Neural Network Compression with Low Rank Structured Layers

Dimitris Papadimitriou

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-214
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-214.html

August 15, 2022

Neural Network Compression with Low Rank Structured Layers

by Dimitris Papadimitriou

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Somayeh Sojoudi
Research Advisor

(Date)

* * * * * * *

Professor Gerald Friedland
Second Reader

(Date)

Neural Network Compression with Low Rank Structured Layers

by

Dimitris Papadimitriou

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Somayeh Sojoudi, Chair
Professor Gerald Friedland

Summer 2022

Abstract

Neural Network Compression with Low Rank Structured Layers

by

Dimitris Papadimitriou

Master of Science in Computer Science

University of California, Berkeley

Professor Somayeh Sojoudi, Chair

Despite many modern applications of Deep Neural Networks (DNNs), the large number of parameters in the hidden layers makes them unattractive for deployment on devices with storage capacity constraints. In this project we first review a number of approaches developed to compress neural networks. We then propose a Data-Driven Low-rank (DDLR) method to reduce the number of parameters in pretrained DNNs and expedite inference by imposing low-rank structure on the fully connected layers, while controlling for the overall accuracy and without requiring any retraining. We pose the problem as finding the lowest rank approximation of each fully connected layer with given performance guarantees. We show that it is possible to significantly reduce the number of parameters in common DNN architectures with only a small reduction in classification accuracy. We compare DDLR with Net-Trim, which is another data-driven DNN compression technique based on sparsity and show that DDLR consistently produces neural networks with fewer parameters while maintaining higher accuracy.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my research advisor Professor Somayeh Sojoudi for her guidance and mentorship during the projects we collaborated. Without her support this Masters degree would not have been possible. I would also like to thank Professors Gerald Friedland, Kyriakos Komvopoulos and Francesco Borrelli, as well as Swayambhoo Jain, for the advice and assistance they provided. The work on this project concerning imposing low displacement rank and low rank structures on the dense layers of neural networks was done in collaboration with Swayambhoo Jain from InterDigital AI Lab in Los Gatos. The main contributions were published as *Data-Driven Low-Rank Neural Network Compression* at the IEEE International Conference on Image Processing (IEEE ICIP 2021).

Chapter 1

Introduction

Running Deep Neural Network (DNN) based methods locally on mobile devices is becoming a necessity for many modern applications. The importance of deploying AI on the edge, in devices such as smartphones, drones and autonomous vehicles, can be mainly attributed to three factors. Using cloud resources to run AI algorithms can lead to delays in inference due to *communication latency*. Furthermore, such communication with the cloud is *energy inefficient* as it requires additional power and is prone to *privacy breaches*, which could have dire consequences.

On the other hand, the number of parameters in neural networks, and more specifically the number of layers and their corresponding sizes, has been continuously increasing since the first widespread architectures from the early 2000s. Larger numbers of parameters inherently require more storage allocation and frequently lead to computationally intensive calculations. These two restrictions can be prohibitive for the current hardware available on edge devices. Hence, deep learning applications on the edge require some kind of model compression and possibly specialized hardware for efficient deployment.

In this project we focus on neural network compression techniques and we propose a method that can compress the fully connected layers of a pre-trained neural network by imposing a low rank structure on them while maintaining certain accuracy levels. Imposing a structure allows us to significantly reduce the number of parameters. Furthermore, layers with specific structure can allow for faster inference by taking advantage of efficient matrix-vector multiplication algorithms.

The rest of the project is organized as follows. In Chapter 2 we summarize already established methods devoted to neutral network compression. Such methods can be implemented during training, post-training and in an alternating “train-compress-re-train" scheme. More emphasis is put on methods developed for already trained neural networks as this is the case for our method as well. In Chapter 3 we introduce our method, called Data-Driven Neural Network (DDLR) compression. DDLR is a post-training method that imposes a low rank structure on the fully connected layers of pre-trained neural networks. We compare its compression performance with another state of the art compression method on a number of datasets.

Notation: We denote matrices and vectors with bold upper and lower case letters, respectively. For scalars, we use nonbold lower case letters. The identity matrix of size n is denoted with $\mathbf{I}_n$ and a n-dimensional vector of ones with $\mathbf{1}_n$. With $||\cdot||_1$, $||\cdot||_2$, $||\cdot||_F$ and $||\cdot||_*$ we denote the ℓ_1, ℓ_2, Frobenius and nuclear norms, respectively. When referring to matrix norms, the ℓ_1 norm is not the ℓ_1 matrix induced norm. Instead, it is defined as $\|\mathbf{X}\|_1 = \sum_{i,j} |\mathbf{X}^{i,j}|$. ReLU is the activation function defined as $\text{ReLU}(x) = max(x, 0)$. The Hadamard product, or element-wise product, is denoted with $\circ$.

Chapter 2

Background on Neural Network Compression Techniques

2.1 Introduction

Applications of neural networks, being the flagship of machine learning tools, on the edge have received a lot of attention lately. As small devices are becoming increasingly popular in everyday life, being able to deploy deep learning applications on them is crucial. Such devices can have limited memory, in the order of KBs or MBs, and weak processors. On the other hand, Figure 2.1 shows the size of some popular neural network architectures proposed in the literature [8, 42, 19, 20, 46, 45, 56, 48, 49]. The architectures that achieve high top-1 accuracy also contain millions of parameters to be stored which can be prohibitive for a large number of edge devices. Additionally, inference can be very slow in such cases. Naturally, compression techniques of such neural networks is an active research area. Compression techniques can be roughly categorized as *Network pruning*, *Knowledge distillation*, *Quantization* and imposing *Structured layers* such as *Sparse*, *Low Displacement Rank* and *Low Rank Layers* [7, 37]. The majority of the compression techniques developed are targeted at removing redundant parameters and hence lead to better generalization of the models. Lately, techniques that aim at compressing neural networks for edge computing applications have been proposed. In the following sections we briefly summarize some well established techniques for neural network compression. We go into detail on some methods that impose sparse and low displacement rank structures as these are related to our work. Compression by imposing low rank structure on the layers is mainly discussed in Chapter 3.

2.2 Network pruning

One of the first approaches in network pruning was Biased Weight Decay [17] in which the authors propose a weighting method that drives small in magnitude parameters to zero, allowing for larger parameters to survive. The optimal brain damage proposed in [29] com-

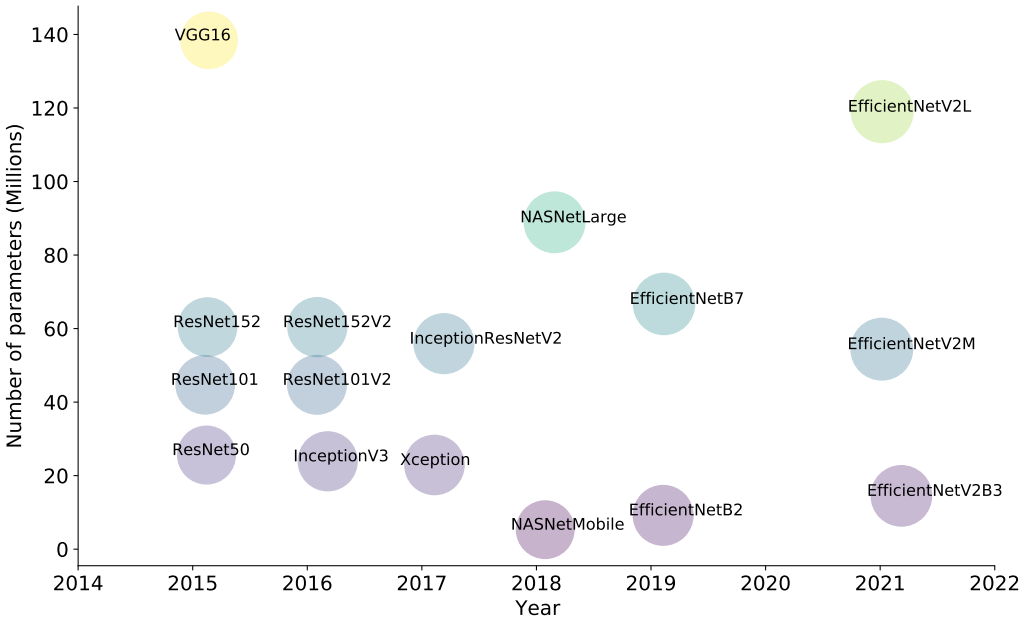


Figure 2.1: Size of popular neural network architectures. Size of the circles is proportional to the top-1 accuracy of the network.

putes the saliency of the network parameters using the second derivative of the objective function, removing the ones with the lowest score. Similarly, [18] relax the previously made assumption on the Hessian of the objective managing to remove more parameters for similar levels of accuracy. Regarding convolutional neural networks, [27] propose a brain damage approach for convolutional layers which are pruned in a group-wise fashion leading to faster inference.

2.3 Knowledge distillation

The concept of knowledge distillation revolves around training smaller (“student") neural networks to mimic the outputs of a larger (“teacher") neural network. Introduced by [6] and further popularized by [21] the goal of knowledge distillation is for the student network to achieve similar class probability outputs with the teacher network. In [3] the authors propose a reinforcement leaning approach to create a student network by selecting layers from the teacher and adjusting their sizes. A progressive blockwise learning scheme is proposed in [52] in which knowledge distillation is implemented as a progressive blockwise function approximation problem. In [9] the authors propose using the correlations between filter

responses within the layers of the teacher network to recommend a compressed network that maintains most of the original models accuracy. Quantized distillation is developed in [39] in which a distillation loss with respect to the teacher network is incorporated in the training process of the student network. Furthermore, the weights of the student network are quantized.

2.4 Quantization

The process of quantization refers to representing parameters, layer outputs and gradients in a reduced number of bits. Most frequently calculations in CPUs and GPUs, unless specified otherwise, use a 32 bit precision in a float32 format. A typical quantization scheme changes the format to a lower precision, like for instance the int8 format. Such a representation reduces storage requirements for the model while potentially leading to faster inference. In [15] the authors compress the dense layers of CNNs using k-means clustering to "assign" the weights to quantized values. Inner product quantization can also be leveraged to accelerate inference in the convolutional and dense layers of CNNs [53]. In [55] the authors propose an improved quantization method that utilizes outlier channel splitting in order to quantize CNNs without the need for re-training. Using an adaptive bit-width scheme has been proposed in the literature [24] to enable deployment of models on devices with different resource budgets. Low rank structures on the layers and quantization are combined in [23] in an end-to-end neural network compression framework.

2.5 Structured layers

This section concerns compression of neural networks by imposing a structure on the convolutional and dense layers. Such structure typically involves sparse, low displacement rank and low rank matrices. In the following subsections we will detail methods that impose such structure on the dense layers of networks as well as some of the benefits and disadvantages of them.

Sparse layers

The majority of neural network pruning techniques, implemented during or after training, lead to sparse layers. The authors in [44], [36], [34] encourage sparse layers while training neural networks by using appropriate regularization. The authors in [1] propose an optimization based approach to impose a sparse structure on an already trained neural network. Furthermore, in [13] the authors suggest that each trained neural network model has a sparsity structure that when it gets re-trained with only those parameters, under the same initialization as the original network, then it can achieve almost as high test accuracy as the original network. Before delving into the details we will first introduce some notation that will be used in subsequent formulations.

Consider a pre-trained CNN or DNN with L dense layers. Let the ℓ_{th} layer of this network be a fully connected dense layer with weight matrix $\mathbf{W}_\ell \in \mathbb{R}^{n_{\ell-1}\times n_\ell}$ with $n_{\ell-1}$ denoting the dimension of its input and n_ℓ being the dimension of its output. The corresponding bias of that layer is denoted with $\mathbf{b}_\ell \in \mathbb{R}^{n_\ell}$. Let also $\mathbf{Y}_{\ell-1} \in \mathbb{R}^{N\times n_{\ell-1}}$ and $\mathbf{Y}_\ell \in \mathbb{R}^{N\times n_\ell}$ denote the input and output data matrices of the ℓ_{th} layer respectively, with the number of rows N corresponding to the number of training data points in the network input matrix $\mathbf{X} \in \mathbb{R}^{N\times n_0}$. We focus on networks that utilize the $\text{ReLU}(x) = max(x, 0)$ activation function as they form the backbone of network architectures. Given the input $\mathbf{Y}_{\ell-1}$ the output of layer ℓ is obtained as follows

$$\mathbf{Y}_\ell = \text{ReLU}(\mathbf{Y}_{\ell-1}\mathbf{W}_\ell + \mathbf{1}_N\mathbf{b}_\ell^T), \tag{2.1}$$

where $\mathbf{1}_N$ is a N-dimensional vector of ones. In order to impose a sparse structure on the weight matrix $\mathbf{W}_\ell$ the authors in [1] minimize the ℓ_1 norm of that matrix as seen in the following optimization problem

$$\begin{aligned} \underset{\mathbf{U}\in\mathbf{R}^{n_{\ell-1}\times n_\ell}}{\text{minimize}} \quad & ||\mathbf{U}||_1 \\ \text{subject to} \quad & ||\text{ReLU}(\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T) - \mathbf{Y}_\ell||_F \leq \epsilon_\ell. \end{aligned} \tag{2.2}$$

This formulation allows for imposing structure on the layers while explicitly controlling for the error of the compressed layer output. It should be noted that the $\|\cdot\|_1$ does not designate the induced ℓ_1 matrix norm but it is the sum of the absolute values of the matrix entries. The layer output error due to compression is controlled by a user specified threshold ϵ_ℓ. Intuitively, we expect as ϵ_ℓ increases the sparsity of the layer to increase more since the constraint is becoming more relaxed. However, the constraint in (2.2) is non-convex due to the composition of the ReLU activation function and the norm function. To alleviate this issue the authors relax the constraint, to obtain the following convex constraint

$$\begin{cases} ||(\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T - \mathbf{Y}_\ell) \circ \mathbf{M}_\ell||_F^2 \leq \epsilon_\ell^2 \\ (\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T) \circ \mathbf{M}'_\ell \leq 0 \end{cases}, \tag{2.3}$$

where $\mathbf{M}_\ell$ is a mask matrix of the same dimension as $\mathbf{Y}_\ell$ selecting the positive entries of $\mathbf{Y}_\ell$ elementwise, i.e. the $(i,j)_{\text{th}}$ entry $\mathbf{M}_\ell^{ij} = 1$ if $\mathbf{Y}_\ell^{ij} > 0$ and $\mathbf{M}_\ell^{ij} = 0$ otherwise. Similarly, $\mathbf{M}'_\ell$ is a mask matrix selecting the non-positive entries of $\mathbf{Y}_\ell$. Intuitively, this constraint penalizes the deviation of the entries of the compressed layer that correspond to the positive entries of the original layer as the latter are the only ones that are not affected by the ReLU activation function. Furthermore, the entries that correspond to the non-positive entries of the original layer are allowed to take any non-positive value. Using the constraint relaxation in (2.3) we obtain the following convex optimization problem

$$\begin{aligned} \underset{\mathbf{U}\in\mathbf{R}^{n_{\ell-1}\times n_\ell}}{\text{minimize}} \quad & ||\mathbf{U}||_1 \\ \text{subject to} \quad & ||(\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T - \mathbf{Y}_\ell) \circ \mathbf{M}_\ell||_F^2 \leq \epsilon_\ell^2 \\ & (\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T) \circ \mathbf{M}'_\ell \leq 0. \end{aligned} \tag{2.4}$$

The optimization problem (2.4) imposes a sparse structure on a single layer of a network. To compress networks with multiple layers each layer can be compressed individually and independently from each other. This process is outlined in Algorithm 2, where compression of each layer is an independent from the rest of the layers optimization problem. Given that each layer is compressed independently, the algorithm allows for parallel implementation. The algorithm requires the initial data matrix as input $\mathbf{X}^{N\times n_0}$, the original trained weight matrices and biases of the layers and the user specified tolerances ϵ_ℓ. The output is a sequence of sparse matrices for each dense layer. Different values of ϵ_ℓ for each layer $\ell = 1, \ldots, L$ lead to different compression ratios. The higher the value the more freedom there is in compressing the layer and hence the more sparse the solutions will be.

Algorithm 1: Parallel Net-trim Algorithm [1]

Input : $\mathbf{X}, \mathbf{W}_\ell, \mathbf{b}_\ell,\ \epsilon_\ell,\ \ell = 1, \ldots, L$

$\mathbf{Y}_0 = \mathbf{X}$

for ℓ= *1,...,L* **do**

  $\mathbf{Y}_\ell = \mathrm{ReLU}(\mathbf{Y}_{\ell-1}\mathbf{W}_\ell + \mathbf{1}_N\mathbf{b}_\ell^T)$

end

for ℓ= *1,..., L* **do**

  $\mathbf{M}_\ell^{i,j} = 1$ if $\mathbf{Y}_\ell^{i,j} > 0$, otherwise 0

  $\mathbf{M}_\ell^{\prime i,j} = 1$ if $\mathbf{Y}_\ell^{i,j} \leq 0$, otherwise 0

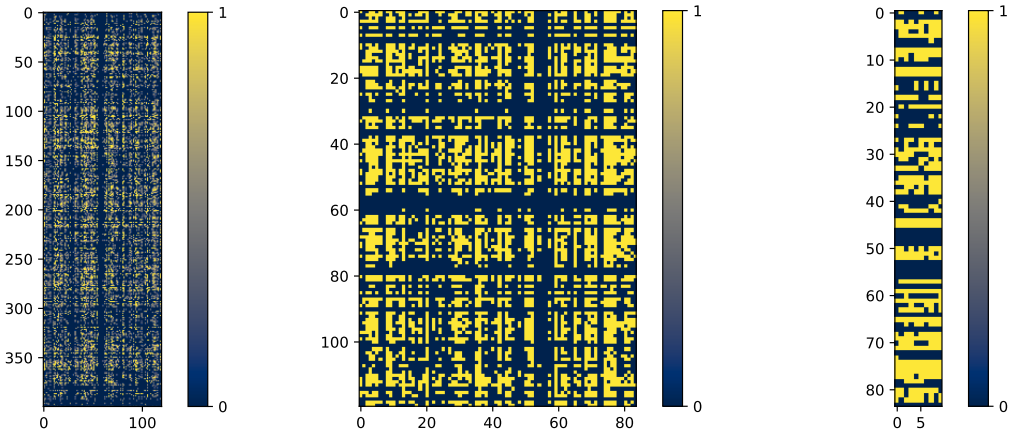
  Solve (2.4)

end

Output : $\hat{\mathbf{U}}_\ell,\ \ell = 1, \ldots, L$

Although sparse structures have the immediate effect of reducing the number of parameters needed to be stored, the benefits regarding the speed in inference are not certain. For instance in the sparse matrix-vector (SpMV) multiplication a typical approach is to perform the product in a row-wise fashion. In this case, if the number of nonzeros are unevenly distributed in the rows of the matrix then the benefit in inference speed is diminished. To showcase this we train a LeNet-5 CNN network [30] on the MNIST [28] dataset and we compress it using Algorithm 1 with $\epsilon = 0.1 \cdot \|\mathbf{Y}_{\ell-1}\|_F$ for each dense layer with $N = 200$ training data points. In Figure 2.2 we plot the sparsity patters acquired on the three dense layers of the network where the values below 0.01 were thresholded. Clearly, the sparsity patterns are such that inference speedup gains are not immediate.

Given that for the compression we randomly selected $N = 200$ data points from the dataset, in Figure 2.3 we repeat the process 10 times, each with different sample of N data points, and we plot the cumulative nonzero occurrence on the second dense layer of the LeNet-5 network. These results also show the nonuniform sparsity pattern of the layer. It should be noted that N is a lot smaller than the total number of training data points. The reason for this is that the optimization problem (2.2) does not scale well with N.



(a) (b) (c)

Figure 2.2: Sparsity patters of *first* (2.2a), *second* (2.2b) and *third* (2.2c) dense layers of LeNet-5 network trained on MNIST dataset and compressed using Algorithm 1.

In conclusion, although sparsifying neural networks can reduce the number of parameters significantly with limited loss in accuracy, the gains in inference acceleration might not be as impressive. To achieve a speedup in inference an approach that imposes structured sparsity, for instance by fixing the number of nonzeros per row, must be implemented.

Low displacement rank layers

Low Displacement Rank (LDR) approximations of fully connected hidden layers in neural networks have not received as much attention as the sparse counterparts. The authors in [54] provide theoretical results on the performance of neural networks that utilize hidden layers with LDR structure. In [43] the authors impose LDR structure on the hidden layers in the training process and show a significant speedup on training and inference compared to other already established methods.

Low displacement rank matrices need $O(n)$ independent parameters and allow for fast matrix operations. More specifically, a matrix is LDR if either the Stein operator ($\mathbf{L} = \Delta_{\mathbf{A},\mathbf{B}} : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n}$) defined by

$$\Delta_{\mathbf{A},\mathbf{B}}[\mathbf{M}] = \mathbf{M} - \mathbf{AMB}, \tag{2.5}$$

or the Sylvester operator ($\mathbf{L} = \nabla_{\mathbf{A},\mathbf{B}} : \mathbb{R}^{n\times n} \mapsto \mathbb{R}^{n\times n}$) defined by

$$\nabla_{\mathbf{A},\mathbf{B}}[\mathbf{M}] = \mathbf{AM} - \mathbf{MB}, \tag{2.6}$$

have low rank. The fixed matrices $\mathbf{A} \in \mathbb{R}^{n\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$ are called operator matrices and we denote the displacement rank of a matrix $\mathbf{M}$ with $\mathbb{L}_{\mathbf{A},\mathbf{B}}(M)$. When both operator

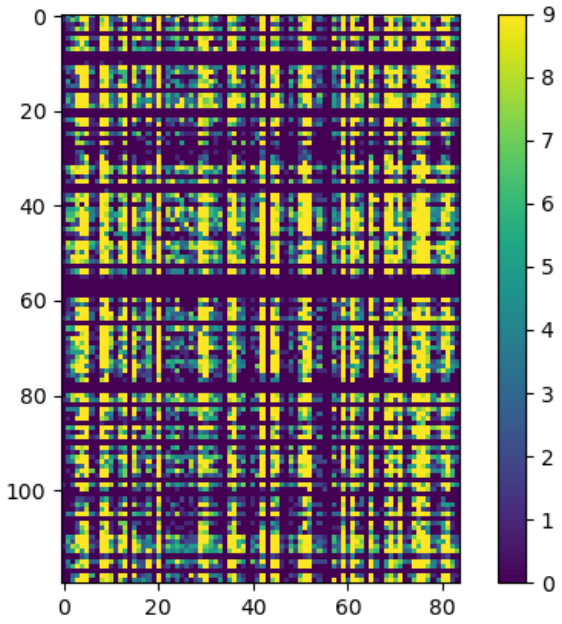


Figure 2.3: Distribution of unpruned entries in the weight matrix of the second layer of LeNet-5. Results obtained by solving problem (3.2) for 10 different subsets of the training data of size 200 each.

$\mathbf{M}$	$\mathbf{A}, \mathbf{B}$	$\mathbb{L}_{\mathbf{A},\mathbf{B}}(\mathbf{M})$
Toeplitz	$\mathbf{Z}_1, \mathbf{Z}_{-1}$	≤ 2
Hankel	$\mathbf{Z}_0, \mathbf{Z}_1$	≤ 2
Cauchy	$diag(c_1, \ldots, c_n), diag(d_1, \ldots, d_n)$	≤ 1
Vandermonde	$diag(1/x_1, \ldots, 1/x_n), \mathbf{Z}_1$	≤ 1

Table 2.1: Structured matrices, their operators and their displacement rank.

matrices are invertible the two operators are equivalent [11]. Table 2.1 shows the structure of a matrix and its displacement ranks for appropriate choice of operator matrices. In the table, $diag(c_1, \ldots, c_n)$ denotes a diagonal matrix with c_i being its diagonal elements and the matrix $\mathbf{Z}_f$ is defined as

$$\mathbf{Z}_f = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & f \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & 0 \end{bmatrix}. \tag{2.7}$$

Matrices with low $\mathbf{L}_{\mathbf{Z}_1,\mathbf{Z}_{-1}}$ are called close-to-Toeplitz or Toeplitz-like matrices. Hankel-like, Cauchy-like and Vandermonde-like matrices are defined analogously.

Obtaining a LDR approximation for the fully connected layers of a neural network has a number of benefits both for storage and inference. In the latter case, faster matrix-vector multiplication may allow inference to take place on mobile devices efficiently. A number of results regarding matrix-vector multiplication when the matrix has low displacement rank are provided in [14]. Regarding storage, such matrices would require storing the operator matrices and $2(\alpha \times n)$ elements instead of n^2, where α denotes the low displacement rank [50]. When displacement rank is low such a decrease in space complexity can be very significant.

The compression algorithm developed in [1] can be modified to impose LDR structures on the dense layers of networks. For instance the ℓth layer of a pre-trained neural network can be compressed by solving the following optimization problem.

$$\begin{aligned} \underset{\mathbf{U}\in\mathbf{R}^{n_{\ell-1}\times n_\ell}}{\text{minimize}} \quad & ||\mathbf{U} - \mathbf{AUB}||_* \\ \text{subject to} \quad & ||(\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T - \mathbf{Y}_\ell) \circ \mathbf{M}_\ell||_F^2 \leq \epsilon_\ell^2 \\ & (\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T) \circ \mathbf{M}'_\ell \leq 0. \end{aligned} \tag{2.8}$$

Problem (2.8) is a convex optimization problem as the objective (composition of linear and convex function) and the constraints are both convex. The nuclear norm is used as a relaxation of the non-convex rank function [40]. The specific structure of the resulting matrix $\mathbf{U}$ is controlled by appropriate choice of operator matrices $\mathbf{A}$ and $\mathbf{B}$. For non-square matrices, one can utilize the approach proposed for square matrices by appropriately padding the layer matrix $\mathbf{W}^\ell$ with zero rows or columns. Combining the square and non-square approach we can now approximate a neural network from input to output. For each hidden layer, problem (2.8) can be solved in parallel and produce the corresponding low displacement rank weight matrix.

To showcase the structure of the compressed matrices we use an artificial spiral dataset (Figure 2.4), presented in detail in the next chapter, to perform binary classification with a DNN. We use a DNN with three hidden layers. The weight matrices have dimensions $\mathbf{W}_1^{2\times 80}$, $\mathbf{W}_2^{80\times 80}$, $\mathbf{W}_3^{80\times 80}$, $\mathbf{W}_4^{80\times 2}$. We run algorithm 1, by solving problem (2.8) in line 8, to impose a Toeplitz-like structure on the two square layers of the networks. In Figure 2.5 we plot the magnitude of the entries of the original weight matrices along with the structured compressed ones for comparison. The Toeplitz-like pattern is clear in the compressed layers.

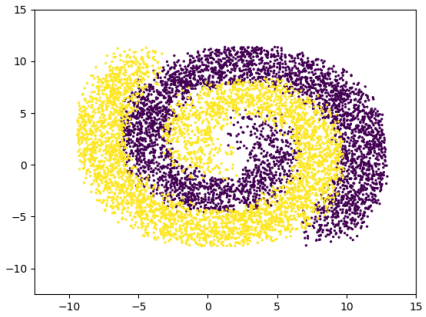
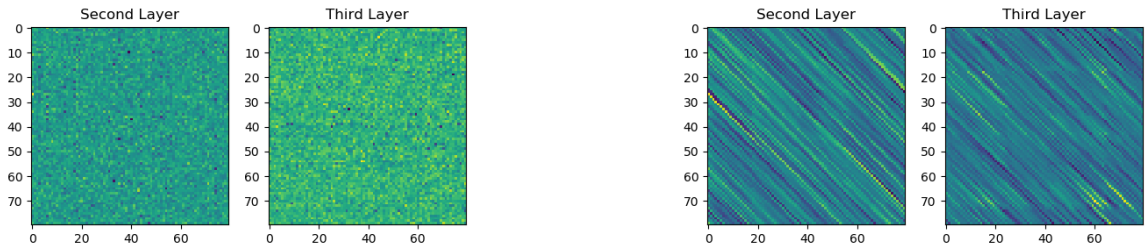


Figure 2.4: Spiral data set with two classes.



(a) Original layers.

(b) LDR layers.

Figure 2.5: Magnitude of the entries of weight matrices for the two square layers of the neural network.

Low rank layers

Using a similar formulation to the ones presented in the sparse and LDR layer sections we can also impose a low rank structure on the dense layers of neural networks. In comparison to LDR we found that low rank structures are simpler to implement and lead to similar compression accuracy and efficiency. As low rank structures form the backbone of this project, the method is presented in the next chapter in its entirety.

Chapter 3

Imposing Low Rank Layers

3.1 Introduction

Recent work suggests that compressing overparametrized DNNs after training leads to a reduction in the overall time and cost of development of DNN based applications [32]. In the context of the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [10] it took multiple years of extensive research and development to reduce the size of initial networks like VGG16 [42] and find computationally efficient alternatives such as MobileNets [22]. In comparison, the DNN compression approach can provide an equivalent reduction in a cost-effective way by automating the research for smaller and efficient DNNs.

In this project, we propose the DDLR approach to compress a given CNN or DNN by imposing a low rank structure on its fully connected layers. While there exist approaches to reduce the number of parameters in a given DNN by imposing structures such as sparsity and low displacement rank, these approaches are not data-driven and as a consequence they require computationally expensive retraining after parameter reduction [7].

Recently, the data-driven sparsity based approach Net-Trim showed that leveraging data during parameter reduction leads to better compression ratios without retraining [1]. While sparsification gives good compression performance for storage and transmission, it is very challenging to get equivalent gains in inference unless special hardware is designed to explicitly exploit the sparsity and custom software implementation is utilized. In contrast, the low rank based structural approximation that factorizes each parameter matrix as the product of two low dimensional matrices has no such requirements [51].

Our results show that our method manages to reduce the number of parameters significantly more than Net-Trim while maintaining accuracy levels comparable to the original *uncompressed* network. The main advantages of this method can be outlined as follows: *(1)* the imposed low rank structure allows for large parameter reduction and fast inference via efficient matrix-vector multiplications, *(2)* each layer can be compressed independently allowing for parallel processing, *(3)* the error due to compression in each layer is controlled explicitly and *(4)* the method does not require retraining to achieve high accuracy.

3.2 Related work

Given the large size of modern DNN architectures many methods for storage and computational complexity reduction have been proposed in the literature. One commonly used technique for parameter reduction is that of network pruning [35, 4]. Network pruning assigns scores to the parameters of a pre-trained neural network and removes parameters based on these scores [16]. A key component of pruning is the need to retrain the model in order to increase accuracy to levels close to the original network. Pruning can be performed on a single parameter basis [1, 26] or by taking into account groups of parameters that ultimately lead to structured layers amenable for efficient computations [31, 33]. Another branch of parameter reduction techniques is that of low rank representation of DNN layers. In [47] the authors present a method to impose a low rank representation on the convolutional layers of CNNs. Closer to our framework, [41] proposes an approach to impose low rank structure on the last dense layer of a DNN while training.

Most of the compression techniques above require further retraining which can be computationally expensive for very large models. DNN compression without retraining is an important practical problem and recently [1] proposed Net-Trim which leverages data to perform sparsity based parameter reduction and achieve better DNN compression without retraining. However, it is very challenging to extend the compression gains to faster inference speeds as sparsity structure requires custom hardware and software support for that purpose. The off-the-shelf graphical processors (GPUs) use single instruction, multiple threads execution models, i.e. the same sequence of operations is computed in parallel on different data to accelerate matrix-vector multiplication. The speed of matrix-vector product is then directly linked to the slowest thread, which might be affected by the number of non-zeros in the computation allocation to each thread and the overhead associated with reading the non-zero entries from the chosen compressed sparse storage format. Consequently, it is quite challenging via sparsity structures, since the non-zero entries could be arbitrarily distributed, to get faster matrix-vector products. The low rank structure on the other hand does not suffer from such issues and provides faster inference using off-the-shelf hardware. Therefore, in this project we extend Net-Trim [1] to compress DNNs by imposing low rank structures on the layers.

3.3 Method

In this section we outline our DDLR method that imposes low rank representations on the weight matrices of the fully connected layers in a pre-trained DNN.

DDLR layers

In order to impose a low rank structure on the weight matrix $\mathbf{W}_\ell$ we need to minimize the $rank(\cdot)$ function of that matrix. Given the non-convexity of the $rank(\cdot)$ function we propose

solving the following optimization problem

$$\begin{aligned} \underset{\mathbf{U}\in\mathbf{R}^{n_{\ell-1}\times n_\ell}}{\text{minimize}} \quad & ||\mathbf{U}||_* \\ \text{subject to} \quad & ||\text{ReLU}(\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T) - \mathbf{Y}_\ell||_F \leq \epsilon_\ell. \end{aligned} \tag{3.1}$$

The objective function uses the well known nuclear norm relaxation of the $rank(\cdot)$ function in order to obtain a convex objective [12] while the constraint requires the output of the compressed layer to be close to the output of the original layer. This allows us to decrease the rank of the weight matrix by controlling the output error of the layer. So problem (3.1) relaxed using (2.3) can now be written as

$$\begin{aligned} \underset{\mathbf{U}\in\mathbf{R}^{n_{\ell-1}\times n_\ell}}{\text{minimize}} \quad & ||\mathbf{U}||_* \\ \text{subject to} \quad & ||(\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T - \mathbf{Y}_\ell) \circ \mathbf{M}_\ell||_F^2 \leq \epsilon_\ell^2 \\ & (\mathbf{Y}_{\ell-1}\mathbf{U} + \mathbf{1}_N\mathbf{b}_\ell^T) \circ \mathbf{M}'_\ell \leq 0, \end{aligned} \tag{3.2}$$

from which we obtain the solution $\hat{\mathbf{U}}_\ell$. This formulation allows for imposing structure on the layers while explicitly controlling for the error of the compressed layer output. This problem is a SDP with quadratic constraints and can be solved with most off-the-shelf solvers like SCS [38] and CVXOPT [2].

Parallel implementation

The optimization problem (3.2) imposes a low rank structure on a single layer of a network. To compress networks with multiple layers we can compress each layer individually and independently from each other. This process is outlined in Algorithm 2, where compression of each layer is an independent of the rest of the layers optimization problem. Given that each layer is compressed independently, the algorithm allows for parallel implementation. The algorithm requires the initial data matrix as input $\mathbf{X}^{N\times n_0}$, the original trained weight matrices and biases of the layers and the user specified tolerances ϵ_ℓ. The output is a sequence of low rank matrices for each dense layer.

It should be noted that Algorithm 2 requires the solution of a SDP (line 8) for each layer. The solution of SDPs can present a computational bottleneck when weight matrices have large dimensions (e.g. the first dense layer of VGG-16) or the number of data samples N used to solve (3.2) is large. In such cases, to alleviate these issues one can *1)* solve (3.2) to suboptimality using fewer iterations and *2)* use only a subset of the whole training set to solve (3.2). Given the size of the layers in the networks studied in the experiments section we will be solving the SDPs to optimality by using only a sample from the original dataset used for training the network.

Algorithm 2: DDLR Algorithm

Input : $\mathbf{X}, \mathbf{W}_\ell, \mathbf{b}_\ell,\ \epsilon_\ell,\ \ell = 1, \ldots, L$

$\mathbf{Y}_0 = \mathbf{X}$

for ℓ= *1,...,L* **do**

  $\mathbf{Y}_\ell = \mathrm{ReLU}(\mathbf{Y}_{\ell-1}\mathbf{W}_\ell + \mathbf{1}_N \mathbf{b}_\ell^T)$

end

for ℓ= *1,...,* L **do**

  $\mathbf{M}_\ell^{i,j} = 1$ if $\mathbf{Y}_\ell^{i,j} > 0$, otherwise 0

  $\mathbf{M}_\ell^{i,j} = 1$ if $\mathbf{Y}_\ell^{i,j} \leq 0$, otherwise 0

  Solve (3.2)

end

Output : $\hat{\mathbf{U}}_\ell,\ \ell = 1, \ldots, L$

3.4 Experiments

We demonstrate the effectiveness of our method on three different datasets. An artificial nested spiral, the MNIST and the CIFAR-10 [25] datasets. Through experimentation we concluded that compression works well when the number of data samples used to solve (3.2) is no less than 5% of the data used to train the network. For this reason, and to deal with the scaling issues of solving SDPs multiple times, we will be training the networks with a subsample of the available data and we will be solving (3.2) using a subset of size N of that sample as presented in the following subsections. For each experiment carried out we use two different values of N in order to study the performance of DDLR with respect to that sample complexity. The main method we will be comparing DDLR with is Net-Trim. Net-Trim on which our method is partially based, imposes a sparse structure on the layers post-training by minimizing the induced ℓ_1 matrix norm [1]. Both DDLR and Net-Trim can be used to compress DNNs, removing redundancies from the networks and leading to faster inference. We will be comparing the compression level and the resulting accuracies obtained from these two methods. The benefits of DDLR regarding the possible inference speedup was discussed in the related work section.

For each of the following datasets we utilize Algorithm 2 for different values of ϵ_ℓ to compress a number of the hidden layers. For each experiment we report the relative accuracy on the test set of the compressed network with respect to the original accuracy of the uncompressed network. We measure the compression by reporting the fraction of parameters needed to be stored for the compressed network with respect to the original. For the sparse matrix experiments we assume that the parameters are stored in COO format. The COO format storage requirement is three times the number of positive entries of a matrix. It should be noted that it is possible to get the same value for the rank for more than one values of ϵ_ℓ. In such cases, we choose the solution that leads to higher accuracy on the test data.

Spiral dataset

The first dataset is a spiral of two-dimensional points representing two classes. The data points were generated by sampling points on the spiral and adding i.i.d. noise uniformly distributed in the interval $[0, 3.5]$ for each dimension. In this experiment, we consider a DNN

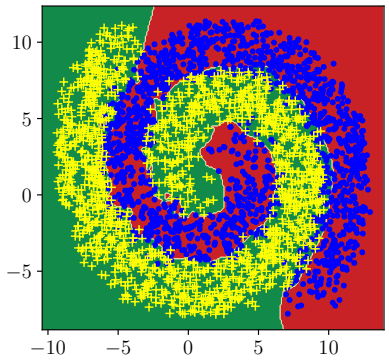


Figure 3.1: Spiral dataset and decision boundary.

classifier with two hidden layers to label the points lying on the spiral. The dataset consists of a total of 1024 points out of which 80% points were used for training the DNN and the remaining 20% were used for testing. The DNN classifier uses the ReLU activation function except for the last layer where the standard *softmax* function is used. The dimensions of the network layers are $\mathbf{W}_1 \in \mathbb{R}^{2\times 80}, \mathbf{W}_2 \in \mathbb{R}^{80\times 80}$ and $\mathbf{W}_3 \in \mathbb{R}^{80\times 2}$. The DNN was trained by minimizing the cross entropy loss using the stochastic batch gradient descent algorithm with a batch size of 32 for 1000 epochs with learning rate 0.001. The data points along with the decision boundary obtained from the trained DNN are shown in Figure 3.1.

We compress only the second layer $\mathbf{W}_2$, as the first and last ones are already low rank given their dimensions, by utilizing Algorithm 2. We implemented the DDLR algorithm under two different scenaria, one using $N = 256$ and another using $N = 512$ data points for the compression, chosen randomly from the original training dataset. In order to obtain various rank approximations we use the following values for the compression error, $\epsilon_\ell \in [0.02, 0.05, 0.08, 0.1, 0.12, 0.15, 0.2, 0.25, 0.3, 0.5, 0.6]\cdot C$, where $C = \|Y_{\ell-1}\|_F$ is used for scaling purposes. We use the elbow rule to threshold the singular values of the solution of (3.2) to obtain the final low rank weight matrix. As expected, larger values of ϵ_ℓ yield layers with lower rank. For both choices of N DDLR seems to outperform Net-Trim achieving test accuracy close to the original using only 60% of the initial parameters. The original accuracy is recovered with about 80% of the original parameters.

MNIST dataset

For the second set of experiments we use the LeNet-5 CNN architecture to classify the handwritten digits of the MNIST dataset. LeNet-5 has three dense layers of dimensions

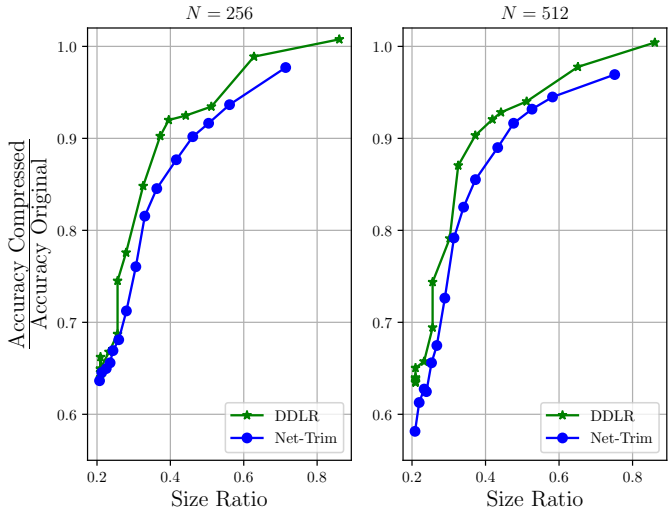


Figure 3.2: Relative test accuracy for DDLR and Net-Trim with varying DNN size ratios on the Spiral dataset.

$\mathbf{W}_1 \in \mathbb{R}^{256\times120}, \mathbf{W}_2 \in \mathbb{R}^{120\times84}$ and $\mathbf{W}_3 \in \mathbb{R}^{84\times10}$ that follow the convolutional layers. We train LeNet-5 using 1024 data samples from the original dataset, a 80%-20% train-test split, a batch size of 64, 30 epochs and a learning rate of 0.001. Using Algorithm 2 we impose a low rank structure on the first two dense layers of LeNet-5 $\mathbf{W_1}$ and $\mathbf{W}_2$ using the following values of $\epsilon_\ell = [0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.2, 0.3] \cdot C$ for each layer, with $C = \|\mathbf{Y}_{\ell-1}\|_F$ being a scaling constant. We use $N = 128$ and $N = 256$ number of samples out of the 1024 data points to compress the network. Figure 3.3 presents the relative accuracy for different size ratios. DDLR achieves high compression while maintaining sufficient accuracy. With a 70% reduction in the number of parameters DDLR can achieve a test accuracy less than 4% lower than that of the original network for both values of N while for $N = 256$ with a 40% reduction in parameters the accuracy is almost identical to the original. Interestingly, we observe that even for $N = 128$ samples, which corresponds to slightly more than 10% of the original data, we are able to obtain high compression associated with high accuracy.

CIFAR-10 dataset

For the final set of experiments we classify the CIFAR-10 image dataset using again the LeNet-5 network. For CIFAR-10 the dense layers of Lenet-5 have dimensions $\mathbf{W}_1 \in \mathbb{R}^{400\times120}$, $\mathbf{W}_2 \in \mathbb{R}^{120\times84}$ and $\mathbf{W}_3 \in \mathbb{R}^{84\times10}$. We use 2048 data points from CIFAR-10 to train our network and $N = 128$ and $N = 256$ subsamples for compression. For this experiment we compress the first two dense layers $\mathbf{W}_1$ and $\mathbf{W}_2$ using the following values for $\epsilon_\ell = [0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.2, 0.3] \cdot C$, where $C = \|\mathbf{Y}_{\ell-1}\|_F$. As expected, for larger N both methods perform better with DDLR still outperforming Net-Trim. Quite

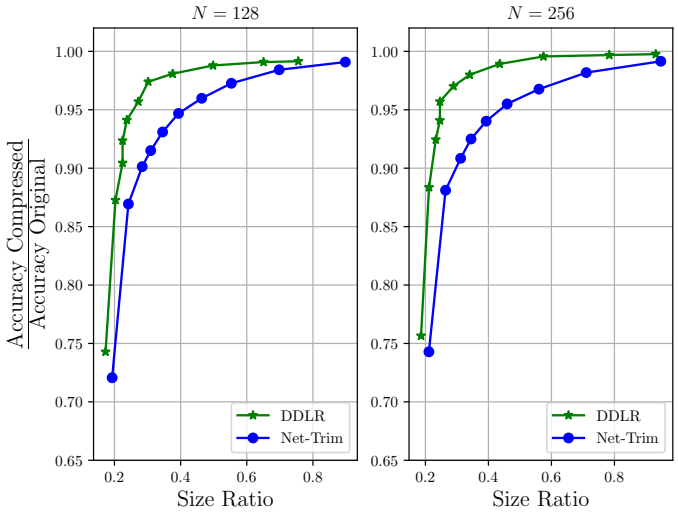


Figure 3.3: Relative test accuracy for DDLR and Net-Trim with varying DNN size ratios on the MNIST dataset.

astonishingly, we observe that with only 50% of the original parameters DDLR achieves an accuracy less than 2% worse in comparison to the accuracy of the original network for $N = 256$. For the same relative accuracy on the other hand Net-Trim reduces only by 20% the total number of parameters needed to be stored. Figure 3.4 contains the curves of the relative accuracies with respect to the parameter ratio.

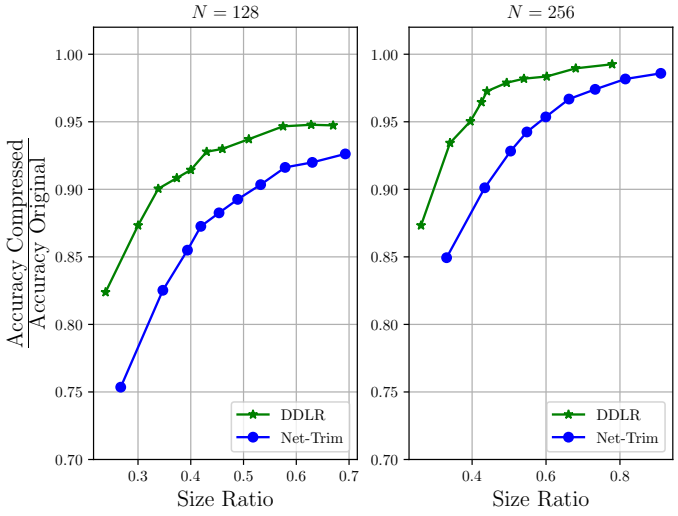


Figure 3.4: Relative test accuracy for DDLR and Net-Trim with varying DNN size ratios on the CIFAR-10 dataset.

3.5 Conclusion and future work

The DDLR Algorithm is an end-to-end approach that compresses a pre-trained DNN by imposing low rank structures on the fully connected layers while controlling for the overall accuracy decrease in the compressed DNN. We demonstrate in a number of datasets and DNN architectures that high parameter reduction can be achieved at a small loss in accuracy while requiring no retraining. Such reduction can be very significant for storing already trained models on edge devices. Furthermore, low rank structured layers allow for fast matrix-vector multiplications without the need for specialized hardware which reduce inference time, something that is of vital importance for AI applications, especially on the edge.

The results of the experiments presented in this project are rather encouraging but also limited due to a number of reasons. First, solving large scale SDPs poses a computational bottleneck. This drawback is an interesting problem that requires theoretical and algorithmic development in future research. An interesting approach in that direction is a reformulation of the optimization problem (3.2) in order to be solved using the Alternating Direction Method of Multipliers (ADMM) [5]. Such an approach can provide better scalability that will allow for compression of networks with larger hidden layers that have been trained on large datasets.

A second potential drawback lies in the parallel implementation of the DDLR Algorithm. Although this allows for a reduction in training time, this layer-wise independence can lead to lower inference accuracy as each layer is “unaware" of the previous layer’s compression. A sequential approach, as in [1], could potentially lead to better performance. Furthermore, given that modern datasets are fairly large and that our method requires sampling form the training dataset in order to compress the layers, it would be interesting to study whether a more informative sampling method, than randomly sampling, can be devised to increase performance. Finally, an interesting direction is to understand the impact of quantization on DNNs already compressed using DDLR, as such quantization can lead to significant additional reduction in the size of the networks.

Bibliography

[1] Alireza Aghasi et al. "Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3180–3189.

[2] Martin S Andersen, Joachim Dahl, and Lieven Vandenberghe. "CVXOPT: A Python package for convex optimization". In: *abel. ee. ucla. edu/cvxopt* 88 (2013).

[3] Anubhav Ashok et al. "N2n learning: Network to network compression via policy gradient reinforcement learning". In: *arXiv preprint arXiv:1709.06030* (2017).

[4] Davis Blalock et al. "What is the state of neural network pruning?" In: *arXiv preprint arXiv:2003.03033* (2020).

[5] Stephen Boyd et al. "Distributed optimization and statistical learning via the alternating direction method of multipliers". In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[6] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. "Model compression". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2006, pp. 535–541.

[7] Yu Cheng et al. "A survey of model compression and acceleration for deep neural networks". In: *arXiv preprint arXiv:1710.09282* (2017).

[8] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1251–1258.

[9] Xavier Suau Cuadros, Luca Zappella, and Nicholas Apostoloff. "Filter distillation for network compression". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2020, pp. 3140–3149.

[10] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[11] Paulo SR Diniz, Wallace A Martins, and Markus VS Lima. "Block Transceivers: OFDM and Beyond". In: *Synthesis Lectures on Communications* 5.1 (2012), pp. 1–206.

[12] Maryam Fazel, Haitham Hindi, and Stephen P Boyd. "A rank minimization heuristic with application to minimum order system approximation". In: *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*. Vol. 6. IEEE. 2001, pp. 4734–4739.

[13] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[14] I Gohberg and V Olshevsky. "Complexity of multiplication with vectors for structured matrices". In: *Linear Algebra and Its Applications* 202 (1994), pp. 163–192.

[15] Yunchao Gong et al. "Compressing deep convolutional networks using vector quantization". In: *arXiv preprint arXiv:1412.6115* (2014).

[16] Song Han et al. "Learning both weights and connections for efficient neural network". In: *Advances in neural information processing systems* 28 (2015), pp. 1135–1143.

[17] Stephen Hanson and Lorien Pratt. "Comparing biases for minimal network construction with back-propagation". In: *Advances in neural information processing systems* 1 (1988).

[18] Babak Hassibi, David G Stork, and Gregory J Wolff. "Optimal brain surgeon and general network pruning". In: *IEEE international conference on neural networks*. IEEE. 1993, pp. 293–299.

[19] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[20] Kaiming He et al. "Identity mappings in deep residual networks". In: *European conference on computer vision*. Springer. 2016, pp. 630–645.

[21] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* 2.7 (2015).

[22] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv preprint arXiv:1704.04861* (2017).

[23] Swayambhoo Jain, Shahab Hamidi-Rad, and Fabien Racapé. "Low rank based end-to-end deep neural network compression". In: *2021 Data Compression Conference (DCC)*. IEEE. 2021, pp. 233–242.

[24] Qing Jin, Linjie Yang, and Zhenyu Liao. "Adabits: Neural network quantization with adaptive bit-widths". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2020, pp. 2146–2156.

[25] Alex Krizhevsky, Geoffrey Hinton, et al. "Learning multiple layers of features from tiny images". In: (2009).

[26] César Laurent et al. "Revisiting Loss Modelling for Unstructured Pruning". In: *arXiv preprint arXiv:2006.12279* (2020).

[27] Vadim Lebedev and Victor Lempitsky. "Fast convnets using group-wise brain damage". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2554–2564.

[28] Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: (2010).

[29] Yann LeCun, John Denker, and Sara Solla. "Optimal brain damage". In: *Advances in neural information processing systems* 2 (1989).

[30] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[31] Hao Li et al. "Pruning filters for efficient convnets". In: *arXiv preprint arXiv:1608.08710* (2016).

[32] Zhuohan Li et al. "Train Big, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 5958–5968.

[33] Shaohui Lin et al. "Towards optimal structured cnn pruning via generative adversarial learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2790–2799.

[34] Baoyuan Liu et al. "Sparse convolutional neural networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 806–814.

[35] Zhuang Liu et al. "Rethinking the value of network pruning". In: *arXiv preprint arXiv:1810.05270* (2018).

[36] Christos Louizos, Max Welling, and Diederik P Kingma. "Learning Sparse Neural Networks through L_0 Regularization". In: *arXiv preprint arXiv:1712.01312* (2017).

[37] Rahul Mishra, Hari Prabhat Gupta, and Tanima Dutta. "A survey on deep neural network compression: Challenges, overview, and solutions". In: *arXiv preprint arXiv:2010.03954* (2020).

[38] B. O'Donoghue et al. "Conic Optimization via Operator Splitting and Homogeneous Self-Dual Embedding". In: *Journal of Optimization Theory and Applications* 169.3 (June 2016), pp. 1042–1068. URL: `http://stanford.edu/~boyd/papers/scs.html`.

[39] Antonio Polino, Razvan Pascanu, and Dan Alistarh. "Model compression via distillation and quantization". In: *arXiv preprint arXiv:1802.05668* (2018).

[40] Benjamin Recht, Maryam Fazel, and Pablo A Parrilo. "Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization". In: *SIAM review* 52.3 (2010), pp. 471–501.

[41] Tara N Sainath et al. "Low-rank matrix factorization for deep neural network training with high-dimensional output targets". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 6655–6659.

[42] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[43] Vikas Sindhwani, Tara Sainath, and Sanjiv Kumar. "Structured transforms for small-footprint deep learning". In: *Advances in Neural Information Processing Systems*. 2015, pp. 3088–3096.

[44] Suraj Srinivas, Akshayvarun Subramanya, and R Venkatesh Babu. "Training sparse neural networks". In: *arXiv preprint arXiv:1611.06694* (2016).

[45] Christian Szegedy et al. "Inception-v4, inception-resnet and the impact of residual connections on learning". In: *Thirty-first AAAI conference on artificial intelligence*. 2017.

[46] Christian Szegedy et al. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2818–2826.

[47] Cheng Tai et al. "Convolutional neural networks with low-rank regularization". In: *arXiv preprint arXiv:1511.06067* (2015).

[48] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *International conference on machine learning*. PMLR. 2019, pp. 6105–6114.

[49] Mingxing Tan and Quoc Le. "Efficientnetv2: Smaller models and faster training". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10096–10106.

[50] Anna Thomas et al. "Learning compressed transforms with low displacement rank". In: *Advances in neural information processing systems* 31 (2018).

[51] Erwei Wang et al. "Deep neural network approximation for custom hardware: where we've been, where we're going". In: *ACM Computing Surveys (CSUR)* 52.2 (2019), pp. 1–39.

[52] Hui Wang et al. "Progressive Blockwise Knowledge Distillation for Neural Network Acceleration." In: *IJCAI*. 2018, pp. 2769–2775.

[53] Jiaxiang Wu et al. "Quantized convolutional neural networks for mobile devices". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 4820–4828.

[54] Liang Zhao et al. "Theoretical properties for neural networks with weight matrices of low displacement rank". In: *arXiv preprint arXiv:1703.00144* (2017).

[55] Ritchie Zhao et al. "Improving neural network quantization without retraining using outlier channel splitting". In: *International conference on machine learning*. PMLR. 2019, pp. 7543–7552.

[56] Barret Zoph et al. "Learning transferable architectures for scalable image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 8697–8710.