Compressing Deep Neural Networks With Sparse Matrix Factorization

Kailun Wu, Yiwen Guo, and Changshui Zhang, *Fellow, IEEE*

***Abstract*— Modern deep neural networks (DNNs) are usually overparameterized and composed of a large number of learnable parameters. One of a few effective solutions attempts to compress DNN models via learning sparse weights and connections. In this article, we follow this line of research and present an alternative framework of learning sparse DNNs, with the assistance of matrix factorization. We provide an underlying principle for substituting the original parameter matrices with the multiplications of highly sparse ones, which constitutes the theoretical basis of our method. Experimental results demonstrate that our method substantially outperforms previous states of the arts for compressing various DNNs, giving rich empirical evidence in support of its effectiveness. It is also worth mentioning that, unlike many other works that focus on feedforward networks like multi-layer perceptrons and convolutional neural networks only, we also evaluate our method on a series of recurrent networks in practice.**

***Index Terms*— Deep neural networks (DNNs), memory efficiency, network compression, sparse representation.**

I. Introduction

In recent years, deep neural networks (DNNs) have achieved remarkable success in research areas of artificial intelligence including computer vision and natural language processing. Convolution neural networks (CNNs) [1]–[3] and long short-term memory (LSTM) networks [4], [5] have become de facto choices for specific visual recognition and sequence modeling tasks. For some other kinds of tasks [6]–[8], other types of networks with different architectures are also exploited to solve them. In general, DNNs can be represented as learnable functions with a large number of parameters that are trained to fit reasonable mappings from the input to output data subspaces. Though undesirable in real-world applications [9], it is broadly believed that their success comes at the cost of overparameterization and, consequently, large computational complexity and memory complexity [10]. The reduction in the number of parameters will alleviate the storage and computation burden, which is beneficial to applying DNNs in more applications that are practical.

Many previous works have been done in the field of network compression. Early works apply a progressive pruning process to sparsify the connection of DNNs [9], [11] and can achieve surprisingly high compression rates on some networks. However, these methods are sensitive to inappropriate pruning in the beginning, since they seem to lack "fault-tolerant" designs. Stemming from this point, Guo *et al.* [12] then proposed to combine pruning with a complementary operation called "splicing" that would relieve the issue. Except for network pruning, there exist a spectrum of other methods that use regularization techniques to generate sparse DNNs, like l_1 penalty and group lasso [13]. These methods can not only make matrices/tensors sparse but also reduce the number of neurons in a well-trained DNN. For instance, Wen *et al.* [14] leverage the group lasso technique to learn a variety of configurations of network models, such as the number of filters or even the number of layers. In addition, one may explore the idea of (empirical) Bayes, which explicitly fashions the *a priori* knowledge of the learnable parameters [15]–[18] and can be regarded as a regularization strategy as well. Typically, the parameters are sampled from a presumed distribution that aligns with efficient encodings when maximizing *a posteriori* probability. In comparison with the others [16], these methods normally require less complex training pipelines and may also involve less manually tuned hyperparameters.

Orthogonal to the progress made on sparsifying DNNs [9], [11], [12], [14], [16]–[20], some other research works [21]–[24] choose to adopt (low-rank) matrix/tensor factorization techniques and exploit cross-filter redundancies for compressions. Even the multiplication of multiple (>2) matrices/tensors can be applied [25]. We refer the interested reader to recent survey articles [10], [26] for more information on this line of research.

In this article, we present a simple yet principled framework that applies a sparse matrix factorization technique to take advantage of the distinct methodologies. Concretely, we advocate using the multiplication of sparse matrices as a substitute for any overparameterized representation in DNNs, in which one can optionally impose tight low-rank constraints for further compressions. Some empirical motivations are presented in Fig. 1. It depicts that the total percentage of zero entries (which reasonably indicates the memory redundancy

Manuscript received December 17, 2018; revised February 22, 2019 and July 7, 2019; accepted October 1, 2019. This work was supported in part by the NSFC under Grant 61876095 and Grant 61751308 and in part by the Beijing Academy of Artificial Intelligence (BAAI). *(Corresponding author: Changshui Zhang.)*

K. Wu and C. Zhang are with the State Key Laboratory of Intelligent Technologies and Systems, Beijing National Research Center for Information Science and Technology (BNRist), Institute for Artificial Intelligence (THUAI), Tsinghua University, Beijing 100084, China, and also with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: wukl14@mails.tsinghua.edu.cn; zcs@mail.tsinghua.edu.cn).

Y. Guo is with the Cognitive Computing Lab, Intel Labs, Beijing 100190, China, with the State Key Laboratory of Intelligent Technologies and Systems, Beijing National Research Center for Information Science and Technology (BNRist), Institute for Artificial Intelligence (THUAI), Tsinghua University, Beijing 100084, China, and also with the Department of Automation, Tsinghua University, Beijing 100084, China (e-mail: yiwen.guo@intel.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2946636

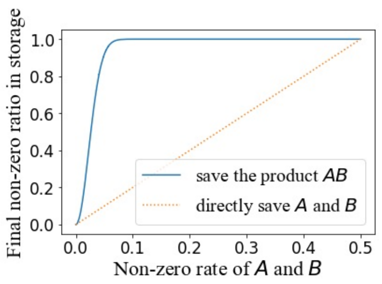


Fig. 1. Compare the storage requirement of two 1000×1000 sparse matrices (i.e., A and B) and their product (i.e., AB) by illustrating the relationship between the percentage of nonzero entries in the two matrices and in storage. We show that it can be much more efficient to substitute the matrix with two sparse ones (if possible) than storing it as normal.

and, to some extent, the computational overhead to be properly eliminated) of two sparse square matrices in storage is substantially higher than the percentage of zero (and redundant) entries of their product in practical cases, which partly motivates us to explore the possibility of DNN compression with sparse matrix factorization. Within our framework, compression ingredients can be flexibly leveraged and combined. In one aspect, we anticipate gaining higher sparsity with the assistance of matrix factorization, regardless of whether the matrices are full rank or not, while in another aspect, further compressions can be obtained with probable low-rank constraints in our framework. Moreover, low-precision [27] and even ternary constraints (see [28], [29]) can also be readily incorporated if necessary.

In Section II-B, we will introduce our framework from a unified perspective of Bayesian inference and maximum *a posteriori*. We will provide theoretical guarantees that the benefit of using factorized sparse matrices is theoretically close to that of applying a single matrix with (l_2) weight decay, which guarantees the prediction accuracy of our compressed models and derives an easy-to-implement formulation for it. Our method is superior to the previous state-of-the-art pruning methods by large margins. The main contributions of our article are listed as follows.

1) We present a novel and alternative framework for compressing DNNs with sparse matrix factorization, which is amenable to theoretical analysis.
2) We study possible behaviors of the sparse regularizations from a theoretic point of view. We prove that the distribution of the entries of the multiplication well approximates the Gaussian distribution.
3) The low-rank and low-precision constraints can be readily incorporated into our framework if necessary. We conduct experiments on the two scenarios and compare the results qualitatively.
4) Unlike many previous works that focus mostly on the CNNs, we implement this framework with different regularizations on various DNNs including multi-layer perceptrons (MLPs), CNNs, LSTMs, and several other types of recurrent networks. We obtain the state-of-the-art compression results (CRs) under comparable accuracy loss.

II. Our Method

A. Notation

We first introduce the notations. In the rest of this article, we will refer to $F(y|x; \{W_i\})$, the forward mapping of a DNN, with input x and output y, and a set of weight matrices $\{W_i\}$. We omit the bias terms for simplicity, considering it is natural to absorb them into the weight matrices. For the ith layer, matrix W_i will be factorized into the multiplication of two matrices A_i and B_i. We let $W_i \in \mathbb{R}^{n\times m}$, $A_i \in \mathbb{R}^{n\times p}$ and $B_i \in \mathbb{R}^{p\times m}$ and we define Θ_d that represents the index set of the weight matrices to be factorized, Θ_c that represents the index set of the weight matrices not to be factorized, and Θ that represents the index set of all weight matrices, i.e., $\Theta = \{i\}$.

B. Unified Perspective of Bayesian Inference

DNNs are generally considered as a series of deterministic functions with learnable parameters. To deeply understand the principle of network compression, let us first consider their optimization from the perspective of Bayesian inference and specifically maximum *a posteriori* $p(\{W_i\}|x, y)$

$$\begin{aligned}\{W_i^*\} &= \arg\max_{\{W_i\}} \mathbb{E}_{x,y}[\log(p(\{W_i\}|x, y))]\\ &= \arg\min_{\{W_i\}} \mathbb{E}_{x,y}[-\log(p(y|x, \{W_i\})p(\{W_i\}|x))] \qquad (1)\end{aligned}$$

given some known input x with desired output y. The obtained set of parameters $\{W_i^*\}$ and function $q(y|x; \{W_i^*\})$ that approximates the likelihood are adopted for inference. In general, the set $\{W_i\}$ is independent of the random vector x; hence, we also have $p(\{W_i\}|x) = p(\{W_i\})$. For classification tasks, y should be the label; hence, the expected negative likelihood term in (1) can be rewritten as

$$\mathbb{E}_{x,y}[-\log(p(y|x, \{W_i\})] = L_{\text{ce}}(p(y|x), q(y|x; \{W_i\}))$$

in which $p(y|x)$ is the underlying conditional probability distribution of y given a specific observation x, and $L_{\text{ce}}(\cdot, \cdot)$ calculates the cross-entropy for two distributions. Consequently, the optimization problem in (1) boils down to minimizing the regularized cross-entropy loss, and we have

$$\{W_i^*\} = \min_{\{W_i\}} L_{\text{ce}}(p(y|x), q(y|x; \{W_i\})) - \log p(\{W_i\}). \quad (2)$$

Let us focus on the second term $-\log p(\{W_i\})$, which is the negative logarithm of *a priori* distribution, and it acts as a regularization in our learning objective. Different distributions correspond to different types of regularizations. For instance, if $p(\{W_i\}) = \prod \exp(-w_{ijk}^2/2)$, in which w_{ijk} is the jkth entry of matrix W_i, and then, the regularization corresponds to the widely used l_2 weight decay.

Then, we consider imposing sparsity onto the DNNs. We follow the definition of "sparseness" in [30] and specifically extend it to probability distributions in this article. Mathematically, the sparsity of an n-dimensional vector z is defined by

$$S(z) = \frac{\sqrt{n} - \|z\|_1/\|z\|_2}{\sqrt{n} - 1}. \qquad (3)$$

TABLE I
SAMPLE TABLE TITLE

Name	p.d.f	$R(\cdot)$	Sparsity	ρ
Laplacian	$\propto \exp(-\frac{\lvert z_l\rvert}{\beta})$	$\lVert z_l\rVert_1$	$1-\sqrt{\frac{1}{2}}$	1
$l_{0.5}$	$\propto \exp(-\frac{\lvert z_l\rvert^{0.5}}{\beta})$	$\lVert z_l\rVert_{0.5}$	$1-\sqrt{\frac{1}{20}}$	$\frac{7\sqrt{30}}{10}$
Log-uniform	$\propto \frac{1}{\lvert z_l\rvert}$	$\ln\lvert z_l\rvert$	1	∞
Bernoulli	$p(z_l=\pm 1)=\alpha$	-	2α	1

Extending it to describing the sparsity of a probability distribution seems straightforward. In principle, if all the entries of a vector are independently sampled from a distribution and the vector dimension has $n \to +\infty$, then the introduced $S(z)$ should indicate the sparsity of z as well as the sparsity of the distribution. Specifically, we defined $S(p)$, in which $p(z_l)$ is the probability density function of the random variable z_l, as the sparsity of the random variable z_l (and the corresponding distribution) as follows:

$$S(p) = 1 - \mathbb{E}_{z_l\sim p(z_l)}(|z_l|)/\sqrt{\mathbb{E}_{z_l\sim p(z_l)}\left(z_l^2\right)}. \tag{4}$$

The above definition appropriately reflects the underlying relationship between probability distributions and their induced sparsity as sampling priors. To be more specific, if we choose a Bernoulli distribution with very small ($\to 0$) probability of letting the random variable z_l being equal to 1, then there should exist $\mathbb{E}_{z_l\sim p(z_l)}(|z_l|) \to 0$ and $\mathbb{E}_{z_l\sim p(z_l)}(z_l^2) = \mathbb{E}_{z_l\sim p(z_l)}(|z_l|)$. As a result, we should have $S(p) \to 1$. Some other famous distributions share a similar result, which means a distribution leading to more zeros processing a large $S(p)$, and vice versa.

We try using different distributions as *a priori* distributions. The corresponding formulations of function $R(\cdot)$ with important information are summarized in Table I, in which "p.d.f" is the abbreviation of "probability density function" and ρ is the finite third absolute moment of the distribution. We know from Table I that the sparsity of the Laplacian $(0, \beta)$ distribution is typically higher than that of the Gaussian distribution centered at zero, which coincides with our experience that the l_1-norm-based regularization tends to induce higher sparsity in practice. Here, we further introduce an "$l_{0.5}$ $(0, \beta)$ distribution" whose probability density function is provided in Table I, as a prior, which induces even higher sparsity than the Laplacian $(0, \beta)$ distribution. Bernoulli is a discrete distribution; hence, there is no corresponding continuous regularization function $R(\cdot)$. In practice, if we introduce such a prior, the entries of the weight matrix will be learned through the combination of $\tanh(\cdot)$ and a thresholding function $f_T(\cdot, \epsilon)$ as $f_T(\tanh(\cdot), \epsilon)$. $\epsilon > 0$ is a threshold value. Function $\tanh(\cdot)$ limits its output to be in the range of $(-1, 1)$, and function $f_T(\cdot, \epsilon)$ is typically represented as

$$f_T(x, \epsilon) = \begin{cases} 1, & x \in (\epsilon, 1) \\ 0, & x \in [-\epsilon, \epsilon] \\ -1, & x \in (-1, -\epsilon). \end{cases} \tag{5}$$

C. Sparse Matrix Factorization

It is plausible to combine the sparse and factorization-based representations at once to boost the compression performance. Recently, Yu *et al.* [31] propose to approximate the functionality of a weight matrix W_i using the summation of a low-rank matrix T_i and a sparse matrix S_i to minimize $\|W_i - T_i - S_i\|^2$ and maintain the relationships between feature maps in the meanwhile. Though effective in memory savings, their method requires huge overheads on the computation, since additional matrix multiplications and additions are introduced.

Unlike Yu *et al.* [31], we try to factorize the weight matrices of one or more layers into the product of two sparse and probably low-rank matrices in this article instead. To impose sparsity, the regularization function $R(\cdot)$ can be directly adopted in the optimization. Benefits of this framework come not only from a reasonable combination of low-rank and sparse representations but also from the theoretical guarantees of its performance. Theorem 1 along with some discussions in Section II-D will highlight it in more detail.

For the weight matrix in the ith layer of a DNN, we might factorize it or not, depending on the practical requirement. Given a set of training data $\{(x_{n'}, y_{n'})\}$, we use the formal mapping of a DNN instead of $q(y_{n'}|x_{n'}; \{W_i\})$ in (2). We have the optimization problem

$$\min_{\{W_i\}} \sum_{n'} L_{\mathrm{ce}}(p(y_{n'}|x_{n'}), F(y_{n'}|x_{n'}; \{W_i\})) + \sum_{i\in\Theta_d}(R(A_i) + R(B_i)) + \sum_{i\in\Theta_c}(R(W_i)) \tag{6}$$

with the constraint $W_i = A_i B_i, \forall i \in \Theta_d$. If $p = \min\{m, n\}$, then the compression may solely come from weight sparsity, but with $p < ((nm)/(n + m))$, we can also encourage low-rank factorization that further imposes efficiency constraints. We emphasize that many instantiations of $R(\cdot)$ do not enforce real sparsity but instead lead to very small weights. We can simply threshold the obtained weights to achieve strict zero in weights with $\varepsilon > 0$, just like the soft pruning techniques [16].

Equation (6) demonstrates the formal objective of our framework. In principle, any candidates in Table I can be chosen for $R(\cdot)$. In Section IV-B, we will experimentally compare the l_1-norm, $l_{0.5}$-norm, and log absolute functions in our framework. With (6), the $W_i = A_i B_i, i \in \Theta_d$ constraint and a set of training samples provided, we can simply derive an optimization problem that minimizes the empirical risk with regularization terms.

We summarize the whole learning algorithm in Algorithm 1. Function $g(\cdot)$ is an adaptive function for the regularization coefficient with two hyperparameters T_0 and T_1. In general, in the very beginning of the training process, a very large scaling factor λ should enforce most of the learnable parameters that become zero, which will cause the notorious vanishing gradient problem and make the training unstable. To resolve the issue, we set a relatively small value for λ for the first several training epochs and gradually increase it during training. After a period of time, the training becomes stable and we should roughly fix the scaling factor to a certain

Algorithm 1 Learning the Sparse Matrix Factorization

```
Set $\lambda_0$, initialize the scaling factor for the regularization term; initialize $A_i$ and $B_i$ for each $i \in \Theta_d$; initialize $W_i$ for each $i \in \Theta_c$; set the number of desired training epochs $T$; collect the set of training sample $\{(x_{n'}, y_{n'})\}$.
for $t = 1, \dots, T$ do
  $\lambda \leftarrow \lambda_0 * g((t - T_0)/T_1)$.
  repeat
    Collect a mini-batch of samples from $\{(x_{n'}, y_{n'})\}$.
    repeat
      Collect $i \in \Theta$.
      If $i \in \Theta_c$, update $W_i$ using back-propagation;
      else, update $A_i$ and $B_i$ using back-propagation.
    until All learnable parameters are updated.
  until All mini-batches are utilized to train.
end for
output $\{W_i\}, \{A_i\}, \{B_i\}$.
```

value. We accomplish this by adopting a sigmoid function for $g(\cdot)$.

D. Theoretical Analysis

From the perspective of Bayesian inference, we analyze the relationship between some probability distributions and their induced sparsity as sampling priors. We advocate factorizing the weight matrix W_i into a product of two matrices (i.e., A_i and B_i) with some priors. Intuitively, the neighboring entries of $A_i B_i$ can be statistically dependent and correlated, raising concerns in our analysis. In addition, the distribution of the entries of $A_i B_i$ (which will be referred to as the equivalent prior and its cumulative distribution function is denoted by $F_{A_i B_i}(\cdot)$) may be too heavily tailed and special to be approximated.

Fortunately, with some nontrivial derivations, we can prove that $F_{A_i B_i}$ well approximates a Gaussian distribution $\mathcal{N}(0, p)$ to a degree of $O((\rho^2/\sqrt{p}))$, in which ρ is the finite third absolute moment of the prior and p is the "common dimension" of $A_i \in \mathbb{R}^{n\times p}$ and $B_i \in \mathbb{R}^{p\times m}$. In fact, when $p \to \infty$, it is roughly equivalent to the central limit theorem. In addition, we also prove that elements in different locations are uncorrelated. See the below theorem for some formal theoretical conclusions.

Theorem 1: Given a bounded zero-mean unit-variance prior whose finite third absolute moment is denoted as $\rho = \mathbb{E}(|z_l|^3)$. If the entries of A_i and B_i are independent and share the same prior, then we have the following.

1) $\sup_{z_l} |F_{A_i B_i}(z_l) - F_{\mathcal{N}}(z_l)| \le ((C\rho^2)/\sqrt{p})$ holds, where $F_{\mathcal{N}}(\cdot)$ is the cumulative distribution function of a Gaussian distribution $\mathcal{N}(0, p)$ and C is a constant and $C > 0$.
2) Arbitrary two different entries of $A_i B_i$ are uncorrelated.

Proof: Let us here omit the subscript i of W_i, A_i, and B_i for better clarity. The entries of W, A, and B in their jth row and kth column are denoted as w_{jk}, a_{jk}, and b_{jk}. As $W = AB$, we have $w_{jk} = \sum_{l=1}^{p} a_{jl} b_{lk}$. The expectation hence satisfies $\mathbb{E}(w_{jk}) = \sum_{l=1}^{p} \mathbb{E}(a_{jl})\mathbb{E}(b_{lk})$. Since $\mathbb{E}(a_{il}) = \mathbb{E}(b_{il}) = 0$, we know $\mathbb{E}(w_{jk}) = 0$ holds. For any $1 \le j \le n, 1 \le k \le m, 1 \le u \le n, 1 \le v \le m$, and $j \ne u$ or $k \ne v$, we have

$$\begin{aligned}\mathbb{E}(w_{ij} w_{uv}) &= \mathbb{E}\left(\sum_{l=1}^{p} a_{jl} b_{lk} \sum_{l'=1}^{p} a_{ul'} b_{l'v}\right)\\ &= \mathbb{E}\left(\sum_{l=1}^{p}\sum_{l'=1}^{p} a_{jl} b_{lk} a_{ul'} b_{l'v}\right)\\ &= \sum_{l=1}^{p}\sum_{l'=1}^{p} \mathbb{E}(a_{jl} b_{lk} a_{ul'} b_{l'v}). \end{aligned} \tag{7}$$

When $j \ne u$ and $k \ne v$, we can get $a_{jl} \ne a_{ul'}$ and $b_{lj} \ne b_{l'v}$ for $\forall l, l'$. Therefore, $a_{jl}, a_{ul'}, b_{lk}, b_{l'v}$ are independent of each other $\mathbb{E}(w_{jk} w_{uv}) = \sum_{l=1}^{p}\sum_{l'=1}^{p} \mathbb{E}(a_{jl} b_{l'k} a_{ul} b_{l'v}) = 0$.

When $j = u$ and $k \ne v$, if $l' \ne l$, $a_{jl}, a_{ul'}, b_{lk}$, and $b_{l'v}$ are still independent of each other and we have $\mathbb{E}(a_{jl} b_{l'k} a_{ul} b_{l'v}) =$ 0, else if $l' = l$, then we have $a_{jl} = a_{ul'}$, $b_{lk} \ne b_{l'v}$, and

$$\begin{aligned}\mathbb{E}(a_{jl} b_{l'k} a_{ul} b_{l'v}) &= \mathbb{E}\big(a_{jl}^2 b_{l'k} b_{l'v}\big)\\ &= \mathbb{E}\big(a_{jl}^2\big)\mathbb{E}(b_{l'k})(b_{l'v}) = 0. \end{aligned} \tag{8}$$

We have the same result for $j \ne u$ and $k = v$. Consequently, we further have

$$\mathbb{E}(w_{jk} w_{uv}) = 0 = \mathbb{E}(w_{jk})\mathbb{E}(w_{uv}) \tag{9}$$

and the uncorrelation among the entries of $A_i B_i$ thus follows. According to the Berry–Esseen theorem [32], [33], if a set of random variables $\{x_l\}_{l=1}^{p}$ share a zero-mean and σ_x-variance distribution $\kappa(x_l)$, let $x = \sum_{l=1}^{p} x_l/\sqrt{p}$ and $\Phi(\cdot)$ be the cumulative distribution function of normal distribution, then we have

$$\sup_x |F(x) - \Phi(x)| \le \frac{C\rho_x}{\sigma_x^3\sqrt{p}} \tag{10}$$

for the cumulative distribution function $F(\cdot)$, in which ρ_x is the finite third absolute moment of $\kappa(x)$ and C is a constant. The value of C has been studied and upper-bounded (e.g., by 7.59 in [33] and 0.56 in [34]). As ρ is the finite third absolute moment of the sampling prior, i.e., $\mathbb{E}(|a_{jl}|^3) = \mathbb{E}(|b_{lk}|^3) = \rho$. Choosing $x_l = a_{jl} b_{lk}$, we should have $\mathbb{E}(x_l) = 0$, $\mathrm{Var}(x_l) = \mathbb{E}(a_{jl}^2 b_{lk}^2) = \mathbb{E}(a_{jl}^2)\mathbb{E}(b_{lk}^2) = 1$, and $\rho_x = \mathbb{E}(|a_{jl} b_{lk}|^3) = \mathbb{E}(|a_{jl}|^3)\mathbb{E}(|b_{lk}|^3) = \rho^2$. [1] According to (10), we further have

$$\begin{aligned}&\sup_x \left|P\left(\sum_{l}^{p} x_l/\sqrt{p} < x\right) - P(x_\Phi < x)\right| \le \frac{C\rho^2}{\sqrt{p}}\\ &\sup_x \left|P\left(\sum_{l}^{p} x_l < x\sqrt{p}\right) - P(x_\Phi\sqrt{p} < x\sqrt{p})\right| \le \frac{C\rho^2}{\sqrt{p}} \end{aligned} \tag{11}$$

in which x_Φ is a random variable with the normal distribution and the probability distribution function of $x_\Phi\sqrt{p}$ is $\mathcal{N}(0, p)$.

[1] Note that the symbol x here is somewhat abused and it has nothing to do with the input signal in the prequel.

Obviously, $P(\sum_l^p x_l < x\sqrt{p}) = F_{A_i B_i}(z_l = x\sqrt{p})$ and it follows that

$$\sup_{z_l} |F_{A_i B_i}(z_l) - F_{\mathcal{N}}(z_l)| \leq \frac{C\rho^2}{\sqrt{p}}. \tag{12}$$

□

Theorem 1 implies that a weight matrix W_i generated as the product of two whose sparsities are induced by previously discussed prior distributions can well approximate a Gaussian prior. Since the regularization effect of such a Gaussian prior is equivalent to the l_2-norm regularization (i.e., weight decay), our sparse matrix factorization theoretically approximates the effect of weight decay. As such, our framework compresses DNNs in a reasonable way and it is theoretically guaranteed.

Moreover, it is obvious that the degree of our approximation is related to the "common dimension" of A_i and B_i. The larger it is, the more similar $F_{A_i B_i}$ should be to a Gaussian distribution. The finite third absolute moment of the distribution also plays an essential role in Theorem 1. The lower it is, the better the approximation should be. Based on this result, we can select appropriate distributions from those candidates in Table I. The log-uniform distribution is nonstrictly bounded, so it does not really fit the condition of our Theorem 1. As for the Laplacian and $l_{0.5}$ distributions, their finite third absolute moments are both relatively low. See ρ in Table I for comparisons. Experimental results also prove that the performance of l_1 and $l_{0.5}$ regularizations is better.

III. Complexity Analysis of a Layer

In this section, we analyze the memory and computational complexity of the models trained with different configurations. We will demonstrate how much performance gain our framework can achieve in theory using sparse matrix factorization.

Let us here focus on a single fully connected layer for simplicity of notations. With some n-dimensional input and m-dimensional output features, the weight matrix is still denoted by $W_i \in \mathbb{R}^{n\times m}$. As suggested in previous works, we define the compression rate of any possibly sparse matrix W in the floating-point format as $t_W = (|W|/|W_{\neq 0}|)$. If W_i is factorized into the product of two matrices $A_i \in \mathbb{R}^{n\times p}$ and $B_i \in \mathbb{R}^{p\times m}$ with compression rates t_{A_i} and t_{B_i}, respectively, then the total compression rate should be the ratio of the original number of entries and the number of the nonzero entries in A_i and B_i

$$\check{t}_{W_i} = \frac{mn}{np/t_{A_i} + mp/t_{B_i}}. \tag{13}$$

For the most extreme situation when $t_{A_i} = 1$ and $t_{B_i} = 1$, $\check{t}_{W_i} > 1$ holds as long as $p < (nm/(n+m))$. Therefore, as previously discussed, we could select any $p \lesssim (nm/(n+m))$ such that the complexity is further reduced.

We now turn to the computational complexity. When an n-dimensional feature is fed into the layer, the required numbers of additions and multiplications are

$$\check{s}^{(\mathrm{mul})} = \check{s}^{(\mathrm{add})} = mn. \tag{14}$$

If the sparse matrix factorization is applied, then the required numbers of multiplications and additions are

$$\tilde{s}^{(\mathrm{mul})} = \tilde{s}^{(\mathrm{add})} = np/t_A + mp/t_B. \tag{15}$$

We also consider the possibility of adopting low precision or even ternary constraints on A_i or B_i. For instance, if the entries of B_i are constrained to be ternary and in the set of $\{-1, 0, +1\}$, more significant compression can be achieved. The number of required bits for model storage will be less, and the compression rate will be

$$\tilde{t}_{W_i} = \frac{mn}{np/t_{A_i} + mp/(t_{B_i}K)} \tag{16}$$

in which K is the ratio of storage for the floating-point weights and the reduced-precision weights and it is 16 in this case. Results can be similarly obtained if both the matrices are ternary.

For the multiplication of one ternary and one (possibly still floating-point) matrix, all floating-point multiplications can be converted into additions; thus, we further have

$$\begin{aligned}\tilde{s}^{(\mathrm{mul})} &= np/t_A \\ \tilde{s}^{(\mathrm{add})} &= np/t_A + mp/t_B.\end{aligned} \tag{17}$$

Results can be similarly obtained if both the matrices are constrained to be ternary.

The memory and computational complexities are comprehensively summarized in Table II. It thereby becomes easy to compare our framework with previous methods in this regard. In the context of memory complexity, our framework achieves higher compression rates than pruning strategies if $p < ((nm)/(m+n))$ and $t_{A_i} = t_{B_i} = t_{W_i}$, in virtue of $((mn)/(np/t_{A_i} + mp/t_{B_i})) < t_{W_i}$. With $t_{A_i}, t_{B_i} < 1$, and $((mn)/(np/t_{A_i} + mp/t_{B_i})) < (mn)/(mp+np)$, our framework can also achieve higher compression rates than the low-rank-based methods.

IV. Experiments

In this section, we validate the effectiveness of our framework and our theoretical conclusions in practice. We conduct experiments on a bunch of popular data sets and a variety of DNNs with different architectures, including the MLPs, CNNs, LSTMs, vanilla recurrent neural network (RNN), and gated recurrent unit (GRU) network. Our sparse matrix factorization is adopted to different layers and we compare the results with some previous state-of-the-art network compression methods, especially the ones generate sparse DNNs.

A. Experimental Setup

We shall first introduce some important experimental settings involving the training and test data sets, our DNN architectures, and the evaluation metrics as follows.

1) Training and Test Data Sets: We evaluate the effectiveness of our method on five different data sets: MNIST [35], CIFAR-10 [36], CIFAR-100 [36], ImageNet [37], and Penn treebank (PTB) [38]. The main information of these data sets is reported in Table III. The MNIST data set consisting of 70 000 labeled gray images will be used as the toy and the starting data set in our experiments. We use the officially allocated 60 000 images for training and the remaining 10 000 for performance evaluation. The CIFAR-10, CIFAR-100, and ImageNet data sets are used for our

TABLE II
MEMORY AND COMPUTATION COMPLEXITY OF DIFFERENT COMPRESSION CONFIGURATIONS

Method	$\lvert W\rvert/\lvert W_{\neq 0}\rvert$	#Multiplications	#Additions
Sparse Factorization (ours)	$\frac{mn}{np/t_A+mp/t_B}$	$\frac{np}{t_A}+\frac{mp}{t_B}$	$\frac{np}{t_A}+\frac{mp}{t_B}$
Sparse Factorization (ours) + Ternary	$\frac{mn}{np/t_A+mp/(t_B K)}$	$\frac{np}{t_A}$	$\frac{np}{t_A}+\frac{mp}{t_B}$
Sparse	t_W	$\frac{mn}{t_W}$	$\frac{mn}{t_W}$
Low-rank	$\frac{mn}{mp+np}$	$mp+np$	$mp+np$
Low-rank + Sparse	$\frac{mn}{mp+np+nm/t_W}$	$mp+np+\frac{mn}{t_W}$	$mp+np+\frac{mn}{t_W}$

TABLE III
INFORMATION OF THE EXPERIMENT DATA SETS

Dataset	#Features	#Categories	#Samples	Continuous?
MNIST	28×28	10	70,000	True
CIFAR-10	32×32×3	10	60,000	True
CIFAR-100	32×32×3	100	60,000	True
ImageNet	224×224×3	1,000	1,331,167	True
PTB	10,000	- *	1,000,000	False

* The task in our experiment is treated as regression.

CNN-based image classification experiments. We also adopt the official split of the training and test sets. The PTB data set is established for natural language processing, and we use it to train and test recurrent networks.

2) Network Architectures: For the MLP networks that are simply stacks of several fully connected layers with interlaced nonlinear activations, we choose a four-layer architecture for experiments. Its two hidden layers are composed of 300 and 100 neurons, respectively. This MLP architecture is widely used and normally referred to as "LeNet-300-100" in network compression articles (see [9], [12]). Considering the number of classes, the feature dimension is finally reduced to ten for output. For the CNNs, multiple widely adopted and famous architectures, namely LeNet-5 [9], VGG-like[2] [17], and some residual networks (including ResNet50 and ResNet164) [3], are chosen as representatives for further experiments. MNIST is used to train and evaluate the LeNet-300-100 and LeNet-5 models, CIFAR-10 is used to train and evaluate the VGG-like models, both CIFAR-100 and ImageNet are used to train and evaluate the ResNets, and they are all implemented following the official settings[3].

We also test our method on a series of recurrent networks including the vanilla RNN, GRU, and LSTM. We will verify the effectiveness of our method on MNIST and PTB. In order to reasonably adapt the MNIST image to the input of recurrent models, we reorganize the 28 × 28 images as features distributed at 28 time steps and each consists of 28 pixels in the same column. Vectors with 28 pixels will be projected into a 32-dimensional input vector. The vanilla RNN and GRU models are trained. The dimensions of their hidden states are uniformly set as 256. We feed their final hidden states to a softmax layer to get the final classification results. For the experiment on PTB corpus, we train the LSTM networks as language models. The data set involves 10 000 vocabularies, which are projected onto a 650-dimensional embedding subspace through an embedding layer, and the number of hidden states is set to be 650. Another larger LSTM owns a 1500-dimensional embedding subspace and 1500 hidden states. The network contains two LSTM layers.

3) Common Settings: If not further explained, the following settings are commonly adopted in all our experiments except for the VGG-like networks and residual networks. As the network optimizer, Adam [39] is applied with a base learning rate of 0.001. The hyperparameters T_0 and T_1 in Algorithm 1 are set to be 30 and 5, respectively. The number of training epochs is different in different experiments. For $f_T(\cdot, \epsilon)$, ϵ is set as 0.5, and ε is set as $\exp(-4)$. The dimension p of the factorized matrices is not uniformly given but should fulfill the same condition that $p \leq 1.2mn/(m+n)$. The sparse regularization function is selected from the l_1, $l_{0.5}$, and log-uniform functions. Their $R(\cdot)$ formulations are: $R_{l_1}(W_i) = \sum(|w_{ijk}|)$, $R_{l_{0.5}}(W_i) = \sum((|w_{ijk}| + \eta)^{0.5})$, and $R_{\ln}(W_i) = \sum \log(|w_{ijk}| + \eta)$, in which η is a very small value 10^{-7} used to avoid the input of $\log(\cdot)$ and the derivative of $|\cdot|^{0.5}$ being out of its feasible domain. We emphasize that there exist many other ways of imposing DNN sparsity that can be more effective within our framework (e.g., the l_0 penalty [40]). The experiments are mainly conducted for a proof of the concept for our framework; hence, we here focus on the primary ones. The scaling factors λ_0 for different layers are set to be different in the experiments. Our general principle is that the wider a layer is, the heavier our penalty should be.

4) Evaluation Metrics: The main evaluation metric adopted in our experiments is the compression rate. Suppose the original model is trained to be composed of dense weight matrices (and this is generally the case in practice), the compression rate should be the ratio of the size of the original matrices and the number of parameters of matrices after pruning. This can be slightly more complicated than the single-layer case discussed in our previous analyses. It is formulated as

$$\frac{\sum_{i\in\Theta} |W_i|}{\sum_{i\in\Theta_d} |A_{i,\neq 0}| + \sum_{i\in\Theta_d} |B_{i,\neq 0}| + \sum_{i\in\Theta_c} |W_{i,\neq 0}|} \tag{18}$$

in which $|W_i|$ is the total number of the entries of matrix W_i, and $|A_{i,\neq 0}|$ and $|B_{i,\neq 0}|$ are the number of nonzero entries of the possibly sparse matrices A_i and B_i, respectively. In the experimental sections, we shall use $(|W|/|W_{\neq 0}|)$ [16] as a simplified representation of (18). In addition, we define the number of parameters after pruning as "Params #" in Tables VIII–XI.

[2]https://github.com/geifmany/cifar-vgg.git
[3]https://github.com/tensorflow/models.git

B. Verification of the Theorem

To start up, we verify the theoretical conclusions in Section II-D. We shall demonstrate how the equivalent prior approximates the Gaussian distribution in practice. We take the first layer of LeNet-300-100 as an example for illustrations. Two different regularization functions l_1 and $l_{0.5}$ are adopted. The distributions of the weight values of the sparse matrices A_i and B_i and their product $A_i B_i$ are shown in Fig. 2. We try to verify two statements: 1) influenced by sparse regularization functions, the statistical distributions of the matrix entries will be empirically close to *a priori* distribution correspondingly and 2) the equivalent distribution of the product of two matrices well approximates the Gaussian distribution.

To facilitate a uniform description, we normalize the weight matrix before estimating the statistical distributions by illustrating the histograms. We summarize the results in Fig. 2. We can see from Fig. 2(a)–(d) that the histograms of the weights are indeed very close to the corresponding distribution with sparse regularizations, which is consistent with the conclusions of the first statement. There might exist some slight difference between the standard *a priori* distribution and the actual one, on account of the likelihood distribution determined by the classification task. We further see apparently from Fig. 2(e) and (f) that the equivalent distribution of the product is very close to the Gaussian distribution, which is also highly consistent with our theory. Since the Gaussian prior corresponds to the l_2 regularization, we know that our framework somewhat plays the role of weight decay and increases the sparsity of the model simultaneously.

C. Experiments With the MLP

We implement the sparse matrix factorization with three different regularization functions l_1, $l_{0.5}$, and $\ln|\cdot|$, corresponding to the Laplacian, $l_{0.5}$, and log-uniform *a priori* distributions. We try different configurations on Θ_d to demonstrate the layerwise performance of our method. There are three parameterized layers in LeNet-300-100. For better clarity, we "label" the layer(s) to be factorized as "1" and the others as "0." For instance, "1-0-0" represents that the first parameterized layer is factorized and the second and the third are not. Recall that, for the layers not to be factorized, we can still impose sparsity on it. Here, we follow this idea and compress all the parameterized layers for LeNet-300-100.

We compare our strategy with the previous state-of-the-art network compression methods, including the progressive pruning method [9], dynamic network surgery (DNS) [12], soft weight sharing (SWS) [16], and sparse variational dropout (SVD) [17]. We adopt the l_1-norm-based regularization to factorize the first parameterized layer, and the corresponding configuration is marked as "1-0-0-l_1" here. As demonstrated in Table IV, our method obtains higher compression with similar or even lower prediction errors than DNS, SWS, and SVD, achieving a new state of the art for this MLP. The results of competitors are cited from this articles. The layerwise scaling factors for regularizing the original l_1 and our factorized l_1 are simply set to be [0.00018, 0.00006, 0.00003] and [0.00012, 0.00004, 0.00002], respectively. Note that despite

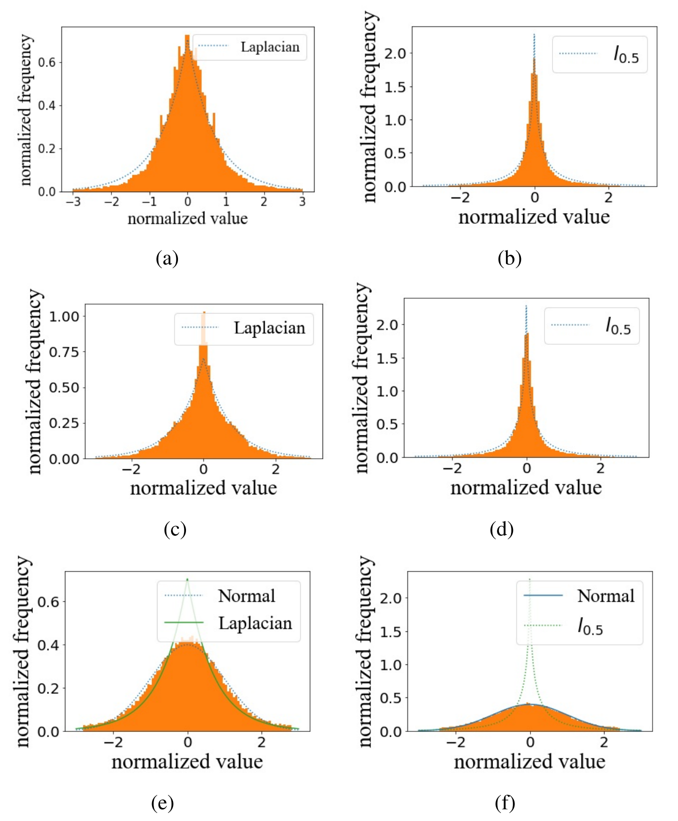


Fig. 2. Histogram of matrix entries if the first layer of LeNet-300-100 is factorized. (a) and (b) Histogram of A_i. (c) and (d) Histogram of B_i. (e) and (f) Histogram of the product $A_i B_i$. The dotted and solid lines represent the corresponding p.d.f of *a priori* distributions, respectively. (a) l_1, A_i. (b) $l_{0.5}$, A_i. (c) l_1, B_i. (d) $l_{0.5}$, B_i. (e) l_1, $A_i B_i$. (f) $l_{0.5}$, $A_i B_i$.

TABLE IV

CRs OF LeNet-300-100. OUR METHOD IS COMPARED WITH THE PROGRESSIVE PRUNING METHOD [9], DNS [12], SWS [16], AND SVD [17] IN DIFFERENT ASPECTS

Network	Method	Error (%)	$1-1/t_W$	$\frac{\lvert W\rvert}{\lvert W_{\neq 0}\rvert}$
LeNet-300-100	-	1.64	$0-0-0$	1
	Progressive [9]	1.59	$92.0-91.0-74.0$	12
	SWS [16]	1.94	-	23
	DNS [12]	1.99	$98.2-98.2-94.5$	56
	SVD [17]	1.92	$98.9-97.2-62.0$	68
	Original l_1	1.92	$98.6-95.4-61.0$	52
(ours)	1-0-0-l_1	1.83	$99.0-95.8-68.6$	**71**
(ours)	1-0-0-l_1 (t)	1.90	$99.9-90.7-49.9$	**75**

there exist a series of other configurations within our framework, we did not carefully tune the empirical choice for simplicity. As will be shown, different configurations with various hyperparameters achieve different results from the adopted ones.

We then analyze the practical relationships between the prediction accuracy and the compression rate with a variety of configurations in our framework. We test its performance on the same parameterized layers with different regularization functions. As shown in Fig. 3, "$l_{0.5}$" and "l_1" appear to be the best consistently. We also evaluate our method on different settings of Θ_d and Θ_c. We test seven different configurations: "1-1-1," "1-1-0," "1-0-1," "0-1-1," "1-0-0," "0-1-0,"

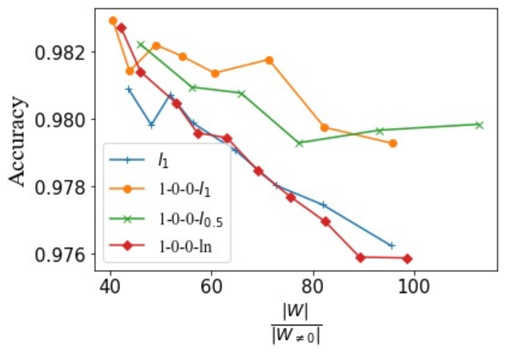


Fig. 3. Relationship between the CR and accuracy under different regularizations. The blue line is the original l_1. The "1-0-0-l_1," "1-0-0-$l_{0.5}$," and 1-0-0-ln" represent the relationships with l_1, $l_{0.5}$, and logarithmic regularizations, respectively, when the first layer of LeNet-300-100 is factorized and the others remain unchanged.

and "0-0-1" with expected relatively low, medium, and high compression rates by using the l_1-norm-based regularization. We set $p = 250$ and train for 300 epochs. Details of how we achieve different levels of network compressions will be carefully introduced in the next paragraph. The results are summarized in Table V. In general, the larger a weight matrix is, the better CR and lower error rate (ER) our method can gain. We observe that if the second and third parameterized layers are factorized, the performance trades off the worst and the accuracy degradations seem to be the most obvious. On the contrary, if the first two layers factorized, the performance is much better. Considering that larger weight matrices generally derive larger p, the observation is vaguely consistent with the conclusion of Theorem 1 and some practical experience. We also observe from Fig. 3 that l_1 and $l_{0.5}$ achieve significantly better performance than the $\ln|\cdot|$ regularization as the finite third absolute moment of the distribution corresponding to the latter can approach ∞.

We further experimentally analyze how the dimension p of the factorized matrix affects the performance of our method. We consider three practical cases with low, medium, and high percentage of zero entries. To achieve such different levels of sparsity and compressions for LeNet-300-100, we set different scaling factors accordingly: $[0.00001, 0.000025, 0.00015]$, $[0.0001, 0.00025, 0.0015]$, and $[0.0002, 0.0005, 0.003]$. All the experiments are conducted to only factorize the first parameterized layer, and the other two are kept as their original format but encouraged to be sparse. We aim to explore the possibility of further compression and also reduction in computation in such a manner. From Table VI, we easily observe that the prediction accuracy does not degrade much with an approximately lower p that suggests a low-rank factorization. According to our previous analysis on the memory and computational complexity, the same amount of computation should be performed for $p = ((mn)/(n+m)) = 217$. With $p < 217$, additional memory and computation reductions are obtained and our framework will benefit from the low-rank property. In addition, it is worth noting that when p is too small, e.g., $p = 50$, the prediction accuracy shall still diminish. We further evaluate the training cost. To run for an epoch, it takes 0.437 and 0.459 s with $p = 50$ and $p = 300$, respectively, while it takes 0.359 s with the reference.

TABLE V

RELATIONSHIP BETWEEN THE CR AND ER ON LeNet-300-100 UNDER VARIANT CONFIGURATIONS OF THE DECOMPOSITION. "L," "M," AND "H" REPRESENT LOW, MEDIUM, AND HIGH COMPRESSION RATES, RESPECTIVELY

Configuration	CR/ER (L)	CR/ER (M)	CR/ER (H)
1-1-1-l_1	17.32/1.71%	67.31/2.09%	102.63/2.36%
1-1-0-l_1	19.13/1.86%	71.63/2.22%	110.26/2.33%
1-0-1-l_1	22.36/1.85%	75.41/2.09%	116.54/2.57%
0-1-1-l_1	21.13/1.82%	57.39/2.24%	81.99/2.68%
0-1-0-l_1	23.40/1.93%	65.74/2.23%	93.24/2.30%
0-0-1-l_1	24.04/1.94%	64.34/2.34%	90.48/2.39%
1-0-0-l_1	24.85/1.71%	81.03/1.85%	125.26/2.46%
Original l_1	25.95/1.94%	66.26/2.23%	106.41/2.52%

TABLE VI

RELATIONSHIP BETWEEN THE CR AND ER ON LeNet-300-100 UNDER VARIANT COMMON DIMENSIONS OF THE DECOMPOSITION. "L," "M," AND "H" REPRESENT LOW, MEDIUM, AND HIGH COMPRESSION RATES, RESPECTIVELY

Common Dimension	CR/ER (L)	CR/ER (M)	CR/ER (H)
p=50	25.98/1.99%	73.56/2.26%	116.68/2.57%
p=100	23.38/1.84%	80.85/1.98%	118.82/2.51%
p=150	23.88/1.78%	78.51/2.10%	125.26/2.43%
p=200	24.85/1.71%	81.03/1.85%	125.26/2.46%
p=217	24.68/1.77%	79.13/2.12%	125.44/2.49%
p=250	23.84/1.76%	82.54/1.92%	125.44/2.53%
p=300	24.82/1.85%	81.93/2.11%	123.35/2.47%
Original l_1	25.95/1.94%	66.26/2.22%	106.41/2.62%

Similarly, we also explore the possibility of ternarizing one of the matrices obtained via factorization (i.e., its entries are in $\{\pm 1, 0\}$). We enforce all the matrices as sparse as in the previous experiments and report the CR in Table IV as well. Specifically, we use (5), and the equivalent parameter matrix is calculated as $W_i = A_i f_T(B_i, \epsilon)$. See the method marked with "(t)" in Table IV and it achieves the highest compression rate among all listed methods, although the test-set prediction accuracy is a little worse than its full-precision sparse counterpart. Our method compresses the number of parameters to 3.55k, while the best compelling method, i.e., SVD [17], achieves 3.91k.

D. Experiments With CNNs

We choose LeNet-5 [35], a VGG-like network, and two residual networks to evaluate the performance of our method on CNNs. For LeNet-5, most of its parameters lie in the first fully connected layer; therefore, we factorize it as described. The common dimension of the factorization is set as $p = 400$. Both the l_1- and $l_{0.5}$-norm-based regularizations are evaluated. The experiment is performed on MNIST. As shown in Table VII, our method can surpass most of the state of the arts except for the SVD that uses extremely sparse regularization and a variational method to approximate the regularization. Although it sometimes achieves a lower ER with a less aggressive compression, we stress that the complexity of such an approximation is very high. Within our framework, the $l_{0.5}$-norm-based regularization achieves relatively better tradeoff between the compression rate and the prediction accuracy, which is consistent with our theoretical discussions in Section II-D. Our method yields a model with

TABLE VII

CRs of LeNet-5. Our Method Is Compared With the Progressive Pruning Method [9], DNS [12], SWS [16], and SVD [17] in Different Aspects

Network	Method	Error (%)	$1-1/t_W$	$\frac{\lvert W\rvert}{\lvert W_{\neq 0}\rvert}$
LeNet-5	-	0.80	$0-0-0$	1
	Progressive [9]	0.77	$34-88-92.0-81$	12
	DNS [12]	0.91	$86-97-99.3-96$	111
	SWS [16]	0.97	-	200
	SVD [17]	0.75	$67-98-99.8-95$	280
	Original l_1	0.97	$43.1-96.7-99.8-95.8$	220
(ours)	l_1	0.88	$51.3-96.5-99.8-96.1$	220
	Original $l_{0.5}$	1.02	$41.3-97.7-99.9-91.2$	282
(ours)	$l_{0.5}$	0.87	$57.7-98.3-99.9-93.0$	**291**

TABLE VIII

CRs of the VGG-Like Network

Network	Method	Error (%)	$\frac{\lvert W\rvert}{\lvert W_{\neq 0}\rvert}$	Compression	Params #
VGG-like	-	6.90	1	-	14.98M
	Original l_1	6.52	14.1	Low	1.06M
(ours)	l_1	6.48	**16.0**	Low	0.94M
	Original l_1	8.26	28.9	High	0.52M
(ours)	l_1	7.33	**31.6**	High	0.47M

only 1.48k parameters, which is superior to all compelling methods (SVD: 1.54k).

Experiments are also performed on some more complex networks, i.e., the VGG-like network, ResNet164, and ResNet50. Specifically, we first pretrain the model without sparse penalty. Then, we directly incorporate the l_1 regularization and train factorized models to get ultimate compressions. As the parameters of these models lie more in convolutional filters, we should also factorize them just like in fully connected layers following a similar idea [23]. Given a convolutional layer that includes $n \times c$ filters with a common spatial size of 3×3, we factorize them into two low-rank groups of $n \times p$ and $p \times c$ filters with the spatial sizes of 3×1 and 1×3 (or 2×2 and 2×2), respectively.

For the VGG-like network, we attempt to factorize the last three convolutional layers (with $n = c = 512$) along with the first fully connected layer (with $m = n = 512$) by setting $p = 300$ and $\varepsilon = \exp(-6)$. The learning rate is set as 0.0005. We try $\lambda_0 = 0.0003$ and 0.0006 to achieve different levels of compressions. In Table VIII, we carefully compare our method with the l_1-norm-based regularization method without factorization. Apparently, our method can achieve higher compression rates under much lower test-set prediction errors, verifying the effectiveness of our method on convolutional layers and CNNs. In addition, our performance gain seems larger under the high compression circumstance, suggesting that our method seeks more reasonable tradeoffs between compression and accuracy. According to Theorem 1 and some analyses based on it, the equivalent regularization of the multiplication of two matrices constrained to be sparse is functionally close to the (l_2) weight decay, which provides a theoretical explanation for it.

Both the residual networks both consist of four large modules, and each module contains some residual blocks.

TABLE IX

CRs of the Residual Networks on CIFAR-100 and ImageNet

Network	Method	Top-1 (%)	$\frac{\lvert W\rvert}{\lvert W_{\neq 0}\rvert}$	Datasets	Params #
ResNet164	-	23.71	1	CIFAR-100	1.69M
	Original l_1	25.25	3.65	CIFAR-100	0.47M
(ours)	l_1	23.91	**3.88**	CIFAR-100	0.44M
ResNet50	-	23.51	1	ImageNet	25.51M
	Original l_1	25.29	6.19	ImageNet	4.12M
(ours)	l_1	24.65	**6.68**	ImageNet	3.82M

TABLE X

CRs of the Vanilla RNN and GRU on the MNIST Data Set. Our Method Is Compared With the Tensor Train (TT) [25] Method

Network	Method	Error (%)	$\frac{\lvert W\rvert}{\lvert W_{\neq 0}\rvert}$	Params #
Vanilla RNN	-	1.63	1	82.17K
	TT [25]	2.74	79.9	1.03K
(ours)	l_1	1.55	**84.5**	0.97K
GRU	-	0.82	1	222.0K
	TT [25]	1.59	68.9	3.2K
(ours)	$l_{0.5}$	1.17	149	1.5K
(ours)	l_1	1.17	**176**	1.3K

We factorize all the convolution layers in the last module that contains 3×3 filters. $p = \lceil 2n/3 \rceil$. For ResNet164, $n = c = 64$. We set $\lambda_0 = 0.000006$ and $\varepsilon = \exp(-3)$. For ResNet50, $n = c = 512$. We set $\lambda_0 = 0.00001$ and $\varepsilon = \exp(-6)$. In Table IX, compared with the l_1-norm-based regularization method, our method can achieve higher compression rates under lower test-set errors. These experimental results prove that our approach can work on more challenging data sets and more complex networks.

E. Experiments With the Vanilla RNN, GRU, and LSTMs

In this section, we evaluate our method with recurrent networks including the vanilla RNN, GRU network, and LSTMs. We refer interested readers to [41] and [42] for more details about these networks.

We first test on the MNIST digit images, for which the rows of each image are fed into the networks sequentially. The vanilla RNN is composed of three parameterized layers. We leverage the l_1-norm-based regularization and factorize the "input-to-hidden" layer and "hidden-to-output" layer. The GRU network is composed of seven parameterized layers, i.e., three "input-to-hidden" layers, three "hidden-to-hidden" layers for different gates or units, and one "hidden-to-output" layer.[4] We adopt the l_1–norm-based regularization to factorize all these layers except for the final output one. λ_0 is set as 0.0003 and ε is set as $\exp(-4)$. Table X compares our method with a previous state of the art. We see that our method outperforms it in compressing different RNNs.

We then use the PTB data set to train the LSTM-based language models. Words are fed into the LSTM models to

[4]The "input-to-hidden" layer indicates the layer that transforms the input vector to a hidden state vector or a gate vector. The "hidden-to-hidden" layer indicates the one that transforms the hidden state vector into another feature space, and the "hidden-to-output" layer indicates the one that transforms its input to an output vector.

TABLE XI
CRs OF LSTMs ON THE PTB DATA SET. OUR METHOD IS COMPARED WITH A LOW-RANK-BASED METHOD [43]

Network	Method	PPL	$\frac{\lvert W\rvert}{\lvert W_{\neq 0}\rvert}$	Params #
LSTM-650-650	-	82.07	1	19.76M
	Low-rank [43]	92.88	4.69	4.20M
(ours)	l_1	88.37	**6.24**	3.17M
LSTM-1500-1500	-	78.29	1	66.02M
	Low-rank [43]	89.46	2.78	21.72M
(ours)	$l_{0.5}$	87.46	**4.53**	14.57M

predict their sequential words in each sentence. The evaluation metric for such a task is the test perplexity (PPL) defined in [44]. We choose LSTM networks composed of ten parameterized layers, i.e., one embedding layer, four "input-to-hidden" layers, four "hidden-to-hidden" layers for different units, and one "hidden-to-output" layer. We evaluate our method on two LSTMs with different dimensions of representations. We denote by "LSTM-650-650" the network with 650-D input and hidden vectors and denote by "LSTM-1500-1500" the network with 1500-D representations. The l_1- and $l_{0.5}$-norm-based regularizations are adopted to the two networks to factorize all these layers except for the final output one, respectively. From Table XI, we see that our method is effective on the two LSTMs and outperforms a low-rank-based method [43].

V. CONCLUSION

In this article, we present an alternative and novel framework for compressing the DNNs. We advocate accomplishing the task by substituting the original parameter matrices with the multiplications of highly sparse ones. It is proven theoretically that the effect of our compression is approximately equivalent to a (l_2) weight decay. Experimental results demonstrate that our method with l_1- and $l_{0.5}$-norm-based regularizations can achieve the state of the art performance in comparison with other pruning and regularization-based methods. The proposed method is proven to be effective not only in the feedforward networks like MLPs and CNNs but also in recurrent networks like the vanilla RNN, GRU, and LSTMs. It can be generalized flexibly from the fully connected layers to convolutional layers in the CNNs. For future works, we attempt to develop methods for automatically adjusting hyperparameters so that the selection of the scaling factors can be incorporated into an end-to-end learning.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–14.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2016, pp. 770–778.

[4] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[5] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, May 2013, pp. 6645–6649.

[6] C. Yan *et al.*, "A fast Uyghur text detector for complex background images," *IEEE Trans. Multimedia*, vol. 20, no. 12, pp. 3389–3398, Dec. 2018.

[7] C. Yan, L. Li, C. Zhang, B. Liu, Y. Zhang, and Q. Dai, "Cross-modality bridging and knowledge transferring for image understanding," *IEEE Trans. Multimedia*, vol. 21, no. 10, pp. 2675–2685, Oct. 2019.

[8] C. Yan *et al.*, "STAT: Spatial-temporal attention mechanism for video captioning," *IEEE Trans. Multimedia*, to be published.

[9] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

[10] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[11] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 3, 2017, Art. no. 32.

[12] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.

[13] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini, "Group sparse regularization for deep neural networks," *Neurocomputing*, vol. 241, pp. 81–89, Jun. 2017.

[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.

[15] A. Honkela and H. Valpola, "Variational learning and bits-back coding: An information-theoretic view to Bayesian learning," *IEEE Trans. Neural Netw.*, vol. 15, no. 4, pp. 800–810, Jul. 2004.

[16] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *Proc. Int. Conf. Learn. Represent.*, 2017, pp. 1–16.

[17] D. Molchanov, A. Ashukha, and D. Vetrov, "Variational dropout sparsifies deep neural networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, 2017, pp. 2498–2507.

[18] E. Lobacheva, N. Chirkova, and D. Vetrov, "Bayesian sparsification of recurrent neural networks," 2017, *arXiv:1708.00077*. [Online]. Available: https://arxiv.org/abs/1708.00077

[19] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–14.

[20] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," 2018, *arXiv:1810.02340*. [Online]. Available: https://arxiv.org/abs/1810.02340

[21] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 1269–1277.

[22] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2015, pp. 1984–1992.

[23] C. Tai, T. Xiao, Y. Zhang, X. Wang, and E. Weinan, "Convolutional neural networks with low-rank regularization," in *Proc. Int. Conf. Learn. Represent.*, 2016, pp. 1–11.

[24] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. Brit. Mach. Vis. Conf.*, 2014, pp. 1–12.

[25] A. Tjandra, S. Sakti, and S. Nakamura, "Compressing recurrent neural network with tensor train," in *Proc. Int. Joint Conf. Neural Netw.*, May 2017, pp. 4451–4458.

[26] J. Cheng, P.-S. Wang, G. Li, Q.-H. Hu, and H.-Q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 1, pp. 64–77, 2018.

[27] J. Cheng, J. Wu, C. Leng, Y. Wang, and Q. Hu, "Quantized CNN: A unified approach to accelerate and compress convolutional networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4730–4743, Oct. 2018.

[28] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 3123–3131.

[29] Y. Guo, A. Yao, H. Zhao, and Y. Chen, "Network sketching: Exploiting binary structure in deep CNNs," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, vol. 2, 2017, pp. 4040–4048.

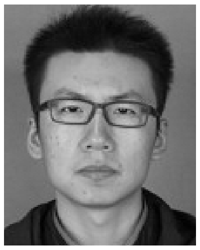
[30] P. O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *J. Mach. Learn. Res.*, vol. 5, no. 1, pp. 1457–1469, Nov. 2004.

[31] X. Yu, T. Liu, X. Wang, and D. Tao, "On compressing deep models by low rank and sparse decomposition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 67–76.

[32] A. C. Berry, "The accuracy of the Gaussian approximation to the sum of independent variates," *Trans. Amer. Math. Soc.*, vol. 49, no. 1, pp. 122–136, 1941.
[33] C.-G. Esseen, "On the Liapunoff limit of error in the theory of probability," *Ark. Mat. Astron. Fys.*, vol. A28, no. 9, pp. 1–19, 1942.
[34] I. Shevtsova, "An improvement of convergence rate estimates in the Lyapunov theorem," *Doklady Math.*, vol. 82, no. 3, pp. 862–864, 2010.
[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.
[36] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009, vol. 1, no. 4.
[37] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[38] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini, "Building a large annotated corpus of English: The Penn treebank," *Comput. Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.
[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. Int. Conf. Learn. Represent.*, 2015, pp. 1–15.
[40] C. Louizos, M. Welling, and D. P. Kingma, "Learning sparse neural networks through L_0 regularization," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–13.
[41] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014, *arXiv:1406.1078*. [Online]. Available: https://arxiv.org/abs/1406.1078
[42] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014, *arXiv:1412.3555*. [Online]. Available: https://arxiv.org/abs/1412.3555
[43] A. M. Grachev, D. I. Ignatov, and A. V. Savchenko, "Neural networks compression for language modeling," in *Proc. 7th Int. Conf. Pattern Recognit. Mach. Intell. (PReMI)*, Kolkata, India, 2017, pp. 351–357.
[44] D. Newman, E. V. Bonilla, and W. Buntine, "Improving topic coherence with regularized topic models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2011, pp. 496–504.

Kailun Wu received the B.E. degree from Tsinghua University, Beijing, China, in 2014. He is currently pursuing the Ph.D. degree with the State Key Laboratory of Intelligent Technology and Systems, Department of Automation, Tsinghua University.

His current research interests include deep learning and sparse coding.



Yiwen Guo received the B.E. degree from Wuhan University, Wuhan, China, in 2011, and the Ph.D. degree from Tsinghua University, Beijing, China, in 2016.

He is currently a Researcher with Cognitive Computing Laboratory, Intel Labs, Beijing. His current research interests include deep learning and adversial learning.

Changshui Zhang (M'02–SM'15–F'18) received the B.S. degree in mathematics from Peking University, Beijing, China, in 1986, and the M.S. and Ph.D. degrees in control science and engineering from Tsinghua University, Beijing, in 1989 and 1992, respectively.

In 1992, he joined the Department of Automation, Tsinghua University, where he is currently a Professor. He has authored more than 200 articles. His current research interests include pattern recognition and machine learning.