chi^{2(1+\beta)}\right)^{\frac{1}{1+\beta}},\quad\beta>0.$$

Subsequently, selecting $\xi\in[0,\sqrt{\tilde{\lambda}_{\min}}]$ warrants that

$$\forall\boldsymbol{v}\in\mathbb{S}^N:\qquad\mathbb{P}\left\{\left|\boldsymbol{v}^\top\boldsymbol{\varphi}\frown\sigma_0\right|\geq\xi\right\}\geq\left(\frac{\tilde{\lambda}_{\min}-\xi^2}{\left(\mathbb{E}\left|\boldsymbol{v}^\top\boldsymbol{\varphi}\frown\sigma_0\right|^{2(1+\beta)}\right)^{\frac{1}{\beta+1}}}\right)^{1+\frac{1}{\beta}}.$$

We can also use the subgaussian properties of $\boldsymbol{\varphi}$ to bound the denominator as follows:

$$\begin{aligned}\forall \boldsymbol{w}\frown u \in \mathbb{S}^N,\ \alpha \geq 1: \quad \left(\mathbb{E}\left|\boldsymbol{w}\frown u^\top \boldsymbol{\varphi}\frown \sigma_0\right|^\alpha\right)^{\frac{1}{\alpha}} &= \left(\mathbb{E}\left|\boldsymbol{w}^\top \boldsymbol{\varphi} + \sigma_0 u\right|^\alpha\right)^{\frac{1}{\alpha}} \\ &\leq \left(\mathbb{E}\left|\boldsymbol{w}^\top \boldsymbol{\varphi}\right|^\alpha\right)^{\frac{1}{\alpha}} + \sigma_0|u| \\ &= \|\boldsymbol{w}\|\left(\mathbb{E}\left|\frac{\boldsymbol{w}^\top}{\|\boldsymbol{w}\|}\boldsymbol{\varphi}\right|^\alpha\right)^{\frac{1}{\alpha}} + \sigma_0|u| \\ &\leq \sqrt{\alpha}\|\boldsymbol{\varphi}\|_{\psi_2}\|\boldsymbol{w}\| + \sqrt{2}\|\boldsymbol{\varphi}\|_{\psi_2}|u| \\ &\leq \sqrt{\alpha+2}\|\boldsymbol{\varphi}\|_{\psi_2},\end{aligned}$$

where the first inequality is a direct application of the Minkowski inequality, the second inequality uses the subgaussian definition (4.3), and the last bound is thanks to the Cauchy–Schwarz inequality. As a result for $\xi \in [0, \sqrt{\tilde{\lambda}_{\min}}]$

$$\inf_{\boldsymbol{v}\in\mathbb{S}^N} \mathbb{P}\left\{\left|\boldsymbol{v}^\top \boldsymbol{\varphi}\frown \sigma_0\right| \geq \xi\right\} \geq \left(\frac{\tilde{\lambda}_{\min} - \xi^2}{2(2+\beta)\|\boldsymbol{\varphi}\|_{\psi_2}^2}\right)^{1+\frac{1}{\beta}}. \tag{4.28}$$

*Combining the bounds.* We can now combine the bounds (4.22) and (4.28), and use $\xi = \sqrt{\tilde{\lambda}_{\min}}/3$ in Theorem 4.8 to state that with probability at least $1 - \exp(-t^2/2)$,

$$\begin{aligned}\inf_{\boldsymbol{v}\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N} \left(\sum_{p=1}^P \left(\boldsymbol{\varphi}_p\frown \sigma_0^\top \boldsymbol{v}\right)^2\right)^{\frac{1}{2}} &\geq \frac{\sqrt{\tilde{\lambda}_{\min}}}{6}\sqrt{P}\left(\frac{2\tilde{\lambda}_{\min}}{9(1+\frac{\beta}{2})\|\boldsymbol{\varphi}\|_{\psi_2}^2}\right)^{1+\frac{1}{\beta}} \\ &\quad - C\|\boldsymbol{\varphi}\|_{\psi_2}\left(\sqrt{s\log\left(\frac{N}{s}\right)+s+1}\right) - \frac{\sqrt{\tilde{\lambda}_{\min}}}{6}t.\end{aligned}$$

Noting that $\tilde{\lambda}_{\min} \leq 2\|\boldsymbol{\varphi}\|_{\psi_2}^2$ and using the basic inequality $(a+b)^2 \leq 2a^2 + 2b^2$ twice yields

$$\left(C\|\boldsymbol{\varphi}\|_{\psi_2}\left(\sqrt{s\log\left(\frac{N}{s}\right)+s+1}\right) + \frac{\sqrt{\tilde{\lambda}_{\min}}}{6}t\right)^2 \lesssim \|\boldsymbol{\varphi}\|_{\psi_2}^2\left(s\log\left(\frac{N}{s}\right)+s+1+t'\right),$$

where $t' = t^2/(36C^2)$. Also, after setting $\beta' = \beta/2$, since $(1+\beta')^{\frac{1}{\beta'}} \lesssim 1$ for $\beta' \geq 1/2$, we can bound the inverse square of the $\sqrt{P}$ coefficient as

$$\frac{36\left(9\left(1+\beta'\right)\|\boldsymbol{\varphi}\|_{\psi_2}^2\right)^{2+\frac{1}{\beta'}}}{\tilde{\lambda}_{\min}\left(2\tilde{\lambda}_{\min}\right)^{2+\frac{1}{\beta'}}} \lesssim \left(1+\beta'\right)^2\frac{\|\boldsymbol{\varphi}\|_{\psi_2}^{4+\frac{2}{\beta'}}}{\tilde{\lambda}_{\min}^{3+\frac{1}{\beta'}}} = \frac{C_{\beta',\boldsymbol{\varphi}}}{\|\boldsymbol{\varphi}\|_{\psi_2}^2},$$

where $C_{\beta',\boldsymbol{\varphi}}$ follows the definition in (4.9). Finally, after combining the bounds we determine

$$\inf_{\boldsymbol{v}\in\mathcal{K}_{\boldsymbol{w}^*,u^*}\cap\mathbb{S}^N} \left(\sum_{p=1}^P \left(\boldsymbol{\varphi}_p\frown \sigma_0^\top \boldsymbol{v}\right)^2\right)^{\frac{1}{2}} \gtrsim \|\boldsymbol{\varphi}\|_{\psi_2}\left(\sqrt{\frac{P}{C_{\beta',\boldsymbol{\varphi}}}} - C'\sqrt{s\log\left(\frac{N}{s}\right)+s+1+t'}\right)^+,$$

which together with Lemma 4.7 would verify the advertised claim in (4.8).

**5. Net-Trim implementation.** In this section we discuss details of an ADMM implementation for the Net-Trim convex program. The approach that we suggest here is based on the *global variable consensus* (see section 7.1 of [4]). This technique is useful in addressing convex optimizations with additively separable objectives.

For $\boldsymbol{W} \in \mathbb{R}^{N\times M}$, $\boldsymbol{X}^{in} \in \mathbb{R}^{N\times P}$, and $\Omega \subseteq \{1,\dots,M\} \times \{1,\dots,P\}$, the Net-Trim central program

$$\underset{\boldsymbol{W}}{\text{minimize}} \ \|\boldsymbol{W}\|_1 \quad \text{subject to} \quad \begin{cases} \left\|\left(\boldsymbol{W}^\top \boldsymbol{X}^{in} - \boldsymbol{X}^{out}\right)_\Omega\right\|_F \le \epsilon, \\ \left(\boldsymbol{W}^\top \boldsymbol{X}^{in}\right)_{\Omega^c} \le \boldsymbol{V}_{\Omega^c} \end{cases} \tag{5.1}$$

can be cast as the equivalent form

$$\underset{\substack{\boldsymbol{W}^{(1)}\in\mathbb{R}^{M\times P} \\ \boldsymbol{W}^{(2)},\boldsymbol{W}^{(3)}\in\mathbb{R}^{N\times M}}}{\text{minimize}} \ f_1\left(\boldsymbol{W}^{(1)}\right) + f_2\left(\boldsymbol{W}^{(2)}\right) \quad \text{subject to} \quad \begin{cases} \boldsymbol{W}^{(1)} = {\boldsymbol{W}^{(3)}}^\top \boldsymbol{X}^{in}, \\ \boldsymbol{W}^{(2)} = \boldsymbol{W}^{(3)}, \end{cases} \tag{5.2}$$

where $f_1(\boldsymbol{W}) = \mathcal{I}_{\left\|\boldsymbol{W}_\Omega - \boldsymbol{X}^{out}_\Omega\right\|_F \le \epsilon}(\boldsymbol{W}) + \mathcal{I}_{\boldsymbol{W}_{\Omega^c} \le \boldsymbol{V}_{\Omega^c}}(\boldsymbol{W})$, and $f_2(\boldsymbol{W}) = \|\boldsymbol{W}\|_1$. Here $\mathcal{I}_C(\cdot)$ represents the indicator function of the set $C$,

$$\mathcal{I}_C(\boldsymbol{W}) = \begin{cases} 0 & \boldsymbol{W} \in C, \\ +\infty & \boldsymbol{W} \notin C. \end{cases}$$

For the convex program (5.2), the ADMM update for each variable at the $k$th iteration follows the standard forms

$$\boldsymbol{W}^{(1)}_{k+1} = \underset{\boldsymbol{W}}{\arg\min} \ f_1(\boldsymbol{W}) + \frac{\rho}{2}\left\|\boldsymbol{W} + \boldsymbol{U}^{(1)}_k - {\boldsymbol{W}^{(3)}_k}^\top \boldsymbol{X}^{in}\right\|_F^2, \tag{5.3}$$

$$\boldsymbol{W}^{(2)}_{k+1} = \underset{\boldsymbol{W}}{\arg\min} \ f_2(\boldsymbol{W}) + \frac{\rho}{2}\left\|\boldsymbol{W} + \boldsymbol{U}^{(2)}_k - \boldsymbol{W}^{(3)}_k\right\|_F^2, \tag{5.4}$$

$$\boldsymbol{W}^{(3)}_{k+1} = \left(\boldsymbol{X}^{in}\boldsymbol{X}^{in\top} + \boldsymbol{I}\right)^{-1}\left(\boldsymbol{X}^{in}\left(\boldsymbol{W}^{(1)}_{k+1} + \boldsymbol{U}^{(1)}_k\right)^\top + \boldsymbol{W}^{(2)}_{k+1} + \boldsymbol{U}^{(2)}_k\right), \tag{5.5}$$

and the dual updates are performed via

$$\boldsymbol{U}^{(1)}_{k+1} = \boldsymbol{U}^{(1)}_k + \boldsymbol{W}^{(1)}_{k+1} - {\boldsymbol{W}^{(3)}_{k+1}}^\top \boldsymbol{X}^{in}, \quad \boldsymbol{U}^{(2)}_{k+1} = \boldsymbol{U}^{(2)}_k + \boldsymbol{W}^{(2)}_{k+1} - \boldsymbol{W}^{(3)}_{k+1}.$$

The update stated in (5.5) is derived by finding the minimizer of the augmented Lagrangian with respect to $\boldsymbol{W}^{(3)}$, which amounts to the minimization

$$\underset{\boldsymbol{W}}{\text{minimize}} \ \frac{\rho}{2}\left\|\boldsymbol{W}^{(1)}_{k+1} + \boldsymbol{U}^{(1)}_k - \boldsymbol{W}^\top \boldsymbol{X}^{in}\right\|_F^2 + \frac{\rho}{2}\left\|\boldsymbol{W}^{(2)}_{k+1} + \boldsymbol{U}^{(2)}_k - \boldsymbol{W}\right\|_F^2.$$

While the updates for $\boldsymbol{W}^{(1)}$ and $\boldsymbol{W}^{(2)}$, as in (5.3) and (5.4), are stated in the general form, they can be further simplified and presented in closed form. To this end, a first observation is that (5.3) can be decoupled into independent minimizations in terms of $\boldsymbol{W}_\Omega$ and $\boldsymbol{W}_{\Omega^c}$, i.e.,

$$\begin{aligned} \boldsymbol{W}^{(1)}_{k+1} = &\underset{\boldsymbol{W}_\Omega:\, \left\|\boldsymbol{W}_\Omega - \boldsymbol{X}^{out}_\Omega\right\|_F \le \epsilon}{\arg\min} \frac{\rho}{2}\left\|\boldsymbol{W}_\Omega + \left(\boldsymbol{U}^{(1)}_k - {\boldsymbol{W}^{(3)}_k}^\top \boldsymbol{X}^{in}\right)_\Omega\right\|_F^2 \\ &+ \underset{\boldsymbol{W}_{\Omega^c}:\, \boldsymbol{W}_{\Omega^c} \le \boldsymbol{V}_{\Omega^c}}{\arg\min} \frac{\rho}{2}\left\|\boldsymbol{W}_{\Omega^c} + \left(\boldsymbol{U}^{(1)}_k - {\boldsymbol{W}^{(3)}_k}^\top \boldsymbol{X}^{in}\right)_{\Omega^c}\right\|_F^2. \end{aligned} \tag{5.6}$$

The first minimization on the right-hand side of (5.6) is basically the problem of finding the closest point of an $\epsilon$-radius Euclidean ball to a given point. For the nontrivial case that the given point is outside the ball, the solution is the intersection of the ball surface with the line connecting the point to the center of the ball. More specifically, for fixed $\boldsymbol{Y}$ and $\boldsymbol{Z}$,

$$\underset{\boldsymbol{W}_\Omega:\ \|\boldsymbol{W}_\Omega-\boldsymbol{Z}_\Omega\|_F\le\epsilon}{\arg\min}\ \frac{\rho}{2}\left\|\boldsymbol{W}_\Omega-\boldsymbol{Y}_\Omega\right\|_F^2=\begin{cases}\boldsymbol{Y}_\Omega & \text{if } \left\|\boldsymbol{Y}_\Omega-\boldsymbol{Z}_\Omega\right\|_F\le\epsilon,\\ \boldsymbol{Z}_\Omega+\epsilon\frac{\boldsymbol{Y}_\Omega-\boldsymbol{Z}_\Omega}{\|\boldsymbol{Y}_\Omega-\boldsymbol{Z}_\Omega\|_F} & \text{else.}\end{cases}$$

The second term in (5.6) is an instance of a projection onto an orthant and can be delivered in closed form as

$$\underset{\boldsymbol{W}_{\Omega^c}:\ \boldsymbol{W}_{\Omega^c}\le\boldsymbol{V}_{\Omega^c}}{\arg\min}\ \frac{\rho}{2}\left\|\boldsymbol{W}_{\Omega^c}-\boldsymbol{Y}_{\Omega^c}\right\|_F^2=\boldsymbol{Y}_{\Omega^c}-\left(\boldsymbol{Y}_{\Omega^c}-\boldsymbol{V}_{\Omega^c}\right)^+.$$

Finally, the solution to (5.4) is the standard soft thresholding operator (e.g., see section 4.4.3 of [4]), which reduces the update to

$$\left(\boldsymbol{W}_{k+1}^{(2)}\right)_{n,m}=S_{1/\rho}\left(\left(\boldsymbol{W}_k^{(3)}-\boldsymbol{U}_k^{(2)}\right)_{n,m}\right),\ \text{where}\ \ S_c(w)=\begin{cases}w-c, & w>c,\\ 0, & |w|\le c,\\ w+c, & w<-c.\end{cases}$$

After combining the steps above, we propose Algorithm 5.1 as a computational scheme to address the Net-Trim central program.

**Algorithm 5.1.** Implementation of the Net-Trim central program.

**input:** $\boldsymbol{X}^{in}\in\mathbb{R}^{N\times P}$, $\boldsymbol{X}^{out}\in\mathbb{R}^{M\times P}$, $\Omega$, $\boldsymbol{V}_\Omega$, $\epsilon$, $\rho$
**initialize:** $\boldsymbol{U}^{(1)},\boldsymbol{U}^{(2)}$ and $\boldsymbol{W}^{(3)}$ `% all initializations can be with 0`
$\boldsymbol{C}\leftarrow\boldsymbol{X}^{in}\boldsymbol{X}^{in\top}+\boldsymbol{I}$
**while** not converged **do**
  $\boldsymbol{Y}\leftarrow\boldsymbol{W}^{(3)\top}\boldsymbol{X}^{in}-\boldsymbol{U}^{(1)}$
  **if** $\left\|\boldsymbol{Y}_\Omega-\boldsymbol{X}_\Omega^{out}\right\|_F\le\epsilon$ **then**
    $\boldsymbol{W}_\Omega^{(1)}\leftarrow\boldsymbol{Y}_\Omega$
  **else**
    $\boldsymbol{W}_\Omega^{(1)}\leftarrow\boldsymbol{X}_\Omega^{out}+\epsilon\left\|\boldsymbol{Y}_\Omega-\boldsymbol{X}_\Omega^{out}\right\|_F^{-1}\left(\boldsymbol{Y}_\Omega-\boldsymbol{X}_\Omega^{out}\right)$
  **end if**
  $\boldsymbol{W}_{\Omega^c}^{(1)}\leftarrow\boldsymbol{Y}_{\Omega^c}-\left(\boldsymbol{Y}_{\Omega^c}-\boldsymbol{V}_{\Omega^c}\right)^+$
  $\boldsymbol{W}^{(2)}\leftarrow S_{1/\rho}(\boldsymbol{W}^{(3)}-\boldsymbol{U}^{(2)})$ `% $S_{1/\rho}$ applies to each element of the matrix`
  $\boldsymbol{W}^{(3)}\leftarrow\boldsymbol{C}^{-1}(\boldsymbol{X}^{in}(\boldsymbol{W}^{(1)}+\boldsymbol{U}^{(1)})^\top+\boldsymbol{W}^{(2)}+\boldsymbol{U}^{(2)})$
  $\boldsymbol{U}^{(1)}\leftarrow\boldsymbol{U}^{(1)}+\boldsymbol{W}^{(1)}-\boldsymbol{W}^{(3)\top}\boldsymbol{X}^{in}$
  $\boldsymbol{U}^{(2)}\leftarrow\boldsymbol{U}^{(2)}+\boldsymbol{W}^{(2)}-\boldsymbol{W}^{(3)}$
**end while**
**return** $\boldsymbol{W}^{(3)}$

To analyze the computational complexity of Algorithm 5.1, note that $\boldsymbol{C}=\boldsymbol{X}^{in}\boldsymbol{X}^{in\top}+\boldsymbol{I}$ and its Cholesky factorization, which together require $\mathcal{O}(N^3)+\mathcal{O}(N^2P)$ flops, need only be calculated once and then used inside the ADMM loop for the linear solves (5.5). Within each ADMM iteration, the main computational steps are the matrix products $\boldsymbol{W}^{(3)\top}\boldsymbol{X}^{in}$ and

$\boldsymbol{X}^{in}(\boldsymbol{W}^{(1)}+\boldsymbol{U}^{(1)})^\top$ which both take $\mathcal{O}(MNP)$ flops, and the $\boldsymbol{W}^{(3)}$ update via the linear solve (5.5), which can be done in $\mathcal{O}(N^2M)$ flops. All of these basic calculations are computationally distributable and can be done in parallel when pruning large networks associated with big data.

**5.1. Net-Trim for convolutional layers.** Since the convolution operator is linear, similar steps as those above can be taken to implement a version of Net-Trim for convolutional layers and inputs in the form of tensors. The main difference is addressing the least-squares update in (5.5), which can be performed by incorporating the adjoint operator. The details of implementing Net-Trim for convolutional layers are presented in section SM2 of the Supplementary Material.

**6. Experiments and remarks.** While the main purpose of this paper is introducing a theoretical framework for a class of pruning techniques in deep learning, we present some experiments which highlight the performance of Net-Trim in real-world problems. Due to space limitation, most details of the simulations along with additional experiments (such as the application of Net-Trim in the $P < N$ regime) are presented in section SM3 of the Supplementary Material. Also Net-Trim implementation is made publicly available online.[7]

Our first set of experiments corresponds to a comparison between the cascade and parallel frameworks. For this experiment we use a fully connected (FC) neural network of size $784 \times 300 \times 1000 \times 100 \times 10$ (composed of four layers), trained to classify the MNIST (Modified National Institute of Standards and Technology) dataset. Throughout the section we refer to this network as the FC model. While the theory supports retraining the network with new samples, in practice Net-Trim can be applied to the dataset used to train the original network. In this experiment we also assess the possibility of applying Net-Trim to only a portion of the training data (i.e., working with a subset of columns in $\boldsymbol{X}$). Clearly, working with smaller $\boldsymbol{X}$ matrices is computationally more desirable.

Figure 3 summarizes the parallel and cascade pruning results, where for the experiments working with a subset of the columns in $\boldsymbol{X}$, we report the average quantities from ten different subsample realizations. A quick comparison between the range of relative discrepancies in panels (a) and (c) (calculated as $\|\hat{\boldsymbol{X}}^{(L)} - \boldsymbol{X}^{(L)}\|_F / \|\boldsymbol{X}^{(L)}\|_F$) reveals that for more or less similar sparsity rates, cascade Net-Trim produces a smaller overall discrepancy compared to the parallel scheme (note the axis ranges). This may be considered as the return for going through a nondistributable scheme. However, a comparison of the test accuracies in panels (b) and (d), and especially for larger values of the sparsity ratio, shows a less significant difference between the test accuracies of the two schemes; specifically that using the parallel scheme and its distributable nature is more desirable for big data.

Our next set of experiments corresponds to the application of Net-Trim to the LeNet convolutional network [15] to highlight its performance against the well-established regularizing methods of Dropout and $\ell_1$ penalty. In these experiments the mean test accuracy and initial model sparsity are reported for the cases of Dropout, $\ell_1$ penalty, and a combination of both. For each run, the tuning parameters ($\lambda$: the coefficient of $\ell_1$ penalty, $p$: the Dropout probability, or both) are varied in a range of values and the mean quantities are reported. It is noteworthy that Net-Trim can always be followed by an optional fine-tuning step (FT), which performs few

[7] To access the algorithm implementation, visit https://dnntoolbox.github.io/Net-Trim/.

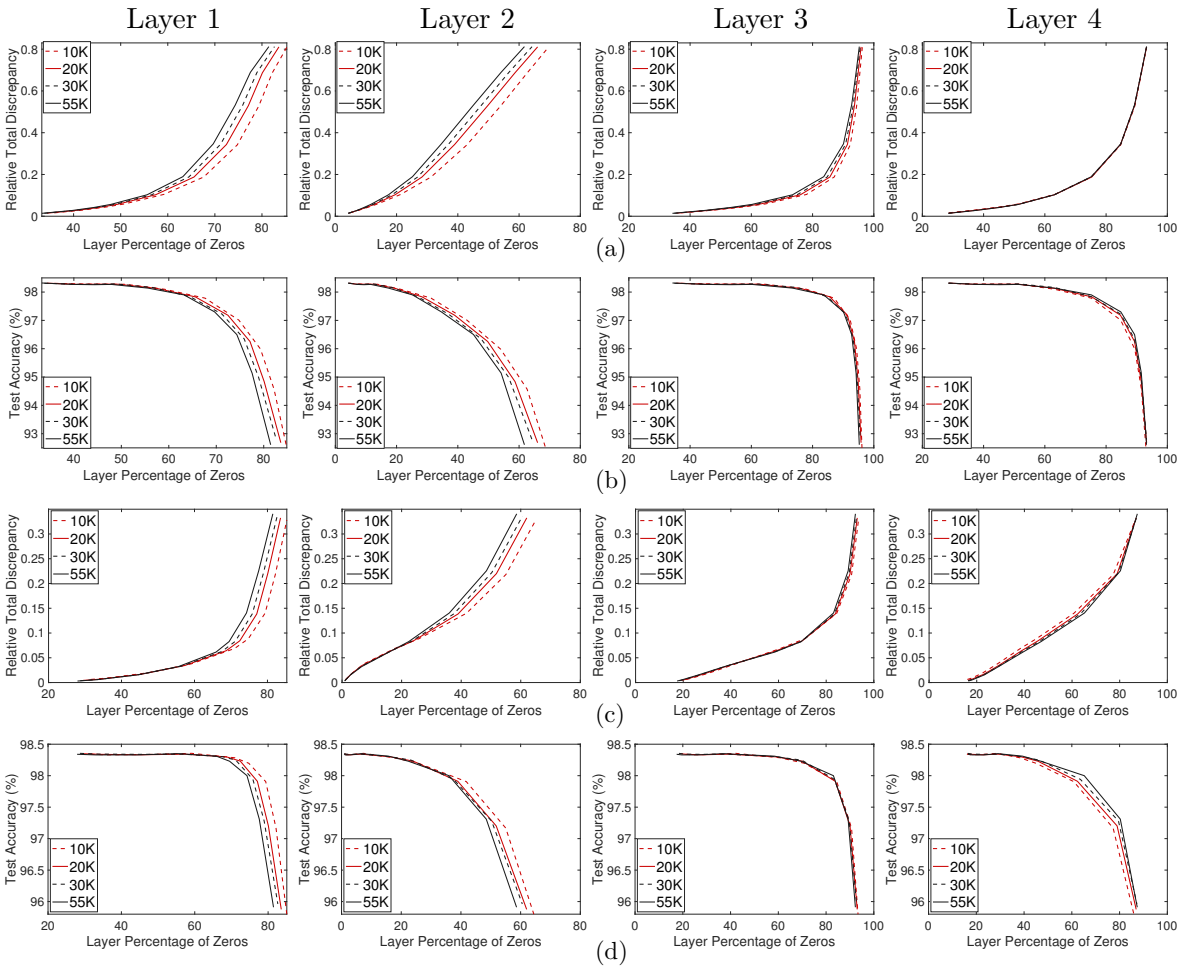



**Figure 3.** *Retraining the FC network that is initially trained with the full MNIST training set and retrained with* $10K$, $20K$, $30K$, *and* $55K$ *samples (each column corresponds to a layer);* (a) *network relative total discrepancy (RTD) versus the layer percentage of zeros (LPZ) after a parallel scheme;* (b) *test accuracy versus LPZ after a parallel scheme;* (c) *RTD versus LPZ after a cascade scheme;* (d) *test accuracy versus LPZ after a cascade scheme.*

training iterations on the weights that Net-Trim has left nonzero. The plots in Figure 4 show how the application of Net-Trim can further contribute to the sparsity and accuracy of the network. For instance, panel (c) indicates that without a loss of accuracy, applying Net-Trim to a network, where almost 88% of the weights are pruned via Dropout and $\ell_1$ regularization, can elevate the sparsity to almost 98%.

While the application of the $\ell_1$ regularizer during the training can control the sparsity of the trained models to some extent, our observation is that fully controlling the final level of sparsity via an $\ell_1$ penalty is not possible. Since the neural network training cost is a complex and nonconvex function, adding an $\ell_1$ penalty and performing the minimization does not necessarily present a consistent behavior in response to the value of the penalty parameter. In

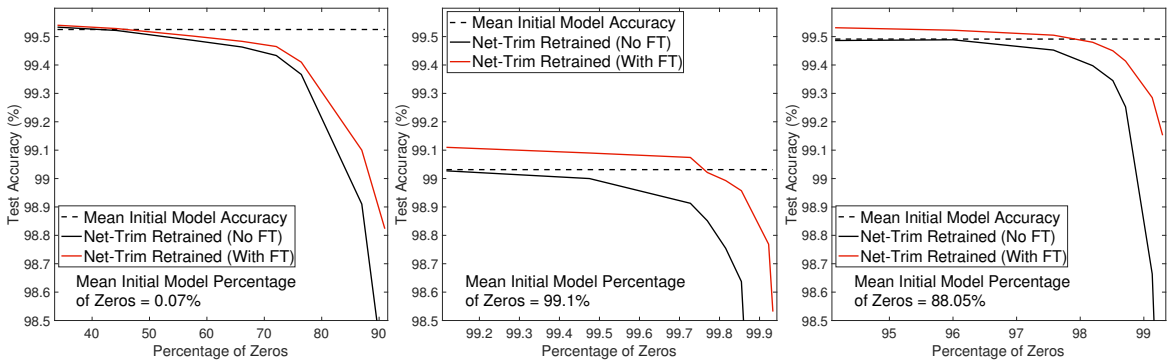



**Figure 4.** *Mean test accuracy versus mean model sparsity after the application of Net-Trim to the LeNet network initially regularized via $\ell_1$ penalty, Dropout, or both (the regularization parameter and Dropout probability are picked from a range of values and the mean accuracy and sparsity are reported);* (a) *a model trained with Dropout only:* $0.3 \leq p \leq 0.8$*;* (b) *a model trained with $\ell_1$ penalty only:* $10^{-5} \leq \lambda \leq 5 \times 10^{-3}$*;* (c) *a model trained with Dropout and $\ell_1$:* $10^{-5} \leq \lambda \leq 2 \times 10^{-4}$*,* $0.5 \leq p \leq 0.75$*.*

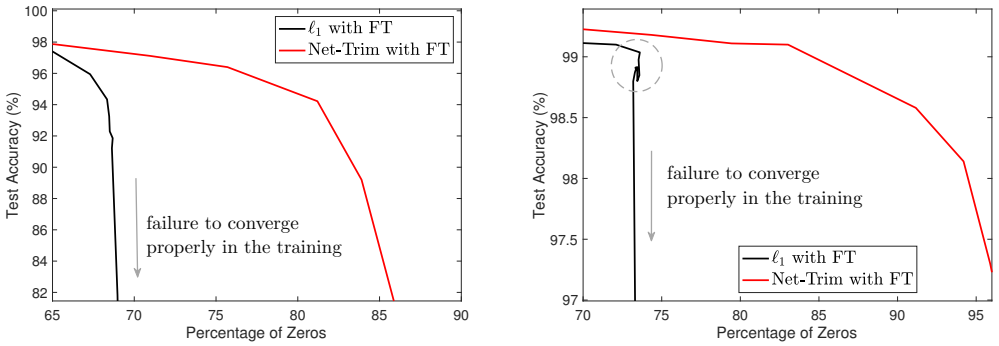



**Figure 5.** *A comparison of the $\ell_1$ pruning during the training followed by a post-fine-tuning step (30 epochs), and Net-Trim with fine-tuning (similar number of epochs); the value of the $\ell_1$ penalty is varied between* $10^{-4}$ *and* $0.02$*;* (a) *FC network;* (b) *LeNet network.*

Figure 5 we have presented an experiment, where a sparse model is generated by the application of an $\ell_1$ penalty during the training, and is later fine-tuned to improve the accuracy. We also train a model without a regularizer, and only perform the Net-Trim as a postprocessing step. The plots reveal that there is a major gain in the sparsity of the models when Net-Trim is applied. Also, while the values presented in panels (a) and (b) are the mean values from ten different experiments, one can observe that the level of sparsity in an $\ell_1$ framework is not a coherent response to the value of the $\ell_1$ penalty (see the circled region in panel (b)). That is, when the $\ell_1$ penalty is increased, the training minimization can get trapped in a stationary point that is not necessarily sparser, while thanks to the convexity of the Net-Trim objective, a monotonic and controlled increase of the sparsity is possible by increasing the discrepancy level, $\epsilon$. We also observe that after some point, increasing the value of the $\ell_1$ penalty prohibits the gradient descent scheme from locating a proper stationary point, and the model accuracy

significantly drops. That is the reason behind the fast drop of the $\ell_1$ accuracy curves in panels (a) and (b).

Another well-known scheme in model pruning is the algorithm by Han, Pool, Tran, and Dally (HPTD) [11]. The HPTD algorithm is a heuristic tool used for network compression, which truncates the small weights across a trained network and performs another round of training on the active weights (same as the fine-tuning scheme explained above). Using tools such as quantization and Huffman coding, more advanced frameworks such as the Deep Compression [10] have been developed later. However, the focus of those works is mainly compressing the network parameters on the memory, and HPTD pruning scheme is yet the most relevant single-module framework that could be compared with the Net-Trim.

Figure 6 presents a comparison between the Net-Trim and HPTD on the FC and LeNet models. For the Net-Trim we use different values of $\epsilon$ to prune the trained networks. To compare the method with the HPTD, after each application of the Net-Trim and counting the number of zeros, the same number of elements are truncated from the initial network to be used for the HPTD implementation. HPTD is followed by a fine-tuning step after the truncation, which is also an optional task for Net-Trim. Nevertheless, both algorithms are compared without fine-tuning, or with fine-tuning using 10 or 30 epochs. The left plots in panels (a) and (b) show that in all scenarios Net-Trim outperforms HPTD in generating more accurate models when the levels of sparsity are matched. The middle plots also show the improvements in the accuracy as a function of the number of epochs required in the fine-tuning process for the two schemes. In both scenarios, Net-Trim requires only few epochs to achieve the top accuracy, while achieving such level of accuracy for the HPTD is either not feasible or takes many fine-tuning epochs.

Next, we present a comparison between the Net-Trim and HPTD on a larger model. We use the CIFAR-10 model trained on an augmented training set of 6.4M samples (the reader is referred to the Supplementary Material for more details about the network architecture and the data augmentation process). For this relatively large dataset we also go through the exercise of retraining the Net-Trim with only part of the training samples, specifically 25K, 50K, and 75K samples of the entire 6.4M training set. A similar set of comparisons between the Net-Trim and the HPTD (as in Figure 6) is presented in Figure 7, noting that the fine-tuning step for both schemes is carried out using all the 6.4M training samples. Similar to the previous experiment, Net-Trim consistently outperforms HPTD in all similar setups.

Aside from such superiority, we highlight the possibility of retraining Net-Trim using only part of the training samples. For instance, a comparison of panels (a) and (b) shows that almost identical results can be achieved in terms of accuracy versus sparsity, when Net-Trim is solved with 25K samples instead of 50K samples. For instance, for both panels, a Net-Trim application followed by a single fine-tuning step can increase the percentage of the zeros in the network to more than 80%, with almost no loss in the model accuracy. Basically, as also discussed previously with reference to Figure 3, for large datasets formulating the Net-Trim with only a portion of the data can be considered as a general computation shortcut.

Another interesting observation, which is more apparent in the left plot of panel (c), is that fine-tuning does not always improve the accuracy of the models after the application of Net-Trim, and especially in low pruning regimes may degrade the accuracy due to phenomena such as overfitting. For example, in panel (c), up to a pruning percentage of almost 65%, a

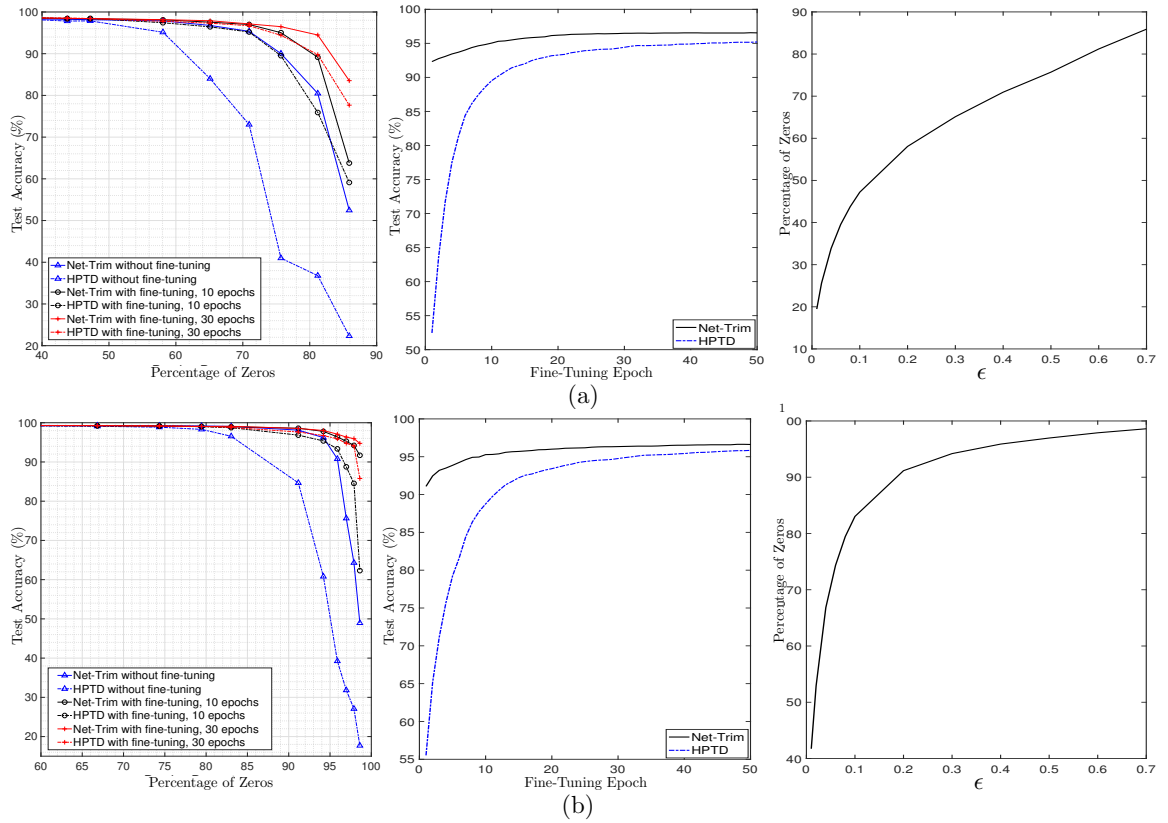



**Figure 6.** *Comparison of Net-Trim and HPTD in different settings for* (a) *FC model,* (b) *LeNet model; the left panels compare Net-Trim and HPTD test accuracy versus percentage of zeros, without fine-tuning, and with fine-tuning using* 10 *and* 30 *epochs; middle panels show the number of fine-tuning epochs and the acquired accuracy using Net-Trim and HPTD; the right panels indicate the percentage of zeros as a function* $\epsilon$ *for Net-Trim.*

fine-tuning step after the Net-Trim slightly degrades the accuracy. While a fine-tuning step is likely to help in the majority of cases, our access to both Net-Trim's plain outcome and the fine-tuned version provides the flexibility of picking the most compressed and accurate model among the two.

We would also like to make a note on the processing times of these networks using the Net-Trim. Thanks to the proposed ADMM scheme, which mainly uses basic operations within each iteration, an application of the Net-Trim to a network like MNIST takes less than a few minutes on a standard desktop computer. More specifically, applying the Net-Trim to each layer of the MNIST networks is on the order of 20 seconds, and the fine-tuning step is on the order of 30 seconds. For the CIFAR-10 network, the Net-Trim application to each layer is about 1 minute, and the fine-tuning step takes about 20–30 minutes, depending on the sparsity of the network.

From a technical standpoint, one of the main drawbacks with the HPTD is the truncation based on the magnitude of the weights, which in many cases may discard connections to the important features and variables in the network. That is mainly the reason that Net-Trim

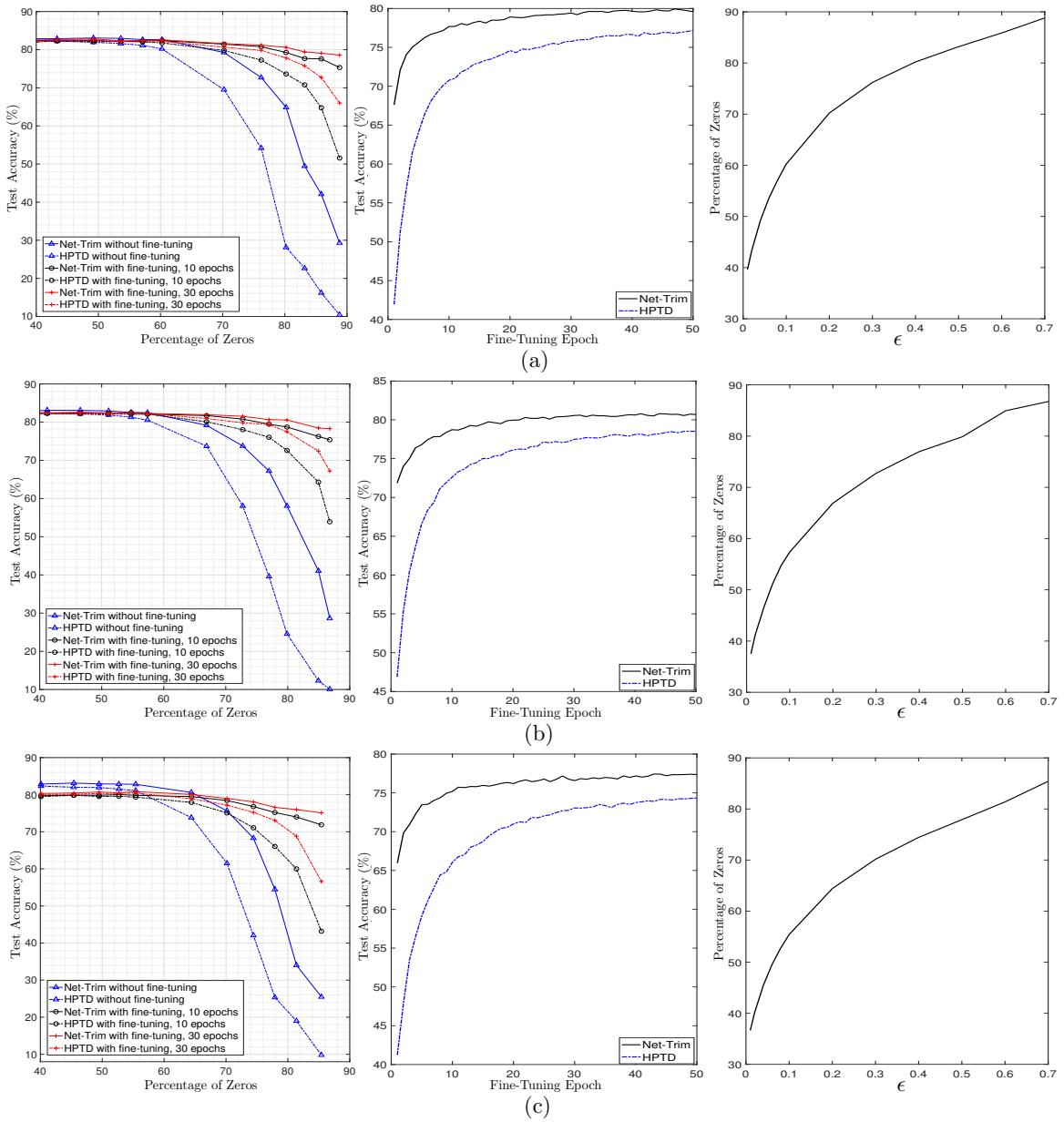



**Figure 7.** *Similar comparison plots as in Figure 6 for CIFAR-10:* (a) *retraining of Net-Trim and HPTD performed using* $25K$ *samples;* (b) *retraining performed using* $50K$ *samples;* (c) *retraining performed using* $75K$ *samples.*

consistently outperforms this method. In fact, Net-Trim can also present vital information about the data structure and important features that are not immediately available using other techniques. In Figure 8 we have depicted the retrained $\hat{\boldsymbol{W}}_1$ matrix of the FC model after applying Net-Trim and HPTD. In panel (b) we can see many columns that are fully zero. After plotting the histogram of the MNIST samples (as in panel (d)), one would immediately

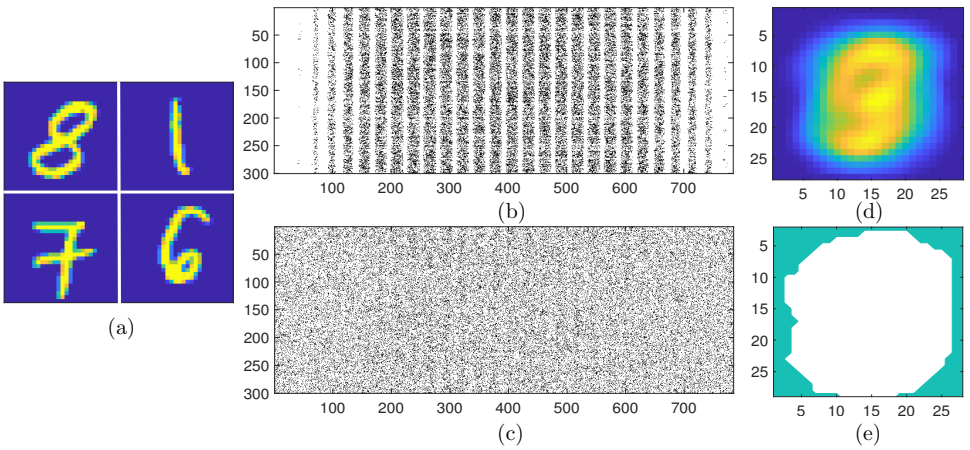



**Figure 8.** *Instant identification of important features using Net-Trim;* (a) *samples from MNIST dataset;* (b) *visualization of* $\hat{\boldsymbol{W}}_1{}^\top$ *in the FC retrained model using Net-Trim;* (c) *similar visualization of* $\hat{\boldsymbol{W}}_1{}^\top$ *using HPTD with a single fine-tuning step;* (d) *histogram of the pixel values in the MNIST data;* (e) *the green mask corresponding to the zero columns in panel* (b).

observe that the zero columns in $\hat{\boldsymbol{W}}_1$ correspond to the boundary pixels with the least level of information. As HPTD only relies on the truncation based on the weight magnitudes, despite the similar number of zeros in panels (b) and (c), the latter does not highlight such data structure. To obtain a similar pattern as panel (b), the authors in [11] suggest an iterative scheme, where instead of a single truncation/fine-tuning, they run a series of truncations each followed by a fine-tuning step. This is not a computationally efficient path as it requires retraining the network multiple times, which can take a lot of time for large datasets and is not guaranteed to identify the right structures.

Finally, we compare the Net-Trim with the dynamic network surgeon (DNS) [9] and Bayesian compression (BC) [16] frameworks, which are both among the most well-known and powerful techniques in pruning neural networks during the course of training. In Figure 9, we have presented a comparison between the three schemes. To have meaningful results, we ran each scheme more than a hundred times to obtain a rich set of pointwise relations between the accuracy and sparsity for each scheme, and then grouped the point clusters by their mean. We observe that Net-Trim clearly outperforms the BC scheme and presents a comparable performance with the DNS. It is, however, worth noting that both DNS and BC are applied during the course of training and they both require learning an increased number of parameters (for example, DNS requires training twice the number of parameters in the model). For that reason, they require a nonconventional training, while Net-Trim is a postprocessing scheme which operates with the original model parameters and works regardless of how the network is trained.

**6.1. Concluding remarks.** Net-Trim can be generalized to a large class of problems, where the architecture of each layer in a trained network is restructured via a program of the type

$$\underset{\boldsymbol{U}\in\mathbb{R}^{N\times M}}{\text{minimize}} \quad \mathcal{R}(\boldsymbol{U}) \qquad \text{subject to} \qquad \sigma\left(\boldsymbol{U}^\top \boldsymbol{X}^{in}\right) \approx \boldsymbol{X}^{out}. \tag{6.1}$$

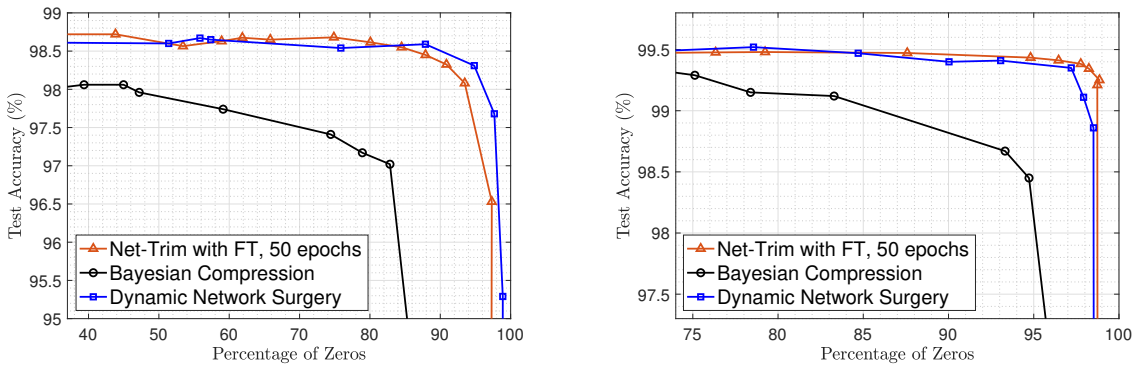



**Figure 9.** *A comparison of Net-Trim with the DNS and BC;* (a) *FC network;* (b) *LeNet network.*

The objective $\mathcal{R}(\cdot)$ aims to promote a desired structure, and the constraint enforces a consistency between the initial and retrained models. While in this paper we merely emphasized on $\mathcal{R}(\boldsymbol{U}) = \|\boldsymbol{U}\|_1$, a variety of other structures may be explored by adaptively selecting the objective. For instance, other than the Ridge and the elastic net penalties as regularizing tools, choosing $\mathcal{R}(\boldsymbol{U}) = \|\boldsymbol{U}\|_{2,1} = \sum_{n=1}^{N} \|\boldsymbol{U}_{n,:}\|$ can promote selection of a subset of the rows in $\boldsymbol{X}^{in}$, and act as a feature selection or node-dropping tool for each layer. Total variation or rank penalizing objectives may also directly apply to network compression problems.

While in this paper we specifically focused on $\sigma = \texttt{ReLU}(\cdot)$ to exploit the convex formulation, other forms of activation may be explored. Even if a convex (re)formulation is suboptimal or not possible, powerful tools from nonconvex analysis would still allow us to have an understanding of when and how well programs of type (6.1) work. Clearly, the techniques used for such type of analysis might be initialization-sensitive and different from those used in this paper.

Net-Trim can specifically become a useful tool when the number of training samples is limited. While overfitting is likely to happen in this situation, Net-Trim allows reducing the redundancy of the models, yet maintaining the consistency with the original model. From a different perspective, Net-Trim may simplify the process of determining the network size. For large networks that are trained with insufficient samples, employing Net-Trim can reduce the size of the models to an order matching the data.

## REFERENCES

[1] J. Achterhold, J. M. Koehler, A. Schmeink, and T. Genewein, *Variational network quantization*, in Proceedings of the International Conference on Learning Representations (ICLR), 2018, https://openreview.net/forum?id=ry-TW-WAb.

[2] A. Aghasi, A. Abdi, N. Nguyen, and J. Romberg, *Net-trim: Convex pruning of deep neural networks with performance guarantee*, in Advances in Neural Information Processing Systems 31, Curran Associates, Red Hook, NY, 2017, pp. 3177–3186.

[3] D. Amelunxen, M. Lotz, M. B. McCoy, and J. A. Tropp, *Living on the edge: Phase transitions in convex programs with random data*, Inf. Inference, 3 (2014), pp. 224–294.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed optimization and statistical learning via the alternating direction method of multipliers*, Found. Trends Mach. Learn., 3 (2011),

pp. 1–122.
[5] E. Candes, J. Romberg, and T. Tao, *Stable signal recovery from incomplete and inaccurate measurements*, Comm. Pure Appl. Math., 59 (2006), pp. 1207–1223.
[6] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, *Compressing neural networks with the hashing trick*, in Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 2285–2294.
[7] F. Girosi, M. Jones, and T. Poggio, *Regularization theory and neural networks architectures*, Neural Comput., 7 (1995), pp. 219–269.
[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, Cambridge, MA, 2016.
[9] Y. Guo, A. Yao, and Y. Chen, *Dynamic network surgery for efficient DNNs*, in Advances In Neural Information Processing Systems, 2016, pp. 1379–1387.
[10] S. Han, H. Mao, and W. J. Dally, *Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding*, in Proceedings of the International Conference on Learning Representations (ICLR), 2016.
[11] S. Han, J. Pool, J. Tran, and W. Dally, *Learning both weights and connections for efficient neural network*, in Proceedings of the 28th International Conference on Neural Information Processing Systems, 2015, pp. 1135–1143.
[12] A. Hoerl and R. Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics, 12 (1970), pp. 55–67.
[13] D. Hsu, S. Kakade, and T. Zhang, *A tail inequality for quadratic forms of subgaussian random vectors*, Electron. Commun. Probab., 17 (2012), 52.
[14] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, in Proceedings of the 32nd International Conference on Machine Learning, 2015, pp. 448–456.
[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proc. IEEE, 86 (1998), pp. 2278–2324.
[16] C. Louizos, K. Ullrich, and M. Welling, *Bayesian compression for deep learning*, in Advances in Neural Information Processing Systems, 2017, pp. 3288–3298.
[17] S. Mendelson, *Learning without concentration*, in Proceedings of the 27th Annual Conference on Learning Theory, 2014, pp. 25–39.
[18] D. Molchanov, A. Ashukha, and D. Vetrov, *Variational dropout sparsifies deep neural networks*, in Proceedings of the 34th International Conference on Machine Learning, Volume 70, PMLR, 2017, pp. 2498–2507.
[19] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, *Structured Bayesian pruning via log-normal multiplicative noise*, in Advances in Neural Information Processing Systems, 2017, pp. 6775–6784.
[20] S. Nowlan and G. Hinton, *Simplifying neural networks by soft weight-sharing*, Neural Comput., 4 (1992), pp. 473–493.
[21] J. Schmidhuber, *Deep learning in neural networks: An overview*, Neural Networks, 61 (2015), pp. 85–117.
[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, J. Mach. Learn. Res., 15 (2014), pp. 1929–1958.
[23] M. Talagrand, *The Generic Chaining: Upper and Lower Bounds of Stochastic Processes*, Springer-Verlag, Berlin, Heidelberg, 2006.
[24] R. Tibshirani, *Regression shrinkage and selection via the lasso*, J. Roy. Statist. Soc. Ser. B, 58 (1996), pp. 267–288.
[25] J. Tropp, *Convex recovery of a structured signal from independent random linear measurements*, in Sampling Theory, a Renaissance, Springer, Cham, 2015, pp. 67–101.
[26] A. van der Vaart and J. Wellner, *Weak Convergence and Empirical Processes: With Applications to Statistics*, Springer-Verlag, New York, 1996.
[27] R. Vershynin, *Introduction to the non-asymptotic analysis of random matrices*, in Compressed Sensing: Theory and Applications, Cambridge University Press, Cambridge, UK, 2012, pp. 210–268, https://doi.org/10.1017/CBO9780511794308.006.
[28] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus, *Regularization of neural networks using DropConnect*, in Proceedings of the 30th International Conference on Machine Learning, 2013.