STRUCTURED DEEP NEURAL NETWORK WITH LOW COMPLEXITY

By

SIYU LIAO

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Electrical & Computer Engineering

written under the direction of

Prof. Bo Yuan

and approved by

_____

_____

_____

_____

New Brunswick, New Jersey

October, 2020

ABSTRACT OF THE DISSERTATION

Structured Deep Neural Network with Low Complexity

By SIYU LIAO

Dissertation Director:

Prof. Bo Yuan

Deep Neural Network (DNN) has achieved great success in many fields. However, many DNN models are both deep and large thereby causing high storage and energy consumption during the training and inference phases. As the size of DNNs continues to grow, it is critical to improve computation efficiency and energy consumption while maintaining the corresponding model performance. Various methods have been proposed for compressing DNN models, which can be categorized into three different levels, model level, structure level, and weight level. This thesis focuses on structure enforcing compression algorithm and embedding quantization method which aims at: *i)* less storage and computation complexity, *ii)* easier hardware implementation because of structured memory access pattern, *iii)* natural language processing oriented embedding binarization. The first chapter introduces the motivation of this dissertation in detail. Chapter 2 goes over the background and the related work about compressing deep neural network. Chapter 3, Chapter 4 and Chapter 5 presents proposed compression methods for fully connected layer, convolution layer and embedding layer. Final chapter 6 discusses possible future directions of this research.

Acknowledgments

First and foremost, I want to thank my advisor Prof. Bo Yuan for his invaluable support and guidance during my Ph.D. study. Many of my research ideas have been influenced by his deep insight and persuasive speeches. He has always been a thoughtful and considerate advisor. There are many times when I work with him on paper submission till the last minute, even if it is midnight. He also understands foreign student's difficulties thoroughly and patiently listens to me talking about long commute, expensive tuition bills, family emergency, and even visa problems.

I am very fortunate to have Prof. Yingying Chen, Prof. Sheng Wei and Prof. Jingjin Yu as my committee members who have always been patient and supportive.

My sincere thanks also goes to Prof. Victor Pan and his student Liang Zhao for their tremendous help in my research. I have the unique privilege to discuss with these experts in the field of structure matrix. The inspiration for my early research actually came from a discussion during the office hour of Prof. Victor Pan.

Special thanks are due to Prof. Robert Haralick and Prof. Jianting Zhang who always tried to support and help by all means during the early days of my Ph.D. study.

I also wish to thank Mr. Chao Xue, Mr. Jean David Ruvini and Mr. Hetunandan Kamisetty for providing the diverse summer internship projects.

I am deeply grateful to my fellow labmates: Yi Xie, Chunhua Deng, Miao Yin, Huy Phan, Xiao Zang, Sui Yang, for the informative discussions, sleepless nights before deadlines, collaborative projects and all the delicious food we shared together. Particularly, I would like to express gratitude to Mr. Min Zhang for his constant encouragement and generous help during the most difficult days when back in New York City.

Finally, I would like to thank my parents Ziyun Liao and Chunlian Chen, and my sister Yuxin Liao who have always been supportive all these years.

Dedication

To my mom, dad and my sister

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Neural network has become one of the most powerful machine learning methods nowadays. It achieves the state-of-the-art performance in many applications, such as face recognition [8], machine translation [9] and even the game of go [10]. Compared with previous small and shallow neural networks, current networks are usually found large and deep. Therefore, people often call it by deep neural network (DNN).

Generally speaking, DNN aims at simulating human brain by passing the pulse (activation) among different neurons. Inside the network, layers of neurons are stacked in a bottom-up manner and activation is transmitted from the bottom layer to the top layer. Such activation at different layers is a transformation of input and to some degree represents certain understanding of input. The final activation output is utilized for regular machine learning tasks like regression or classification. Note that with the raw input gradually becoming into high level representation, this can save a lot of feature engineering efforts compared with traditional machine learning methods [11]. People no longer need to manually process the raw data into certain internal representation based on their domain expertise. Thus, the DNN is a representation learning method.

Although neural network was proposed decades ago, as a supervised learning method, its learning capability was limited by the data volume and the computing power. Back in 1989, the classical neural network called LeNet [12] only consists of 5 layers with 50000 training images. Thanks to the Moore's Law and the prevalence of graphics processing unit (GPU), current ResNet [13] winning the ImageNet challenge in 2015 takes 152 layers and over 1M training images. The power of DNN is finally discovered under the present data and hardware resource. More importantly, DNN is applicable to many different areas including computer vision, natural language processing, biology, computational chemistry

and so on.

Although DNN has achieved the state-of-the-art performance, people find current DNNs take much memory space and floating point operations (FLOPs). The large model size has become a challenge when deploying on resource-constrained platforms like embedded mobile devices. Fig. 1.1 shows the model size and FLOPs of various recent DNNs. All these DNN models contain millions of parameters and take Giga FLOPs for single image inference. Especially, their energy consumption also surpass the capability of regular mobile devices [14].

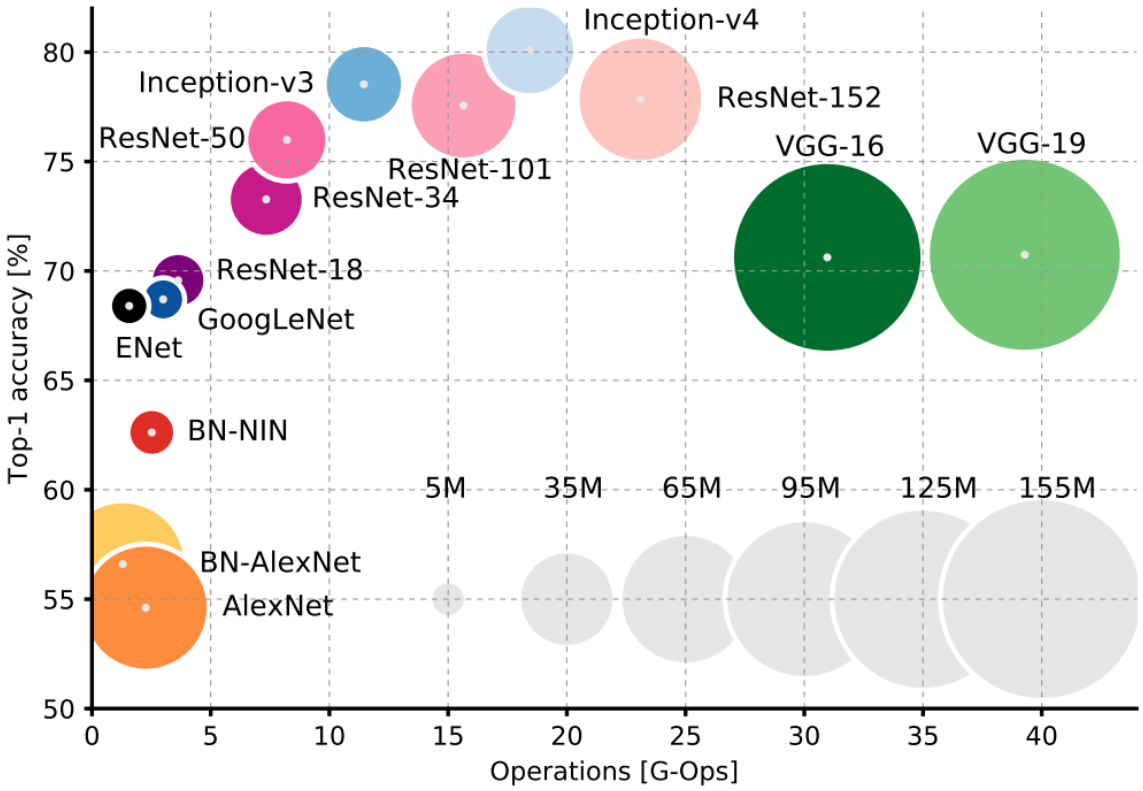


Figure 1.1: Model size and FLOPs of various recent DNNs [4].

1.1 Motivation

Many existing research works show that current DNN models are often over-parameterized [15][16][17]. There is a great potential to compress these large models into small ones. This work aims at proposing algorithms for generating DNN models with low space and time complexity. The main observation is that parameters of DNN are often organized

Table 1.1: Parameterized Layer Analysis.

Name	**CNN**	**RNN**
Convolution Layer	Y	N
Fully Connected Layer	Y	Y
Embedding Layer	N	Y
Batch Normalization Layer	Y	Y

in the form of matrix and even tensor. It is natural to compress these parameters by finding their relationship and re-organizing into a more compact form, for example low rank representation. Instead of finding the underlying hidden relation, this work compresses DNNs into certain form with a strong structure like circulant structure. More importantly, such structures are united under the same theoretical framework, i.e., low displacement rank (LDR). Unlike many other compression methods, the proposed method is theoretically sound in the context of neural network, which means the universal approximation theorem is also guaranteed after compression [6].

Table. 1.1 summarizes common parameterized layers in regular DNN model, i.e., Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN). Y stands for a layer can appear in the corresponding DNN and N stands for not appearing. Although many implementations like Tensorflow [18] name a single layer of RNN as RNN layer, it is regarded as fully connected layer in this table because their computation is essentially implemented in the same way (matrix multiplication). Other variants of DNN model like long short-term memory (LSTM) [19] and convolutional LSTM [20] are not discussed since they are also parameterized by layers mentioned in the table. Note that batch normalization layer [21] contains *much less* parameters compared with other layers, i.e., only scaling and shifting parameters. Therefore, this work only focuses on convolution layer, fully connected layer and embedding layer.

1.2 Dissertation Outline

This work proposes several algorithms to compress convolution layer, fully connected layer and embedding layer. All compression algorithms are applicable to DNNs that contain these layers inside the model architecture. In summary, the contribution of this work can

be listed as following:

- Structure matrix based compression is developed for fully connected layer and convolution layer. The training algorithm including forward and backward propagation are discussed and a warm up initialization method is also provided.
- Inspired from communications, permuted diagonal matrix based compression is developed for fully connected and convolution layer. The detailed forward and backward propagation are provided and a initialization from pre-trained model is also developed.
- Isotropic iterative quantization (IIQ) is developed for compressing word embedding. It transforms embedding into binary representation and can also serve the embedding layer for DNN models.

Chapter 2 provides the background for DNN and related compression techniques. Basic concepts and formulation are formally defined and introduced. Literature review over DNN compression is summarized and discussed in three different levels, including structure level, weight level and representation level.

Chapter 3 describes the structure matrix based compression for DNN models, which reduces the space and time complexity for convolution and fully connected layers. Especially, the convolution layer is extended from the fully connected layer by imposing structure on specific dimensions of the convolution tensor.

Chapter 4 takes the permuted diagonal matrix from the communication community and apply to compressing DNN models. Permuted diagonal matrix was proposed for regularizing the decoding matrix of linear code like LDPC [22], which is used for accelerating the underlying belief propagation. This chapter develops the forward and backward propagation of DNN based on permuted diagonal matrix.

Chapter 5 describes isotropic iterative quantization method which is inspired from the word isotropy property in natural language processing and also the locality preserving hashing (LSH) community. Therefore, IIQ is developed by combining the specific property of NLP applications with the LSH method. This method is as theoretically sound as proposed methods above and it is applicable to all point-wise mutual information (PMI) based embedding.

Chapter 2

Background

2.1 Deep Neural Network

A deep neural network (DNN) is an artificial neural network consisting of multiple layers between input $\mathcal{X}$ and output $\mathcal{Y}$. Let $\mathcal{D}$ be the data distribution over $(\mathcal{X}, \mathcal{Y})$ and let $\mathcal{S}$ be given data with N i.i.d samples drawn from $\mathcal{D}$ such that $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$, $\mathbf{x}_i \in \mathcal{X}$ and $\mathbf{y}_i \in \mathcal{Y}$. DNN describes a family of functions $\mathcal{H}$ composed of multiple differentiable layer functions $f_1(\cdot), \ldots, f_n(\cdot)$,

$$\mathcal{H} = \{f : \mathcal{X} \to \mathcal{Y} \mid \forall \mathbf{x} \in \mathcal{X},\ f(\mathbf{x}) = f_n(\ldots f_2(f_1(\mathbf{x})))\}, \tag{2.1}$$

where each layer function $f_i(\cdot)$ for $i = 1, \ldots, n$ is usually a non-linear function. We denote the output of each layer function as activation $\mathbf{a}_i = f_i(\cdot)$. For example, $f_k(\cdot)$ can be a parametric hyperbolic tangent function,

$$\mathbf{a}_k = f_k(\mathbf{a}_{k-1}) = \tanh(\mathbf{W}_k \times \mathbf{a}_{k-1} + \mathbf{b}_k), \tag{2.2}$$

where $\mathbf{W}_k$ and $\mathbf{b}_k$ are weight matrix and bias vector of the k-th layer respectively.

The learning goal of DNN is to find the mapping with the least discrepancy between network output $\mathbf{y}' = f(\mathbf{x}_i)$ and target output $\mathbf{y}$. This can be formulated as minimizing a non-negative loss function $\tilde{L}(f) = \mathbb{E}_{\mathcal{D}}[L(\mathbf{y}, \mathbf{y}')]$. Unfortunately, the true data distribution is often unknown so people try to minimize over the given training data instead. Following is an example of training loss based on Euclidean distance:

$$\min_{f \in \mathcal{H}} \hat{L}(f) = \mathbb{E}_{\mathcal{S}}[L(\mathbf{y}, \mathbf{y}')] = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{y}_i - \mathbf{y}'_i||^2. \tag{2.3}$$

However, there is generalization gap between $\hat{L}(f)$ and $\tilde{L}(f)$ since training data is limited and can't represent the true data distribution. The gap can be observed in practice that

feeding well-trained DNN with extra testing data incurs higher loss. This is because the true data distribution results in extra loss unseen in training data and the extra loss is non-negative.

Since DNN is described as an optimization problem, it is natural to calculate derivative of parameters and solve with existing optimization algorithms. For example, stochastic gradient descent is often used in training DNN based on first order derivatives. A well known algorithm for computing first order derivatives is called back-propagation algorithm [23]. The essential idea is based on the chain rule in calculus as following:

$$\hat{L}' = L'(f(\mathbf{x}))f'(\mathbf{x}) = L'(f(\mathbf{x}))f'_n \dots f'_1(\mathbf{x}). \tag{2.4}$$

It can be seen that computation goes with the expanding of function derivatives. Therefore, back-propagation algorithm means derivatives are calculated from last layer to the first layer. On the contrary, we call the process of computing from input to network output as forward propagation.

2.1.1 Convolutional Neural Network

In computer vision, the input data is in high dimensional space. For example, a gray scale image is represented as a matrix with each entry (pixel) indicating the amount of light so it is in two dimensional space. A colorful RGB image consists of three similar matrices at red, green and blue channels, respectively. This colorful image is then in three dimensional space. Traditional DNN with one dimensional input will need lots of parameters when feeding with high dimensional input data. For example, a $224 \times 224 \times 3$ colorful image will flatten into a vector with size 150528. Such over-parameterization can be a waste of resource and result in over-fitting problem.

People proposed so-called convolutional neural network (CNN) to simulate the visual cortex. Two dimensional convolution operation is applied in small region of the input image, which is also known as receptive field. CNN can save people's efforts by learning different convolutions from training data. Fig. 2.1 is an example of convolutional neural network for image classification. There are three types of layers including convolution layer, pooling layer and fully connected layer. Input image is transformed into different feature maps at

each layer. The last fully connected layer outputs the classification probability. In this way, CNN automatically extracts features from any input image and perform the classification accordingly.

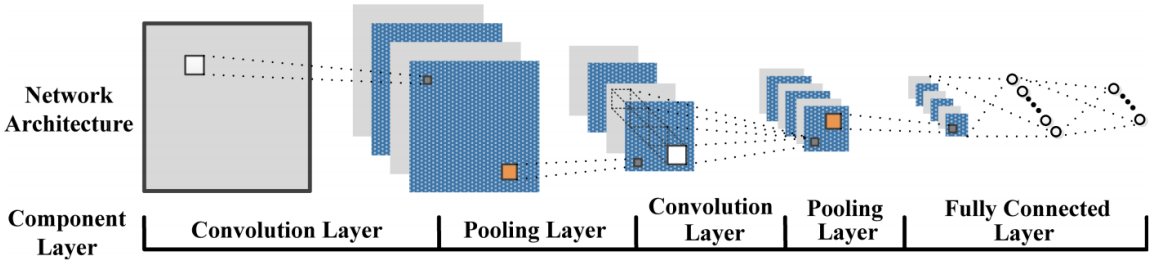


Figure 2.1: Convolutional Neural Network [5].

A convolution layer handles input $\mathbf{X} \in \mathbb{R}^{c_0 \times w_0 \times h_0}$ and outputs $\mathbf{Y} \in \mathbb{R}^{c_2 \times w_2 \times h_2}$ with convolution kernels $\mathbf{W} \in \mathbb{R}^{c_0 \times c_2 \times w_1 \times h_1}$. c_0 and c_2 are number of channels of input and output respectively. w_0, h_0 and w_2, h_2 are width and height of input and output respectively. w_1 and h_1 are the width and height of the convolution kernel. The formal computation is given as following:

$$\mathbf{Y}(m,i,j) = \sum_{n=0}^{c_0} \sum_{p=0}^{w_1} \sum_{q=0}^{y_1} \mathbf{W}(n,m,p,q) \cdot \mathbf{X}(n, i-p, j-q). \tag{2.5}$$

Although convolution layer outputs effective local feature maps, a small modification, like shifting or rotation, in input will generate different output. People use pooling layer to discard irrelevant local features against those small transformations of input. Pooling operation often takes the summation, maximum or average of local features.

Fully connected layer takes input $\mathbf{X} \in \mathbb{R}^n$ and outputs $\mathbf{Y} \in \mathbb{R}^m$ with weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$ and bias vector $\mathbf{b} \in \mathbb{R}^m$. It is essentially a linear layer but sometimes composed with point-wise non-linear function like rectified linear unit (ReLU) function:

$$\mathbf{Y} = f(\mathbf{W} \cdot \mathbf{X} + \mathbf{b}) = \max(0, \mathbf{W} \cdot \mathbf{X} + \mathbf{b}), \tag{2.6}$$

where $f(\cdot)$ is the ReLU function taking the point-wise maximum between zero and each input value.

2.1.2 Long Short-term Memory

Long short-term memory (LSTM) is a type of widely used RNN for various sequence-involved tasks. Compared with traditional RNN, LSTM alleviates the problem of graident vanishing problem when training regular RNN but may still encounter the gradient exploding problem. In general, a LSTM takes a sequence of input $\mathbf{x}_1, \ldots, \mathbf{x}_T$ and generates a sequence of output $\mathbf{y}_1, \ldots, \mathbf{y}_T$, where T is number of time steps. The computation in LSTM can be formulated for each time step $1 \leq t \leq T$ as follows:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{U}_{ir}\mathbf{y}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{U}_{fr}\mathbf{y}_{t-1} + \mathbf{b}_f) \\
\mathbf{g}_t &= h(\mathbf{W}_{gx}\mathbf{x}_t + \mathbf{U}_{gr}\mathbf{y}_{t-1} + \mathbf{b}_g) \\
\mathbf{c}_t &= \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{g}_t \circ \mathbf{i}_t \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{U}_{or}\mathbf{y}_{t-1} + \mathbf{b}_o) \\
\mathbf{y}_t &= \mathbf{o}_t \circ h(\mathbf{c}_t)
\end{aligned}, \tag{2.7}$$

where $\mathbf{W}$ and $\mathbf{U}$ indicate weight matrices and $\mathbf{b}$ indicates bias parameters for each gate in LSTM, respectively. The $\circ$ is the element-wise product. $\sigma(\cdot)$ and $h(\cdot)$ are sigmoid and hypertangent functions, respectively. The $\mathbf{i}_t$, $\mathbf{f}_t$, $\mathbf{o}_t$ are called input gate, forget gate and output gate, respectively. The $\mathbf{c}_t$ is the cell of the LSTM and those gates control the information flow in and out of the cell. Generally speaking, sequence dependencies are tracked by the cell.

2.2 DNN Compression

In this section, we introduce different DNN compression algorithms and summarize them with three different levels, i.e., model level, structure level and weight level. Note that these methods are applicable to those parametric layers in DNN.

2.2.1 Model Level

Model level compression algorithms aim at finding a compact architecture by studying the *relations between layers or models*. A typical example is knowledge distillation [24], where people distill a large model into a small model. The main idea is training the small one with soft labels from the large model because those soft labels are assumed containing more information. Many other works design a whole new DNN model that consists of a compact sub-architecture, which is composed of multiple light-weighted layers. For example, GooLeNet [25] has a compact model size because they restrict all convolutions into 1×1, 3×3 and 5×5 small convolutions. MobileNet [26] utilizes a special architecture that performs small depth-wise and 1×1 point-wise convolutions as the new convolution operation. SqueezeNet [27] proposes three compact DNN design strategies, such as switching 3×3 convolution into 1×1 convolution, reducing input channels before 3×3 convolution and late down-sampling. ShuffleNet [28] solves the problem of limited input channel for group convolution and comes up with the compact architecture composed of point-wise group convolution and channel shuffle operation.

2.2.2 Structure Level

Structure level compression algorithms focus on the *relation between weight parameters* within a single layer. A typical case is using singular value decomposition (SVD) for compressing DNN weight matrices [**?**]. It is based on the matrix rank, i.e., the dimension of the column space of the matrix. Moreover, people also apply high order tensor decomposition methods like tensor train [29], tensor ring[30], block term decomposition [31] and so on. More generally, [32] proposes to combine sparse matrix with low rank matrix for compression.

Another example is using weight sharing [15] to restrict similar weight parameters to be the same. Similarly, hashing based compressing algorithm [33] is proposed to determine the parameter relation through a hashing function. However, it is still unknown to what extent people can enforce certain relation on weight parameters. Unlike aforementioned algorithms, [34] learns a structure of sparsity inside DNN via group lasso regularization.

It is also extended to the weight tensor by grouping along one of the dimension. But this merely depends on the optimization process and people cannot set a target sparsity ahead of training.

2.2.3 Weight Level

Weight level compression algorithms aim at choosing or representing each single weight parameters. For example, it is straightforward to remove weight parameter that is very small. Pruning with threshold turns out to be a useful method in practice [15]. Since single precision (32 bits) is often implemented in practice, Another straightforward idea is using fewer bits to represent weight parameters. Low precision can directly reduce model size in general. People have explored fixed point representation and dynamic fixed point representation [35]. Furthermore, ternary bits training [36] and even binary training [37] algorithms are developed and have demonstrate their effectiveness on several standard DNN models. Unlike these uniform quantization methods, an adaptive precision selection algorithm for each layer is proposed and outperforms others over many large DNN models on ImageNet dataset.

Chapter 3

Structure Matrix

Low displacement rank (LDR) construction is a type of structure-imposing technique for network model reduction and computational complexity reduction. By regularizing the weight matrices of neural networks using the format of LDR matrices (when weight matrices are square) or the composition of multiple LDR matrices (when weight matrices are non-square), a *strong structure* is naturally imposed to the construction of neural networks. Since an LDR matrix typically requires $O(n)$ independent parameters and exhibits fast matrix operation algorithms [38], an immense space for network model and computational complexity reduction can be enabled. Pioneering work in this direction [39][40] applied special types of LDR matrices (structured matrices), such as circulant matrices and Toeplitz matrices, for weight representation. Other types of LDR matrices exist such as Cauchy matrices, Vandermonde matrices, etc., as shown in Figure 3.1.

Compared with other types of network compression approaches, the LDR construction shows several unique advantages. First, unlike heuristic weight-pruning methods [15][16] that produce irregular pruned networks, the LDR construction approach always guarantees the strong structure of the trained network, thereby avoiding the storage space and computation time overhead incurred by the complicated indexing process. Second, as a "train from scratch" technique, LDR construction does not need extra re-training, and hence eliminating the additional complexity to the training process. Third, the reduction in space complexity and computational complexity by using the structured weight matrices are significant. Different from other network compression approaches that can only provide a heuristic compression factor, the LDR construction can enable the model reduction and computational complexity reduction in Big-O complexity: The storage requirement is reduced from $O(n^2)$ to $O(n)$, and the computational complexity can be reduced from $O(n^2)$

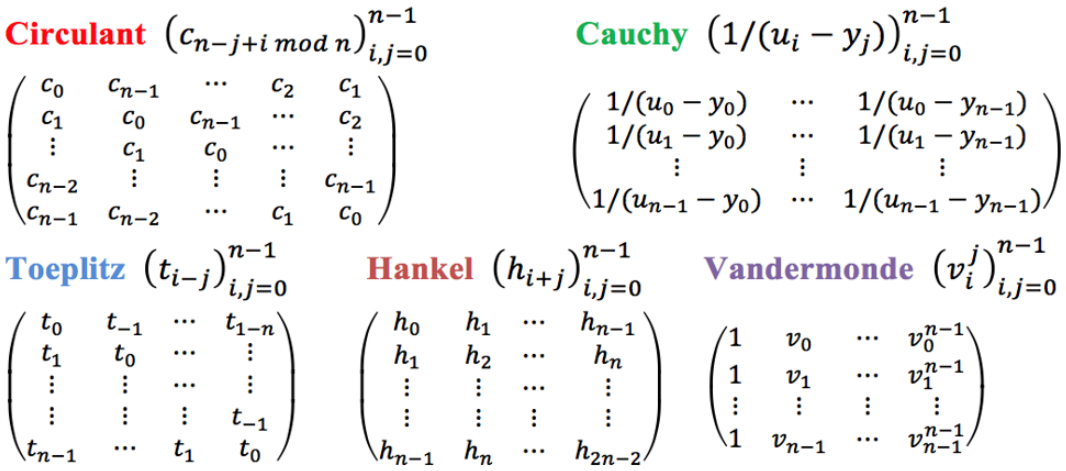


Figure 3.1: Example of commonly used LDR (structured) matrices [6], i.e., circulant, Cauchy, Toeplitz, Hankel, and Vandermonde matrices.

to $O(n \log n)$ or $O(n \log^2 n)$ because of the existence of fast matrix-vector multiplication algorithm [38][41] for LDR matrices. For example, when applying structured matrices to the fully-connected layers of AlexNet using ImageNet dataset [2], the storage requirement can be reduced by more than 4,000X while incurring negligible degradation in overall accuracy [39].

3.1 LDR Neural Networks

In this section we study the viability of applying LDR matrices in neural networks. Without loss of generality, we focus on a feed-forward neural network with one fully-connected (hidden) layer, which is similar network setup as cybenko1989approximation. Here the input layer (with n neurons) and the hidden layer (with kn neurons)[1] are assumed to be fully connected with a weight matrix $\mathbf{W} \in \mathbb{R}^{n \times kn}$ of displacement rank at most r corresponding to displacement operators $(\mathbf{A}, \mathbf{B})$, where $r \ll n$. The domain for the input vector $\mathbf{x}$ is the n-dimensional hypercube $I^n := [0,1]^n$, and the output layer only contains one neuron. The neural network can be expressed as:

$$y = G_{\mathbf{W},\theta}(\mathbf{x}) = \sum_{j=1}^{kn} \alpha_j \sigma(\mathbf{w}_j{}^T \mathbf{x} + \theta_j). \tag{3.1}$$

[1]Please note that this assumption does not sacrifice any generality because the n-by-m case can be transformed to n-by-kn format with the nearest k using zero padding [39].

Here $\sigma(\cdot)$ is the activation function, $\mathbf{w}_j \in \mathbb{R}^n$ denotes the j-th column of the weight matrix $\mathbf{W}$, and $\alpha_j, \theta_j \in \mathbb{R}$ for $j = 1, ..., kn$. When the weight matrix $\mathbf{W} = [\mathbf{w}_1|\mathbf{w}_2|\cdots|\mathbf{w}_{kn}]$ has a low-rank displacement, we call it an LDR neural network. Matrix displacement techniques ensure that LDR neural network has much lower space requirement and higher computational speed comparing to classical neural networks of the similar size.

3.1.1 Problem Statement

In this paper, we aim at providing theoretical support on the accuracy of function approximation using LDR neural networks, which represents the "effectiveness" of LDR neural networks compared with the original neural networks. Given a continuous function $f(\mathbf{x})$ defined on $[0, 1]^n$, we study the following tasks:

- For any $\epsilon > 0$, find an LDR weight matrix $\mathbf{W}$ so that the function defined by equation (4) satisfies
$$\max_{\mathbf{x}\in[0,1]^n} |f(\mathbf{x}) - G_{\mathbf{W},\theta}(\mathbf{x})| < \epsilon. \tag{3.2}$$
- Fix a positive integer n, find an upper bound ϵ so that for any continuous function $f(\mathbf{x})$ there exists a bias vector θ and an LDR matrix with at most n rows satisfying equation (3.2).
- Find a multi-layer LDR neural network that achieves error bound (3.2) but with fewer parameters.

The first task is handled in Section 3.1.2, which is the *universal approximation property* of LDR neural networks. It states that the LDR neural networks could approximate an arbitrary continuous function arbitrarily well and is the underpinning of the widespread applications. The error bounds for shallow and deep neural networks are derived in Section 5. In addition, we derived explicit back-propagation expressions for LDR neural networks in Section 3.1.4.

3.1.2 The Universal Approximation Property of LDR Neural Networks

We call a family of matrices S to have *representation property* if for any vector $\mathbf{v} \in \mathbb{R}^n$, there exists a matrix $\mathbf{M} \in S_{\mathbf{A},\mathbf{B}}$ such that v is a column of M. Note that all five types of

LDR matrices shown in Fig. **??** have this representation property because of their explicit pattern. In this section we will prove that this property also holds for many other LDR families. Based on this result, we are able to prove the universal approximation property of neural networks utilizing only LDR matrices.

Theorem 1. *Let* $\mathbf{A}$, $\mathbf{B}$ *be two* $n \times n$ *non-singular diagonalizable matrices. Define* $S^r_{A,B}$ *as the set of matrices* $\mathbf{M}$ *such that* $\Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M})$ *has rank at most* r*. Then the representation property holds for* $S^r_{A,B}$ *if* A *and* B *satisfy*

i) $\mathbf{A}^q = a\mathbf{I}$ *for some positive integer* $q \leq n$ *and a scalar* $a \neq 0$*; ii)* $(\mathbf{I} - a\mathbf{B}^q)$ *is nonsingular; iii) the eigenvalues of* $\mathbf{B}$ *have distinguishable absolute values.*

Proof. It suffices to prove for the case $r = 1$, as increasing r only provides more candidate matrices to choose from. By the property of Stein displacement, any matrix $\mathbf{M} \in S$ can be expressed in terms of $\mathbf{A}$, $\mathbf{B}$, and its displacement as follows:

$$\mathbf{M} = \sum_{k=0}^{q-1} \mathbf{A}^k \Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M}) \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1}. \tag{3.3}$$

Next we express $\Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M})$ as a product of two vectors $\mathbf{g} \cdot \mathbf{h}^T$ since it has rank 1. Also write $\mathbf{A} = \mathbf{Q}^{-1}\mathbf{\Lambda}\mathbf{Q}$, where $\mathbf{\Lambda} = \mathbf{diag}(\lambda_1, ..., \lambda_n)$ is a diagonal matrix generated by the eigenvalues of $\mathbf{A}$. Now define $\mathbf{e}_j$ to be the j-th unit column vector for $j = 1, ..., n$. Write

$$\begin{aligned}
\mathbf{Q}\mathbf{M}\mathbf{e}_j =& \mathbf{Q} \sum_{k=0}^{q-1} \mathbf{A}^k \Delta_{\mathbf{A},\mathbf{B}}(\mathbf{M}) \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1} \mathbf{e}_j \\
=& \mathbf{Q} \sum_{k=0}^{q-1} (\mathbf{Q}^{-1}\mathbf{\Lambda}\mathbf{Q})^k \mathbf{g}\mathbf{h}^T \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1} \mathbf{e}_j \\
=& \Big(\sum_{k=0}^{q-1} s_{\mathbf{h},j} \mathbf{\Lambda}^k \Big) \mathbf{Q}\mathbf{g}.
\end{aligned} \tag{3.4}$$

Here we use $s_{\mathbf{h},j}$ to denote the resulting scalar from matrix product $\mathbf{h}^T \mathbf{B}^k (\mathbf{I} - a\mathbf{B}^q)^{-1} \mathbf{e}_j$ for $k = 1, ..., n$. Define $\mathbf{T} := (\mathbf{I} - a\mathbf{B}^q)^{-1}$. In order to prove the theorem, we need to show that there exists a vector $\mathbf{h}$ and an index k such that the matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j} \mathbf{\Lambda}^k$ is nonsingular. In order to distinguish scalar multiplication from matrix multiplication, we use notation $a \circ \mathbf{M}$ to denote the multiplication of a scalar value and a matrices whenever necessary. Rewrite

the expression as

$$\begin{aligned}
&\sum_{k=0}^{q-1} s_{\mathbf{h},j}\mathbf{\Lambda}^k \\
=&\sum_{k=0}^{q-1} \mathbf{h}^T \cdot \big(\mathbf{B}^k\mathbf{T}\mathbf{e}_j \circ \mathbf{diag}(\lambda_1^k, ..., \lambda_n^k)\big) \\
=&\sum_{k=0}^{q-1} \mathbf{diag}(\mathbf{h}^T \cdot \mathbf{B}^k \cdot \mathbf{T} \cdot [\lambda_1^k\mathbf{e}_j| \cdots |\lambda_n^k\mathbf{e}_j]) \\
=&\mathbf{diag}\Big(\mathbf{h}^T \cdot \Big(\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_1^k\mathbf{e}_j\Big), ..., \mathbf{h}^T \cdot \Big(\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_n^k\mathbf{e}_j\Big)\Big).
\end{aligned}$$

The diagonal matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j}\Lambda^k$ is nonsingular if and only if all of its diagonal entries are nonzero. Let $\mathbf{b}_{ij}$ denote the column vector $\sum_{k=0}^{q-1} \mathbf{B}\mathbf{T}^k\lambda_i^k\mathbf{e}_j$. Unless for every j there is an index i_j such that $\mathbf{b}_{i_jj} = \mathbf{0}$, we can always choose an appropriate vector $\mathbf{h}$ so that the resulting diagonal matrix is nonsingular. Next we will show that the former case is not possible using proof by contradiction. Assume that there is a column $\mathbf{b}_{i_jj} = \mathbf{0}$ for every $j = 1, 2, \cdots, n$, we must have:

$$\begin{aligned}
\mathbf{0} =&[\mathbf{b}_{i_11}|\mathbf{b}_{i_22}| \cdots |\mathbf{b}_{i_nn}] \\
=&\Big[\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_{i_1}^k\mathbf{e}_1| \cdots |\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T}\lambda_{i_n}^k\mathbf{e}_n\Big] \\
=&\sum_{k=0}^{q-1} \mathbf{B}^k\mathbf{T} \cdot \mathbf{diag}(\lambda_{i_1}^k, ..., \lambda_{i_n}^k).
\end{aligned}$$

Since $\mathbf{B}$ is diagonalizable, we write $\mathbf{B} = \mathbf{P}^{-1}\mathbf{\Pi}\mathbf{P}$, where $\mathbf{\Pi} = \mathbf{diag}(\eta_1, ..., \eta_n)$. Also we have $\mathbf{T} = (\mathbf{I} - a\mathbf{B}^q)^{-1} = \mathbf{P}^{-1}(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{P}$. Then

$$\begin{aligned}
\mathbf{0} &= \sum_{k=0}^{q-1} \mathbf{B}\mathbf{T}^k\mathbf{diag}(\lambda_{i_1}^k, ..., \lambda_{i_n}^k) \\
&= \mathbf{P}^{-1}\Big[\sum_{k=0}^{q-1} \mathbf{\Pi}^k(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{diag}(\lambda_{i_1}^k, ..., \lambda_{i_n}^k)\Big]\mathbf{P} \\
&= \mathbf{P}^{-1}\sum_{k=0}^{q-1} \mathbf{diag}\Big((\lambda_{i_1}\eta_1)^k, ..., (\lambda_{i_n}\eta_n)^k\Big)(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{P} \\
&= \mathbf{P}^{-1}\mathbf{diag}\Big(\sum_{k=0}^{q-1}(\lambda_{i_1}\eta_1)^k, ..., \sum_{k=0}^{q-1}(\lambda_{i_n}\eta_n)^k\Big)(\mathbf{I} - a\mathbf{\Pi}^q)^{-1}\mathbf{P}.
\end{aligned}$$

This implies that $\lambda_{i_1}\eta_1, ..., \lambda_{i_n}\eta_n$ are solutions to the equation

$$1 + x + x^2 + \cdots + x^{q-1} = 0. \tag{3.5}$$

By assumption of matrix $\mathbf{B}$, $\eta_1, ..., \eta_k$ have different absolute values, and so are $\lambda_{i_1}\eta_1, ..., \lambda_{i_1}\eta_1$, since all λ_k have the same absolute value because $\mathbf{A}^q = a\mathbf{I}$. This fact suggests that there are q distinguished solutions of equation (3.5), which contradicts the fundamental theorem of algebra. Thus it is incorrect to assume that matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j}\mathbf{\Lambda}^k$ is singular for all $\mathbf{h} \in \mathbb{R}^n$. With this property proven, given any vector $\mathbf{v} \in \mathbb{R}^n$, one can take the following procedure to find a matrix $\mathbf{M} \in S$ and a index j such that the j-th column of $\mathbf{M}$ equals $\mathbf{v}$:

i) Find a vector $\mathbf{h}$ and a index j such that matrix $\sum_{k=0}^{q-1} s_{\mathbf{h},j}\Lambda^k$ is non-singular;

ii) By equation (3.4), find

$$\mathbf{g} := \mathbf{Q}^{-1}\big(\sum_{k=0}^{q-1} s_{\mathbf{h},j}\mathbf{\Lambda}^k\big)^{-1}\mathbf{Q}\mathbf{T}\mathbf{v};$$

iii) Construct $\mathbf{M} \in S$ with $\mathbf{g}$ and $\mathbf{h}$ by equation (3.3). Then its j-th column will equal to $\mathbf{v}$.

With the above construction, we have shown that for any vector $\mathbf{v} \in \mathbb{R}^n$ one can find a matrix $\mathbf{M} \in S$ and a index j such that the j-th column of $\mathbf{M}$ equals $\mathbf{v}$, thus the theorem is proved. □

Our main goal of this section is to show that neural networks with many types of LDR matrices (LDR neural networks) can approximate continuous functions arbitrarily well. In particular, we are going to show that Toeplitz matrices and circulant matrices, as specific cases of LDR matrices, have the same property. In order to do so, we need to introduce the following definition of a *discriminatory* function and state one of its key property as Lemma 2. A function $\sigma(u) : \mathbb{R} \to \mathbb{R}$ is called as discriminatory if the zero measure is the only measure μ that satisfies the following property:

$$\int_{I^n} \sigma(\mathbf{w}^\mathbf{T}\mathbf{x} + \theta)d\mu(\mathbf{x}) = 0, \forall \mathbf{w} \in \mathbb{R}^n, \theta \in \mathbb{R}. \tag{3.6}$$

Lemma 2. *[cf. [42]] Any bounded, measurable sigmoidal function is discriminatory.*

Now we are ready to present the universal approximation theorem of LDR neural networks with n-by-kn weight matrix $\mathbf{W}$:

Theorem 3 (Universal Approximation Theorem for LDR Neural Networks)**.** *Let σ be any continuous discriminatory function and $S^r_{A,B}$ be a family of LDR matrices having representation property. Then for any continuous function $f(\mathbf{x})$ defined on I^n and any $\epsilon > 0$, there*

exists a function $G(\mathbf{x})$ *in the form of equation* (3.1) *so that its weight matrix consists of* k *submatrices from* $S^r_{A,B}$ *and*

$$\max_{\mathbf{x}\in I^n} |G(\mathbf{x}) - f(\mathbf{x})| < \epsilon. \tag{3.7}$$

Proof. Denote the i-th $n \times n$ submatrix of $\mathbf{W}$ as $\mathbf{W}_i$. Then $\mathbf{W}$ can be written as

$$\mathbf{W} = \big[\mathbf{W}_1|\mathbf{W}_2|...|\mathbf{W}_k\big]. \tag{3.8}$$

Let S_{I^n} denote the set of all continuous functions defined on I^n. Let U_{I^n} be the linear subspace of S_{I^n} that can be expressed in form of equation (3.1) where $\mathbf{W}$ consists of k sub-matrices with displacement rank at most r. We want to show that U_{I^n} is dense in the set of all continuous functions S_{I^n}.

Suppose not, by Hahn-Banach Theorem, there exists a bounded linear functional $L \neq 0$ such that $L(\bar{U}(I^n)) = 0$. Moreover, By Riesz Representation Theorem, L can be written as

$$L(h) = \int_{I^n} h(\mathbf{x}) d\mu(\mathbf{x}), \forall h \in S(I^n),$$

for some measure μ.

Next we show that for any $\mathbf{y} \in \mathbb{R}^n$ and $\theta \in \mathbb{R}$, the function $\sigma(\mathbf{y^T x} + \theta)$ belongs to the set U_{I^n}, and thus we must have

$$\int_{I^n} \sigma(\mathbf{y^T x} + \theta) d\mu(\mathbf{x}) = 0. \tag{3.9}$$

For any vector $\mathbf{y} \in \mathbb{R}^n$, Theorem 1 guarantees that there exists an $n \times n$ LDR matrix $\mathbf{M} = [\mathbf{b}_1|\cdots|\mathbf{b}_n]$ and an index j such that $\mathbf{b}_j = \mathbf{y}$. Now define a vector $(\alpha_1, ..., \alpha_n)$ such that $\alpha_j = 1$ and $\alpha_1 = \cdots = \alpha_n = 0$. Also let the value of all bias be θ. Then the LDR neural network function becomes

$$\begin{aligned} G(\mathbf{x}) =& \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{b}_i^\mathbf{T} \mathbf{x} + \theta) \\ =& \alpha_j \sigma(\mathbf{b}_j^\mathbf{T} \mathbf{x} + \theta) = \sigma(\mathbf{y^T x} + \theta). \end{aligned} \tag{3.10}$$

From the fact that $L(G(\mathbf{x})) = 0$, we derive that

$$\begin{aligned} 0 =& L(G(\mathbf{x})) \\ =& \int_{I^n} \sum_{i=1}^{n} \alpha_i \sigma(\mathbf{b}_i^\mathbf{T} \mathbf{x} + \theta) = \int_{I_n} \sigma(\mathbf{y^T x} + \theta) d\mu(\mathbf{x}). \end{aligned}$$

Since $\sigma(t)$ is a discriminatory function by Lemma 2. We can conclude that μ is the zero measure. As a result, the function defined as an integral with measure μ must be zero for any input function $h \in S(I^n)$. The last statement contradicts the property that $L \neq 0$ from the Hahn-Banach Theorem, which is obtained based on the assumption that the set U_{I^n} of LDR neural network functions are not dense in S_{I^n}. As this assumption is not true, we have the universal approximation property of LDR neural networks. □

Reference work [39], [40] have utilized a circulant matrix or a Toeplitz matrix for weight representation in deep neural networks. Please note that for the general case of n-by-m weight matrices, either the more general Block-circulant matrices should be utilized or padding extra columns or rows of zeroes are needed [39]. Circulant matrices and Topelitz matrices are both special form of LDR matrices, and thus we could apply the above universal approximation property of LDR neural networks and provide theoretical support for the use of circulant and Toeplitz matrices in [39], [40]. Moreover, it is possible to consolidate the choice of parameters so that a block-Toeplitz matrix also shows Toeplitz structure globally. Therefore we arrive at the following corollary.

Any continuous function can be arbitrarily approximated by neural networks constructed with Toeplitz matrices or circulant matrices (with padding or using Block-circulant matrices).

3.1.3 Error Bounds on LDR Neural Networks

With the universal approximation property proved, naturally we seek ways to provide error bound estimates for LDR neural networks. We are able to prove that for LDR matrices defined by $O(n)$ parameters (n represents the number of rows and has the same order as the number of columns), the corresponding structured neural network is capable of achieving integrated squared error of order $O(1/n)$, where n is the number of parameters. This result is asymptotically equivalent to Barron's aforementioned result on general neural networks, indicating that **there is essentially no loss for restricting to LDR matrices**.

The functions we would like to approximate are those who are defined on a n-dimensional ball $B_r = \{\mathbf{x} \in \mathbb{R}^n : |\mathbf{x}| \leq r\}$ such that $\int_{B_r} |\mathbf{x}||f(\mathbf{x})|\mu(d\mathbf{x}) \leq C$, where μ is an arbitrary

measure normalized so that $\mu(B_r) = 1$. Let's call this set Γ_{C,B_r}. barron1993universal considered the following set of bounded multiples of a sigmoidal function composed with linear functions:

$$G_\sigma = \{\alpha\sigma(\mathbf{y}^\mathbf{T}\mathbf{x} + \theta) : |\alpha| \leq 2C, \mathbf{y} \in \mathbb{R}^n, \theta \in \mathbb{R}\}. \tag{3.11}$$

He proved the following theorem:

Theorem 4 (barron1993universal)**.** *For every function in Γ_{C,B_r}, every sigmoidal function σ, every probability measure, and every $k \geq 1$, there exists a linear combination of sigmoidal functions $f_k(\mathbf{x})$ of the form*

$$f_k(\mathbf{x}) = \sum_{j=1}^{k} \alpha_j \sigma(\mathbf{y}_j^\mathbf{T}\mathbf{x} + \theta_j), \tag{3.12}$$

such that

$$\int_{B_r} (f(\mathbf{x}) - f_k(\mathbf{x}))^2 \mu(d\mathbf{x}) \leq \frac{4r^2C}{k}. \tag{3.13}$$

Here $\mathbf{y}_j \in \mathbb{R}^n$ and $\theta_j \in \mathbb{R}$ for every $j = 1, 2, ..., N$, Moreover, the coefficients of the linear combination may be restricted to satisfy $\sum_{j=1}^{k} |c_j| \leq 2rC$.

Now we will show how to obtain a similar result for LDR matrices. Fix operator $(\mathbf{A}, \mathbf{B})$ and define

$$\begin{aligned} S_\sigma^{kn} = \Big\{ \sum_{j=1}^{kn} \alpha_j \sigma(\mathbf{y}_j^\mathbf{T}\mathbf{x} + \theta_j) : |\alpha_j| \leq 2C, \mathbf{y}_j \in \mathbb{R}^n, \\ \theta_j \in \mathbb{R}, j = 1, 2, ..., N, \\ \textit{and } [\mathbf{y}_{(i-1)n+1} | \mathbf{y}_{(i-1)n+2} | \cdots | \mathbf{y}_{in}] \\ \textit{is an LDR matrix, } \forall i = 1, ..., k \Big\}. \end{aligned} \tag{3.14}$$

Moreover, let G_σ^k be the set of function that can be expressed as a sum of no more than k terms from G_σ. Define the metric $||f - g||_\mu = \sqrt{\int_{B_r} (f(\mathbf{x}) - g(\mathbf{x}))^2 \mu(d\mathbf{x})}$. Theorem 4 essentially states that the minimal distance between a function $f \in \Gamma_{C,B}$ and G_σ^m is asymptotically $O(1/n)$. The following lemma proves that G_σ^k is in fact contained in S_σ^{kn}.

Lemma 5. *For any $k \geq 1$, $G_\sigma^k \subset S_\sigma^{kn}$.*

Proof. Any function $f_k(\mathbf{x}) \in G_\sigma^k$ can be written in the form

$$f_k(\mathbf{x}) = \sum_{j=1}^{k} \alpha_j \sigma(\mathbf{y}_j^\mathbf{T}\mathbf{x} + \theta_j). \tag{3.15}$$

For each $j = 1, ..., k$, define a $n \times n$ LDR matrix $\mathbf{W}_j$ such that one of its column is $\mathbf{y}_j$. Let $\mathbf{t}_{ij}$ be the i-th column of $\mathbf{W}_j$. Let i_j correspond to the column index such that $\mathbf{t}_{ij} = \mathbf{y}_j$ for all j. Now consider the following function

$$G(\mathbf{x}) := \sum_{j=1}^{k} \sum_{i=1}^{n} \beta_{ij} \sigma(\mathbf{t}_{ij}^{\mathbf{T}} \mathbf{x} + \theta_j), \tag{3.16}$$

where $\beta_{i_j j}$ equals α_j, and $\beta_{ij} = 0$ if $i \neq i_j$. Notice that we have the following equality

$$\begin{aligned} G(\mathbf{x}) :=& \sum_{j=1}^{k} \sum_{i=1}^{n} \beta_{ij} \sigma(\mathbf{t}_{ij}^{\mathbf{T}} \mathbf{x} + \theta_j) \\ =& \sum_{j=1}^{k} \beta_{i_j j} \sigma(\mathbf{t}_{ij}^{\mathbf{T}} \mathbf{x} + \theta_j) \\ =& \sum_{j=1}^{k} \alpha_j \sigma(\mathbf{y}_j^{\mathbf{T}} \mathbf{x} + \theta_j) = f_k(\mathbf{x}). \end{aligned}$$

Notice that the matrix $\mathbf{W} = [\mathbf{W}_1 | \mathbf{W}_2 | \cdots | \mathbf{W}_k]$ consists k LDR submatrices. Thus $f_k(\mathbf{x})$ belongs to the set S_σ^{kn}. □

By Lemma 5, we can replace G_σ^k with S_σ^{kn} in Theorem 4 and obtain the following error bound estimates on LDR neural networks:

Theorem 6. *For every disk $B_r \subset \mathbb{R}^n$, every function in Γ_{C,B_r}, every sigmoidal function σ, every normalized measure μ, and every $k \geq 1$, there exists neural network defined by a weight matrix consists of k LDR submatrices such that*

$$\int_{B_r} (f(\mathbf{x}) - f_{kn}(\mathbf{x}))^2 \mu(d\mathbf{x}) \leq \frac{4r^2 C}{k}. \tag{3.17}$$

Moreover, the coefficients of the linear combination may be restricted to satisfy $\sum_{k=1}^{N} |c_k| \leq 2rC$.

Theorem 6 is the first theoretical result that gives a general error bound on LDR neural networks. Empirically, [39] reported that circulant neural networks are capable of achieving the same level of accuracy as AlexNet with 1.18x-3.60x speedup and more than 90% of space saving on fully-connected layers. [40] applied Toeplitz-type LDR matrices to several benchmark image classification datasets, retaining the performance of state-of-the-art models while providing more than 3.5 compression.

The next theorem naturally extended the result from liang2016deep to LDR neural networks, indicating that LDR neural networks can also benefit a parameter reduction if one uses more than one layers. More precisely, we have the following statement:

Theorem 7. *Let f be a continuous function on $[0,1]$ and is $2n+1$ times differentiable in $(0,1)$ for $n = \lceil \log \frac{1}{\epsilon} + 1 \rceil$. If $|f^{(k)}(x)| \leq k!$ holds for all $x \in (0,1)$ and $k \in \big[2n+1\big]$, then for any $n \times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ satisfying the conditions of Theorem 1, there exists a LDR neural network $G_{\mathbf{A},\mathbf{B}}(x)$ with $O(\log \frac{1}{\epsilon})$ layers, $O(\log^2 \frac{1}{\epsilon})$ binary step units, $O(\log^3 \frac{1}{\epsilon})$ rectifier linear units such that*

$$\max_{x \in [0,1]} |f(x) - G_{\mathbf{A},\mathbf{B}}(x)| < \epsilon.$$

Proof. The theorem with better bounds and without assumption of being LDR neural network is proved in liang2016deep as Theorem 4. For each binary step unit or rectifier linear unit in the construction of the general neural network, attach $(n-1)$ dummy units, and expand the weights associated to this unit from a vector to an LDR matrix based on Theorem 1. By doing so we need to expand the number units by a factor of order $\log \frac{1}{\epsilon}$, and the asymptotic bounds are relaxed accordingly. □

3.1.4 Training LDR Neural Networks

In this section, we reformulate the gradient computation of LDR neural networks. The computation for propagating through a fully-connected layer can be written as

$$\mathbf{y} = \sigma(\mathbf{W}^T \mathbf{x} + \theta), \tag{3.18}$$

where $\sigma(\cdot)$ is the activation function, $\mathbf{W} \in \mathbb{R}^{n \times kn}$ is the weight matrix, $\mathbf{x} \in \mathbb{R}^n$ is input vector and $\theta \in \mathbb{R}^{kn}$ is bias vector. According to Equation (7), if $\mathbf{W}_i$ is an LDR matrix with operators $(\mathbf{A}_i, \mathbf{B}_i)$ satisfying conditions of Theorem 1, then it is essentially determined by two matrices $\mathbf{G}_i \in \mathbb{R}^{n \times r}, \mathbf{H}_i \in \mathbb{R}^{n \times r}$ as

$$\mathbf{W}_i = \Big[\sum_{k=0}^{q-1} \mathbf{A}_i^k \mathbf{G}_i \mathbf{H}_i^T \mathbf{B}_i^k \Big] (\mathbf{I} - a\mathbf{B}_i^q)^{-1}. \tag{3.19}$$

To fit the back-propagation algorithm, our goal is to compute derivatives $\frac{\partial O}{\partial \mathbf{G}_i}$, $\frac{\partial O}{\partial \mathbf{H}_i}$ and $\frac{\partial O}{\partial \mathbf{x}}$ for any objective function $O = O(\mathbf{W}_1, \ldots, \mathbf{W}_k)$.

In general, given that $\mathbf{a} := \mathbf{W^T x} + \theta$, we can have:

$$\frac{\partial O}{\partial \mathbf{W}} = \mathbf{x}(\frac{\partial O}{\partial \mathbf{a}})^T, \frac{\partial O}{\partial \mathbf{x}} = \mathbf{W}\frac{\partial O}{\partial \mathbf{a}}, \frac{\partial O}{\partial \theta} = \frac{\partial O}{\partial \mathbf{a}}\mathbf{1}. \tag{3.20}$$

where $\mathbf{1}$ is a column vector full of ones. Let $\hat{\mathbf{G}}_{ik} := \mathbf{A}_i^k\mathbf{G}_i$, $\hat{\mathbf{H}}_{ik} := \mathbf{H}_i^T\mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}$, and $\mathbf{W}_{ik} := \hat{\mathbf{G}}_{ik}\hat{\mathbf{H}}_{ik}$. The derivatives of $\frac{\partial O}{\partial \mathbf{W}_{ik}}$ can be computed as following:

$$\frac{\partial O}{\partial \mathbf{W}_{ik}} = \frac{\partial O}{\partial \mathbf{W}_i}. \tag{3.21}$$

According to Equation (3.20), if we let $\mathbf{a} = \mathbf{W}_{ik}$, $\mathbf{W} = \hat{\mathbf{G}}_{ik}^T$ and $\mathbf{x} = \hat{\mathbf{H}}_{ik}$, then $\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}}$ and $\frac{\partial O}{\partial \hat{\mathbf{H}}_{ik}}$ can be derived as:

$$\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}} = \big[\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}^T}\big]^T = \big[\hat{\mathbf{H}}_{ik}\frac{\partial O}{\partial \mathbf{W}_{ik}}\big]^T = (\frac{\partial O}{\partial \mathbf{W}_{ik}})^T\hat{\mathbf{H}}_{ik}^T, \tag{3.22}$$

$$\frac{\partial O}{\partial \hat{\mathbf{H}}_{ik}} = \hat{\mathbf{G}}_{ik}^T\frac{\partial O}{\partial \mathbf{W}_{ik}}. \tag{3.23}$$

Similarly, let $\mathbf{a} = \hat{\mathbf{G}}_{ik}$, $\mathbf{W} = (\mathbf{A}_i^k)^T$ and $\mathbf{x} = \mathbf{G}_i$, then $\frac{\partial O}{\partial \mathbf{G}_i}$ can be derived as:

$$\begin{aligned}\frac{\partial O}{\partial \mathbf{G}_i} &= \sum_{k=0}^{q-1}(\mathbf{A}_i^k)^T(\frac{\partial O}{\partial \hat{\mathbf{G}}_{ik}}) \\ &= \sum_{k=0}^{q-1}(\mathbf{A}_i^k)^T(\frac{\partial O}{\partial \mathbf{W}_{ik}})^T\hat{\mathbf{H}}_{ik}^T.\end{aligned} \tag{3.24}$$

Substituting with $\mathbf{a} = \hat{\mathbf{H}}_{ik}$, $\mathbf{W} = \mathbf{H}_i^T$ and $\mathbf{x} = \mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}$, we have $\frac{\partial O}{\partial \mathbf{H}_i}$ derived as:

$$\begin{aligned}\frac{\partial O}{\partial \mathbf{H}_i} &= \sum_{k=0}^{q-1}\mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}(\frac{\partial O}{\partial \hat{\mathbf{H}}_{ik}})^T \\ &= \sum_{k=0}^{q-1}\mathbf{B}_i^k(\mathbf{I} - a\mathbf{B}_i^q)^{-1}(\frac{\partial O}{\partial \mathbf{W}_{ik}})^T\hat{\mathbf{G}}_{ik}.\end{aligned} \tag{3.25}$$

3.2 Block Circulant Fully Connected Layer

In this section, we introduce the detailed circulant matrix based neural network as in [43].

3.2.1 Circulant Matrix Based Neural Network

A circulant matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$ is defined by a vector $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ as following:

$$\mathbf{W} = \begin{bmatrix} w_1 & w_n & \ldots & w_3 & w_2 \\ w_2 & w_1 & w_n & & w_3 \\ \vdots & w_2 & w_1 & \ddots & \vdots \\ w_{n-1} & & \ddots & \ddots & w_n \\ w_n & w_{n-1} & \ldots & w_2 & w_1 \end{bmatrix}. \tag{3.26}$$

It can be seen that values on diagonals are the same. Thus, the space complexity is linear $O(n)$ rather than $O(n^2)$ for general dense matrix. Moreover, the circulant matrix multiplication can be represented via FFT:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \text{IFFT}(\text{FFT}(\mathbf{w}) \circ \text{FFT}(\mathbf{x})), \tag{3.27}$$

where $\mathbf{a} \in \mathbb{R}^n$ is the multiplication result, and $\mathbf{x} \in \mathbb{R}^n$ is the input vector. The $\circ$ stands for component-wise multiplication between two vectors. The computation complexity is now reduced to $O(n \log n)$. In summary, circulant matrix representation of weight matrix can lead to low complexity in terms of both space storage and computation time.

Given that most neural network training methods require first derivative of weight parameters, we present the calculation for the first derivative of circulant matrix:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}}, \tag{3.28}$$

where L is the loss function of the neural network and $\frac{\partial \mathbf{a}}{\partial \mathbf{w}} \in \mathbb{R}^{n \times n}$ is a Jacobian matrix. Fortunately, it is found that because of circulant structured weight matrix, this computation can also be accelerated via Fast Fourier Transform [43]:

$$\frac{\partial L}{\partial \mathbf{w}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}}) \circ \text{FFT}(\mathbf{x}')), \tag{3.29}$$

where $\mathbf{x}' = (x_1, x_n, x_{n-1}, \ldots, x_2)$.

According to the back-propagation algorithm [23], we also provide the first derivative of vector $\mathbf{a}$ so that it can be used for gradient computation of the other layer:

$$\frac{\partial L}{\partial \mathbf{x}} = \frac{\partial L}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{x}}, \tag{3.30}$$

where $\frac{\partial \mathbf{a}}{\partial \mathbf{x}} \in \mathbb{R}^{n \times n}$ is also an Jacobian matrix. Similarly, this can also be accelerated via Fast Fourier Transform [43]:

$$\frac{\partial L}{\partial \mathbf{x}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}}) \circ \text{FFT}(\mathbf{w}')), \tag{3.31}$$

where $\mathbf{w}' = (w_1, w_n, w_{n-1}, \ldots, w_2)$.

3.2.2 Block Circulant Based Neural Network

Note that circulant matrix is always a square matrix. In practice, neural network layer size can be arbitrarily determined according to different applications. Therefore, weight matrices are often not square as desired. To fit with a general setting of weight matrix $\mathbf{W} \in \mathbb{R}^{m \times n}$, a straightforward design is to split weight matrix into square blocks. These blocks are set to be circulant matrix, and the entire matrix is then called block circulant matrix [43].

Let b be the block size and there are $\frac{m}{b} \times \frac{n}{b}$ blocks in total. Denote these block with $\mathbf{C}_{i,j} \in \mathbb{R}^{b \times b}$ for $i = 1, \ldots, \frac{m}{b}$ and $j = 1, \ldots, \frac{n}{b}$. Then weight matrix multiplication is reformulated as below:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \begin{bmatrix} \mathbf{C}_{1,1} & \ldots & \mathbf{C}_{1,\frac{n}{b}} \\ \vdots & & \vdots \\ \mathbf{C}_{\frac{m}{b},1} & \ldots & \mathbf{C}_{\frac{m}{b},\frac{n}{b}} \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \ldots \\ \mathbf{a}_{\frac{m}{b}} \end{bmatrix}, \tag{3.32}$$

where $\mathbf{a}_i \in \mathbb{R}^b$ is a column vector. Since each $\mathbf{C}_{i,j}$ is a circulant matrix, we define $\mathbf{w}_{i,j}$ as the vector determining the matrix. Similarly, $\mathbf{a}_i$ can also be computed via FFT as aforementioned:

$$\mathbf{a}_i = \sum_{j=1}^{n/b} \mathbf{C}_{i,j}\mathbf{x}_j = \text{IFFT}(\sum_{j=1}^{n/b} \text{FFT}(\mathbf{w}_{i,j}) \circ \text{FFT}(\mathbf{x}_j)), \tag{3.33}$$

where $\mathbf{x}_j \in \mathbb{R}^b$ is a slice of vector $\mathbf{x}$ corresponding to the block $\mathbf{C}_{i,j}$.

Moreover, the first derivative of block circulant matrix is provided as follows:

$$\frac{\partial L}{\partial \mathbf{w}_{i,j}} = \frac{\partial L}{\partial \mathbf{a}_i}\frac{\partial \mathbf{a}_i}{\partial \mathbf{w}_{i,j}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{x}'_j)), \tag{3.34}$$

$$\begin{aligned}\frac{\partial L}{\partial \mathbf{x}_j} &= \sum_{i=1}^{m/b} \frac{\partial L}{\partial \mathbf{a}_i} \frac{\partial \mathbf{a}_i}{\partial \mathbf{x}_j} \\ &= \text{IFFT}(\sum_{i=1}^{m/b} \text{FFT}(\frac{\partial L}{\partial \mathbf{a}_i}) \circ \text{FFT}(\mathbf{w}'_{i,j}))\end{aligned}, \tag{3.35}$$

where $\mathbf{w}'_{i,j} \in \mathbb{R}^b$ and $\mathbf{x}'_j \in \mathbb{R}^b$ are defined similarly as in Eq. 3.29 and Eq. 3.31, respectively.

In summary, block circulant matrix takes $\frac{mn}{b}$ number of parameters and the multiplication complexity is $O(\frac{mn}{b} \log b)$. We also present the corresponding pseudocode for matrix multiplication and derivative computation as follows. The forward propagation algorithm is for block circulant matrix multiplication. The backward propagation algorithm is for computing the first derviative of block circulant matrix and the input vector.

3.2.3 Circulant Approximation

Although we have described the training algorithm for circulant matrix based neural network, this requires training the entire network from scratch. In this section, we introduce another training option, i.e., converting from an existing dense neural network model to a circulant structured neural network model. Compared with training from scratch, this method can provide a warm up initialization which can converge faster than random initialization.

The essential idea is based on the circulant approximation algorithm as in [44]. For a dense matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$, let $\mathbf{w} \in \mathbb{R}^n$ be the vector for its closest circulant matrix. The optimal $\mathbf{w}$ can be found via following:

$$w_k = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{n} \mathbf{W}_{i,j} \times \mathbb{1}\big[\mathbf{W}_{i,j}\big]_{i-j \bmod n = k}, \tag{3.36}$$

where $w_k \in \mathbf{w}$ for $k = 1, \ldots, n$, and $\mathbb{1}\big[\,\cdot\,\big]$ is an indicator function. Here optimal is achieved under the Frobenius norm of the difference of given dense matrix and the approximated circulant matrix. Moreover, for the block circulant matrix, it is applicable to each block to achieve the approximation. The approximation algorithm is also summarized in pseudocode as below.

Instead of using an indicator function, it is easier to directly accumulate over the target weight parameter. Thus, using for loops to sweep over all possible indices can also give the

Algorithm 1: Circulant Approximation

Input: **W**
Output: **w**
initialize **w** as a zero vector;
for $i \leftarrow 1$ *until* n **do**
  for $j \leftarrow 1$ *until* n/b **do**
    $k \leftarrow (i - j) \bmod n$;
    $w_k \leftarrow w_k + \mathbf{W}_{i,j}$;
  end
end
for $i \leftarrow 1$ *until* n **do**
  $w_i \leftarrow w_i/n$;
end
return **w**;

circulant approximation. This algorithm can be applied to weight matrices inside a given pre-trained dense model and generate weight matrices for the target circulant structured model. As a result, this naive algorithm design has the $O(n^2)$ computation complexity.

3.3 Block Circulant Convolution Layer

In general, a convolutional layer maps a 3-dimensional input tensor $\mathcal{X} \in \mathbb{R}^{W_0 \times H_0 \times C_0}$ into a 3-dimensional output tensor $\mathcal{Y} \in \mathbb{R}^{W_2 \times H_2 \times C_2}$ through convolution with a 4-dimensional kernel tensor $\mathcal{W} \in \mathbb{R}^{W_1 \times H_1 \times C_0 \times C_2}$. Here W_i and H_i for $i = 0, 1, 2$, are the spatial width and height of the input, kernel, and output tensor, respectively; C_0 and C_2 are the number of input channels and output channels. The convolution operation is expressed as follows:

$$\mathcal{Y}(w_2, h_2, c_2) = \sum_{w_1=1}^{W_1} \sum_{h_1=1}^{H_1} \sum_{c_0=1}^{C_0} \Big(\mathcal{X}(w_2 - w_1, h_2 - h_1, c_0) \cdot \mathcal{W}(w_1, h_1, c_0, c_2) \Big). \tag{3.37}$$

Although stride can be set for convolution, we consider the case of stride that equals to 1 to make a better understanding of circulant convolution. It should be noted that stride wouldn't affect our convolution algorithm and design. Moreover, we can express Eq. 4.8 in the form of a fiber multiplied by a slice as below:

$$\mathcal{Y}(w_2, h_2, :) = \sum_{w_1=1}^{W_1} \sum_{h_1=1}^{H_1} \Big(\mathcal{X}(w_2 - w_1, h_2 - h_1, :) * \mathcal{W}(w_1, h_1, :, :) \Big), \tag{3.38}$$

where $*$ and : denote the matrix-vector multiplication and the range of indices, respectively.

Circulant convolutional layer. Different from a conventional convolutional layer, the circulant convolutional layer has a weight tensor $\mathcal{W}$ that exhibits circulant structure. In other words, the $\mathcal{W}$ of a circulant convolution layer is a 4D *circulant tensors* [45]. In general, a circulant tensor can exhibit circulant structure along any pair of its dimensions. However, as W_1 and H_1 are usually much smaller than C_0 and C_2 for tensor $\mathcal{W}$, we impose the circulant structure along the input channel and output channel dimensions to achieve high model-size compression ratio. Note that in practice we need to partition the tensor $\mathcal{W}$ into circulant sub-tensors of size $W_1 \times H_1 \times N \times N$. This is necessary because the circulant structure requires that the two corresponding dimension must be equal, while C_0 and C_2 are usually not the same. Larger N means larger compression ratio but it could hurt the model performance to some degree. By adjusting the partition size N we can balance the trade-off between compression ratio and model accuracy.

More specifically, let N be the partition size with $C_0 = R \times N$ and $C_2 = S \times N$ [2], then $\mathcal{W}$ can be defined by a 4-dimensional base tensor $\mathcal{W}' \in \mathbb{R}^{W_1 \times H_1 \times RN \times S}$:

$$\mathcal{W}(w_1, h_1, c_0, c_2) = \mathcal{W}'(w_1, h_1, p, q), \tag{3.39}$$

where p, q are indices satisfying $\lfloor c_0/N \rfloor = \lfloor p/N \rfloor$, $\lfloor c_2/N \rfloor = q$, and $c_0 - c_2 \equiv p \pmod{N}$. Fig. 3.2 illustrates the circulant structure of weight tensor $\mathcal{W}$. From this figure, it can be seen that the circulant structure is imposed to $\mathcal{W}$ along the input/output channel dimensions. The block-circulant weight tensor consists of six circulant weight sub-tensors, where different colors represent different circulant weight sub-tensors. Each circulant weight sub-tensor consists of sixteen kernel filters that are represented in different colors such as green and yellow.

3.3.1 Fast Forward and Backward Propagation

Eq. 3.39 shows that the weight tensor $\mathcal{W}$ of a circulant convolutional layer exhibits the circulant structure and has the reduced number of independent parameters. Besides, according to the tensor theory [45], circulant tensor also has the advantage of fast multiplication.

[2] Zero-padding is needed when N does not divide C_0 or C_2.

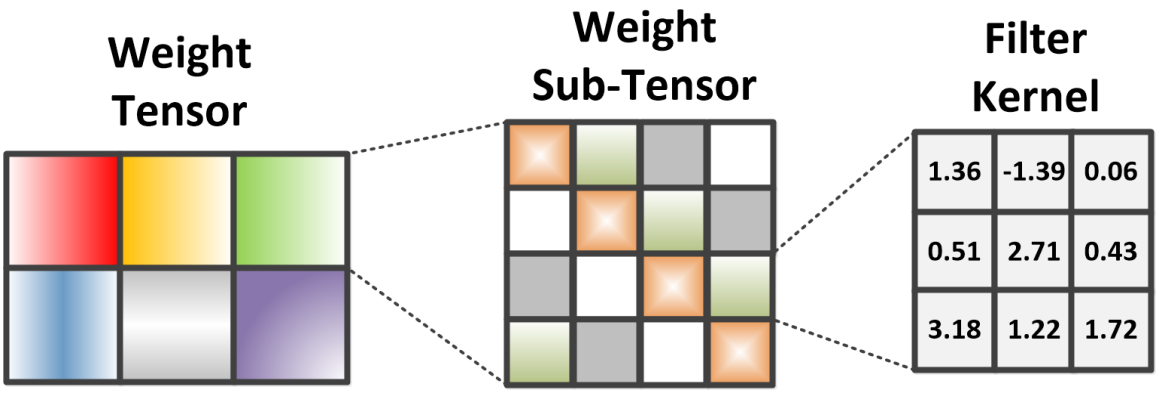


Figure 3.2: Illustration of a circulant weight tensor [7]. Blocks of the same color in the middle share the same set of kernel weights (on the right). This significantly reduces the total amount of parameters needed to represent this tensor. In addition, the placement of blocks displays a circulant structure, facilitating FFT-based fast algorithms.

Since multiplication is the kernel computation in neural network training and inference, the existence of fast multiplication of circulant tensor enables the immediate reduction in computational cost. Next, we describe the fast forward and backward propagation schemes by leveraging the fast multiplication of circulant weight tensor.

Fast forward propagation. We first present the fast forward propagation scheme. Recall that Eq. 3.38 is the forward propagation scheme for a general convolutional layer. To ease the notation, define $N_k = ((k-1)N+1, ..., kN)$ for $k = 1, ..., \max(R, S)$, and rewrite Eq. 3.38 as below:

$$\mathcal{Y}(w_2, h_2, N_i) = \sum_{w_1=1}^{W_1} \sum_{h_1=1}^{H_1} \sum_{j=1}^{R} \Big(\mathcal{X}(w_2 - w_1, h_2 - h_1, N_j) * \mathcal{W}(w_1, h_1, N_j, N_i) \Big), \tag{3.40}$$

where $i \in \{1, \ldots, S\}$. According to [45, 46], Fast Fourier Transform (FFT) can be used to accelerate the multiplication of a fiber and a slice of circulant tensor with time complexity reduced from $O(N^2)$ to $O(N \log N)$. Therefore, when $\mathcal{W}$ is a circulant tensor, Eq. 3.40 can be reformulated using FFT as below:

$$\mathcal{Y}(w_2, h_2, N_i) = \text{ifft}\Big(\sum_{w_1=1}^{W_1} \sum_{h_1=1}^{H_1} \sum_{j=1}^{R} \text{fft}\big(\mathcal{X}(w_2 - w_1, h_2 - h_1, N_j)\big) \circ \text{fft}\big(\mathcal{W}'(w_1, h_1, N_j, N_i)\big) \Big). \tag{3.41}$$

Here $\circ$ is the element-wise multiplication.

Fast backward propagation. Now consider backward propagation. Given loss function L, it is well known that the goal of backpropgation algorithm [12] is to compute gradients of loss function L with respect to each weight and input. Hence according to the chain rule, the gradient computation for circulant convolutional layer can be derived from Eq. 3.39 and Eq. 3.40 as below:

$$\frac{\partial L}{\partial \mathcal{W}'(w_1,h_1,p,q)} = \sum_{w_2=1}^{W_2}\sum_{h_2=1}^{H_2}\sum_{c_2=(q-1)N+1}^{qN} \frac{\partial L}{\partial \mathcal{Y}(w_2,h_2,c_2)}\frac{\partial \mathcal{Y}(w_2,h_2,c_2)}{\partial \mathcal{W}'(w_1,h_1,p,q)}, \tag{3.42}$$

$$\frac{\partial L}{\partial \mathcal{X}(x,y,c_0)} = \sum_{w_1=1}^{W_1}\sum_{h_1=1}^{H_1}\sum_{c_2\equiv c_0(\ mod\ N)} \frac{\partial L}{\partial \mathcal{Y}(w_1+x,h_1+y,c_2)}\frac{\partial \mathcal{Y}(w_1+x,h_1+y,c_2)}{\partial \mathcal{X}(x,y,c_0)}. \tag{3.43}$$

Again, according to [45], when $\mathcal{W}$ is a circulant tensor, Eq. 3.42 and Eq. 3.43 can also be accelerated by using FFT as below:

$$\frac{\partial L}{\partial \mathcal{W}'(w_1,h_1,N_j,i)} = \text{ifft}(\sum_{w_2=1}^{W_2}\sum_{h_2=1}^{H_2}\text{fft}(\frac{\partial L}{\partial \mathcal{Y}(w_2,h_2,N_i)}) \circ \text{fft}(\mathbf{x}'_{\mathbf{j}})), \tag{3.44}$$

$$\frac{\partial L}{\partial \mathcal{X}(x,y,N_j)} = \text{ifft}(\sum_{w_1=1}^{W_1}\sum_{h_1=1}^{H_1}\sum_{i=1}^{S}\text{fft}(\frac{\partial L}{\partial \mathcal{Y}(w_1+x,h_1+y,N_i)}) \circ \text{fft}(\mathbf{w}'_{\mathbf{j},\mathbf{i}})), \tag{3.45}$$

where $\mathbf{x}'_{\mathbf{j}}$ and $\mathbf{w}'_{\mathbf{j},\mathbf{i}}$ are fibers $\mathcal{X}(w_1,h_1,T)$ and $\mathcal{W}'(x,y,(j-1)N+T,i)$ with $T=(1,N,...,2)$.

Capability of training Circulant CNN from scratch. It should be noted that the gradient computations described in Eq. 3.44 and Eq. 3.45 are actually based on $\mathcal{W}'$. Since we can always construct the circulant tensor $\mathcal{W}$ from base tensor $\mathcal{W}'$ using Eq. 3.39, Eq. 3.44 and 3.45 imply that the circulant structure of weight tensor $\mathcal{W}$ is always kept during the training phase. In other words, if we initialize $\mathcal{W}$ as the circulant tensor at the initialization stage of training, then during the training procedure Eq. 3.44 and 3.45 can guarantee $\mathcal{W}$ always exhibit circulant structure. Therefore, a circulant CNN can be completely trained from the scratch.

3.3.2 Conversion from Non-circulant Tensor to Circulant Tensor

Forward and backward propagation section indicates that a circulant convolutional layer can be trained from scratch. In this subsection, we also present a conversion technique that

can directly convert a non-circulant weight tensor to a circulant one. Such conversion is very useful when a pre-trained model is already available and needs to be imposed with circulant structure.

Specifically, the proposed conversion technique is based on the circulant approximation approach [47] used for circulant matrix. In matrix theory, let $\mathbf{Z_1} \in \mathbb{R}^{N\times N}$ denote a permutation matrix as following:

$$\mathbf{Z_1} = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & & \ddots & & 1 \\ 1 & 0 & 0 & \dots & 0 \end{bmatrix}. \tag{3.46}$$

Then a circulant matrix $\mathbf{W}_{circ} \in \mathbb{R}^{N\times N}$ with its first row $\mathbf{w} = (w_0, w_1, \dots, w_{N-1})$ can be represented in the polynomial form of $\mathbf{Z_1}$ as follows:

$$\mathbf{W}_{circ} = \textstyle\sum_{i=0}^{N-1} w_i {\mathbf{Z_1}}^i. \tag{3.47}$$

According to [47], for a non-circulant matrix $\mathbf{W}_{non-circ} \in \mathbb{R}^{N\times N}$, its nearest circulant matrix $\mathbf{W}_{circ}$ (measured in the Frobenius norm) is given by projection:

$$\begin{aligned} \mathbf{w} &= proj_N \mathbf{W}_{non-circ}, \\ \forall w_i \in \mathbf{w}, w_i &= \frac{1}{N}\langle \mathbf{W}_{non-circ}, {\mathbf{Z_1}}^i \rangle_{\mathbf{F}}, \end{aligned} \tag{3.48}$$

where $\langle \cdot, \cdot \rangle_{\mathbf{F}}$ is the Frobenius inner product.

Note that the 4-D weight tensor of a convolutional layer can be viewed as a matrix of size $W_1 \times W_2$ where each entry is a matrix of size $C_0 \times C_2$. Therefore, by using Eq. 3.48, the conversion from a non-circulant tensor $\mathcal{W}_{non-circ}$ to a circulant tensor $\mathcal{W}_{circ}$ can be achieved by performing the projection as follows:

$$\mathcal{W}'(w_1, h_1, N_j, i) = proj_N \mathcal{W}_{non-circ}(w_1, h_1, N_j, N_i), \tag{3.49}$$

where $\mathcal{W}'$ is the base tensor that defines circulant tensor $\mathcal{W}_{circ}$, and the mapping from $\mathcal{W}'$ to $\mathcal{W}_{circ}$ is given in Eq. 3.39.

Capability of training Circulant CNN from a pre-trained model. Based on the conversion scheme shown in Eq. 3.49, any non-circulant convolutional layer of a pre-trained model can be directly converted to a circulant convolutional layer. Typically such

Table 3.1: Comparison with [1] in terms of FFT, time and space complexity, where $N = C_0 = C_2$.

Approach	Time Complexity	Space Complexity	FFT Type
Original	$O(W_2H_2W_1H_1N^2)$	$O(W_1H_1N^2)$	N/A
This Work	$O(W_2H_2W_1H_1N\log N)$	$O(W_1H_1RNS)$	1-D
[1]	$O(N^2W_0H_0\log W_0H_0)$	$O(W_1H_1N^2)$	2-D

direct conversion brings non-negligible accuracy drop incurred by the approximation error. In order to recover the accuracy, further re-training on the converted model is needed by following the backward propagation scheme in Eq. 3.44 and 3.45. Consequently, a non-circulant pre-trained model can be imposed with circulant structure by using the proposed circulant conversion and re-training schemes with preserving high accuracy.

3.3.3 Efficiency on Space and Computation

Table 3.1 summarizes the space and time complexity of the circulant convolutional layers. It can be seen that the proposed circulant structure-imposing approach enables simultaneous improvement on both space efficiency and computation efficiency. Larger N can result in larger FFT size and lower space and time complexity. Also, compared with the 2-D FFT-based fast convolution in [1], our 1-D FFT-based approach has much lower space and time complexity since N is typically much larger than R and S.

3.4 Block Toeplitz Fully Connected Layer

In this section the proposed approach is to utilize Toeplitz matrix to represent weight matrices in DNN models. Mathematically, a Toeplitz matrix $\mathbf{W} \in \mathbb{R}^{n\times n}$ can be defined by a vector $\mathbf{w} = (w_{1-n}, w_{2-n}, \ldots, w_0, \ldots, w_{n-1})$, where w_i is a scalar for $1-n \leq i \leq n-1$:

$$\mathbf{W} = \begin{bmatrix} w_0 & w_{-1} & \ldots & w_{1-n} \\ w_1 & w_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & w_{-1} \\ w_{n-1} & \ldots & w_1 & w_0 \end{bmatrix}. \tag{3.50}$$

It should be noted that there are $2n-1$ parameters when defining a square Toeplitz matrix; while the conventional unstructured matrix with the same size contains n^2 parameters.

3.4.1 Impose Toeplitz Structured on DNNs

Section 2.2 shows that the Toeplitz matrix has much lower space complexity ($O(n)$) than conventional matrix ($O(n^2)$). Encouraged by this characteristics, we propose to impose Toeplitz structure on the construction of DNN models. In other words, the weight matrices of layers of DNNs are now enforced to be Toeplitz matrices. To accommodate this change, both the forward propagation and backward propagation schemes need to be reformulated as follows.

Forward propagation: To perform Toeplitz matrix-based forward propagation, a straightforward method is to simply replace $\mathbf{W}$ in forward propagation with Toeplitz format and conduct matrix-vector multiplication. Although this simple change works, it is not optimal in efficiency. This is because as a type of structured matrix, Toeplitz matrix is inherently affiliated with fast matrix-vector multiplication. Specifically, as pointed out in [46], the multiplication between Toeplitz matrix and vector can be performed using Fast Fourier Transform (FFT) and its inverse (IFFT) as follows:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \text{IFFT}(\text{FFT}(\mathbf{w}') \circ \text{FFT}(\mathbf{x}'))_{1:n}, \tag{3.51}$$

where $\mathbf{w}' = (w_0, \ldots, w_{1-n}, 0, w_{n-1}, \ldots, w_1) \in \mathbb{R}^{2n}$, $\mathbf{x}' = (\mathbf{x}, 0, \ldots, 0) \in \mathbb{R}^{2n}$, $\mathbf{x} \in \mathbb{R}^n$ is the original input, and $\circ$ means the element-wise product. The subscript $1:n$ means that we take the first n elements from IFFT result as vector $\mathbf{a} \in \mathbb{R}^n$. Notice that because the computational complexity of FFT/IFFT is $O(n \log n)$, the overall computational complexity is reduced from $O(n^2)$ to $O(n \log n)$ and the space complexity is reduced from $O(n^2)$ to $O(n)$ too. This means that imposing Toeplitz structure on DNN models can save the storage and accelerate the execution simultaneously.

Backward propagation: The essence of backward propagation is to calculate gradients. Similar to the procedure proposed in [39], we derive the gradient computation given objective function L with respect to $\mathbf{w}'$ as follows:

$$\frac{\partial L}{\partial \mathbf{w}'} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'}) \circ \text{FFT}(\mathbf{x}'')), \tag{3.52}$$

where $\mathbf{x}'' = (x_1, 0, \ldots, 0, x_n, \ldots, x_2) \in \mathbb{R}^{2n}$, and $\frac{\partial L}{\partial \mathbf{a}'} = (\frac{\partial L}{\partial \mathbf{a}}, 0, \ldots, 0) \in \mathbb{R}^{2n}$. Moreover, the

gradients of input $\mathbf{x}$ can be also calculated as:

$$\frac{\partial L}{\partial \mathbf{x}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'}) \circ \text{FFT}(\mathbf{w}''))_{1:n}, \tag{3.53}$$

where $\mathbf{w}'' = (w_0, w_1, \ldots, w_{n-1}, 0, w_{1-n}, \ldots, w_{-1}) \in \mathbb{R}^{2n}$, and the first n elements of IFFT result will be vector $\frac{\partial L}{\partial \mathbf{x}}$.

3.4.2 Impose Block-Toeplitz Structure on DNNs

From the perspective of deployment, simply imposing square Toeplitz structure on DNN models is challenging. This is because using Toeplitz matrix renders a fixed compression ratio while in practice it always requires flexibility for compression effect. To address this challenge, we propose to impose block-Toeplitz structure on the construction of DNNs. In general, a block-Toeplitz matrix consists of multiple square Toeplitz matrices, and it can fit weight matrices in any shape [3]. Accordingly, the forward and backward propagation schemes need to be re-investigated in more general scenarios.

Forward propagation: Let $\mathbf{W} \in \mathbb{R}^{m \times n}$ be the weight matrix, which is divided into multiple square blocks in size $b \times b$. There are $m/b \times n/b$ blocks and each is a Toeplitz matrix $\mathbf{w}_{ij}$ that can be defined using $2b-1$ weight parameters, where $i \in \{1, \ldots, m/b\}$, $j \in \{1, \ldots, n/b\}$. Similarly we can divide the input vector $\mathbf{x}$ into different $\mathbf{x}_j \in \mathbb{R}^b$, and output vector $\mathbf{a}$ into different $\mathbf{a}_i \in \mathbb{R}^b$. Then the product of matrix and vector can be re-written as follows:

$$\mathbf{a} = \mathbf{W}\mathbf{x} = \begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_{m/b} \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n/b} \mathbf{w}_{1j}\mathbf{x}_j \\ \vdots \\ \sum_{j=1}^{n/b} \mathbf{w}_{m/b,j}\mathbf{x}_j \end{bmatrix}, \tag{3.54}$$

where the calculation of $\mathbf{a}_i$ can be accelerated with FFT/IFFT as in Eqn. 3.51. Algorithm 4 summarizes the scheme of block-Toeplitz matrix-based forward propagation.

Backward propagation: In the scenario of using block Toeplitz matrix, similar to the derivation in Section 3.1, the corresponding calculation of gradient descent can also be

[3] Zero-padding may be required If one dimension of matrix is not the multiple of another one. This will not cause storage or computation overhead.

written as:

$$\frac{\partial L}{\partial \mathbf{w}'_{ij}} = \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'_i}) \circ \text{FFT}(\mathbf{x}''_j)), \tag{3.55}$$

$$\frac{\partial L}{\partial \mathbf{x}_j} = \sum_{i=1}^{m/b} \text{IFFT}(\text{FFT}(\frac{\partial L}{\partial \mathbf{a}'_i} \circ \text{FFT}(\mathbf{w}''_{ij}))_{1:b}, \tag{3.56}$$

where $\mathbf{a}'_i$, $\mathbf{x}''_j$ and $\mathbf{w}'_{ij}$ are defined similarly as in Eqn. 3.52, and $\mathbf{w}''_{ij}$ is similar to $\mathbf{w}''$ as in Eqn. 3.53, respectively. Algorithm 3 summarizes the scheme of block-Toeplitz matrix-based backward propagation.

Notice that similar to the case of Toeplitz matrix, block-Toeplitz matrix also achieves simultaneous reduction in space and computational complexity. Specifically, the space complexity is reduced from $O(n^2)$ to $O(n^2/b)$ and computational complexity is reduced from $O(n^2)$ to $O(\frac{n^2}{b}\log b)$. Hence such reduction can be precisely controlled by adjusting the block size b.

Algorithm 2: Block-Toeplitz Matrix-based Forward Propagation

Input: $\mathbf{w}'_{11}, ..., \mathbf{w}'_{m/b,j}, \mathbf{x}, b$
Output: $\mathbf{a}$

Partition $\mathbf{x} \in \mathbb{R}^n$ into n/b vectors, $\mathbf{x}_1, \ldots, \mathbf{x}_{n/b}$;
for $i \leftarrow 1$ *until* m/b **do**
  $\mathbf{a}_i \leftarrow 0$;
  for $j \leftarrow$ *until* n/b **do**
    $\mathbf{a}_i \leftarrow \mathbf{a}_i + \text{IFFT}(\text{FFT}(\mathbf{w}'_{ij}) \circ \text{FFT}(\mathbf{x}_j))_{1:b}$;
  end
end
return $\mathbf{a}$;

Algorithm 3: Block-Toeplitz Matrix-based Backward Propagation

Input: $\frac{L}{\mathbf{a}'_1}, \ldots, \frac{L}{\mathbf{a}'_{m/b}}, \mathbf{x}''_1, \ldots, \mathbf{x}''_{n/b}, b$
Output: $\frac{L}{\mathbf{x}}, \frac{L}{\mathbf{w}''_{11}}, \ldots, \frac{L}{\mathbf{w}''_{m/b,n/b}}$

for $j \leftarrow 1$ *until* n/b **do**
  $\frac{L}{\mathbf{x}_j} \leftarrow 0$;
  for $i \leftarrow$ *until* m/b **do**
    $\frac{L}{\mathbf{w}'_{ij}} \leftarrow \text{IFFT}(\text{FFT}(\frac{L}{\mathbf{a}'_i}) \circ \text{FFT}(\mathbf{x}''_j))$;
    $\frac{L}{\mathbf{x}_j} \leftarrow \frac{L}{\mathbf{x}_j} + \text{IFFT}(\text{FFT}(\frac{L}{\mathbf{a}'_i}) \circ \text{FFT}(\mathbf{w}''_{ij}))_{1:b}$;
  end
end
return $\frac{L}{\mathbf{x}}, \frac{L}{\mathbf{w}''_{11}}, \ldots, \frac{L}{\mathbf{w}''_{m/b,n/b}}$;

3.5 Experiments

Dataset, Baseline & Experiment Environment. We evaluate our circulant structure-imposing approaches on two typical image classification datasets: CIFAR-10 [3] and ImageNet ILSVRC-2012 [2]. For each dataset, we take classical network models (ResNet [13] for CIFAR-10 and AlexNet [48] for ImageNet) as the baseline models. The compressed circulant CNN models are generated by replacing convolutional layers of the baseline models with circulant convolutional layers. All models in this paper are trained using NVIDIA GeForce GTX 1080 GPUs and Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz.

Selection of Training Strategy. As presented in Section imposing CircConv, the circulant CNN model can be trained either from scratch or re-trained from a pre-trained non-circulant model. In our experiments we evaluate these two different training strategies on different datasets. Experimental results show that with the same compression configuration setting, the compressed circulant CNN models generated by these two training strategies have very similar test accuracies. Therefore in this paper we only report the results using training-from-scratch strategy.

3.5.1 ResNet on CIFAR-10

In this experiment, ResNet-32 is selected as the baseline model due to its high accuracy and easiness of training. The training data is augmented by following the method in [49]: First pad each side of the image with four pixels and then apply 32×32 sized random crops with horizontal flipping. The compressed ResNet-32 models are trained using stochastic gradient descent (SGD) optimizer with learning rate 0.1, momentum 0.9, batch size 64 and weight decay 0.0001.

Model setting. ResNet-32 consists of 15 *convolutional blocks*, where each convolutional block contains two or three convolutional layers. Considering the number of possible compression configurations on different convolutional layers is very large, we choose to make the layers in the same block have the same compression ratio. In other words, a *block-wise* compression strategy is adopted. Notice that because the first few convolutional layers of

Table 3.2: Compression Configurations. For the convolutional block with compression ratio i, all the convolutional layers in that block has the same compression ratio i.

Partitioned Block ID	Model ID						
	1	2	3	4	5	6	7
1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1
3	1	1	2	2	2	4	4
4	1	1	2	2	2	4	4
5	1	1	2	2	2	4	4
6	1	1	1	1	1	1	1
7	1	1	2	2	4	4	8
8	1	1	2	2	4	4	8
9	1	1	2	2	4	4	8
10	1	1	2	2	4	4	8
11	1	1	1	1	1	1	1
12	1	2	2	4	4	4	16
13	1	2	2	4	4	4	16
14	2	2	2	4	4	4	16
15	2	2	2	4	4	4	16
baseline [13]:ResNet-32 without partitioning							

ResNet are very sensitive for compression [13], in this experiment we do not impose circulant structure to the convolutional layers in the 1st and 2nd blocks of ResNet-32. Besides, the 6th and 11th blocks are not compressed due to their small weight tensor.

Table 3.2 shows the detailed compression configurations for different convolutional blocks of ResNet-32. Here we explore 7 different compression configurations and then obtain 7 compressed models. For each compressed model, the compression ratios for its component convolutional blocks are listed in the row direction. Here each number i in a specific compression configuration scheme indicates the compression ratio as i for the convolutional layers in the corresponding convolutional block. When the block is associated with 1, that means the corresponding convolutional block is not compressed. Notice that due to the sensitivity of front blocks, for all the 7 models in Table 3.2 the compression ratios of the front blocks are typically less than those of the later blocks.

Trade-off between accuracy and model size. Figure 3.3 shows the test error for 7 compressed models. It can be seen that model 1 even achieves slightly better performance with a smaller model size than the baseline. Moreover, model 2, 3 and 4 achieve around 50% reduction in model size with negligible accuracy drop. With more aggressive compression

configurations are selected (such as model 5, 6 and 7), more reduction in model size can be further achieved with slight increase of test error.

Trade-off between accuracy and FLOPs. Our experiment also shows that the use of circulant convolutional layer helps reduce computational cost significantly. As shown in Figure 3.4, compressed model 1 and 2 achieve fewer FLOPs than baseline with the same or even less test error. For model 3, it can achieve 50% reduction in FLOPs with negligible test error increase. An interesting discovery is that though model 4, 5 and 6 have more aggressive compression configurations than model 3, their corresponding reduction in FLOPs are less than what model 3 achieves. This is because the convolutional layers in the model 3 are mainly compressed with the factor of 2, which corresponds to 2-point FFT computation that only needs real number operations.

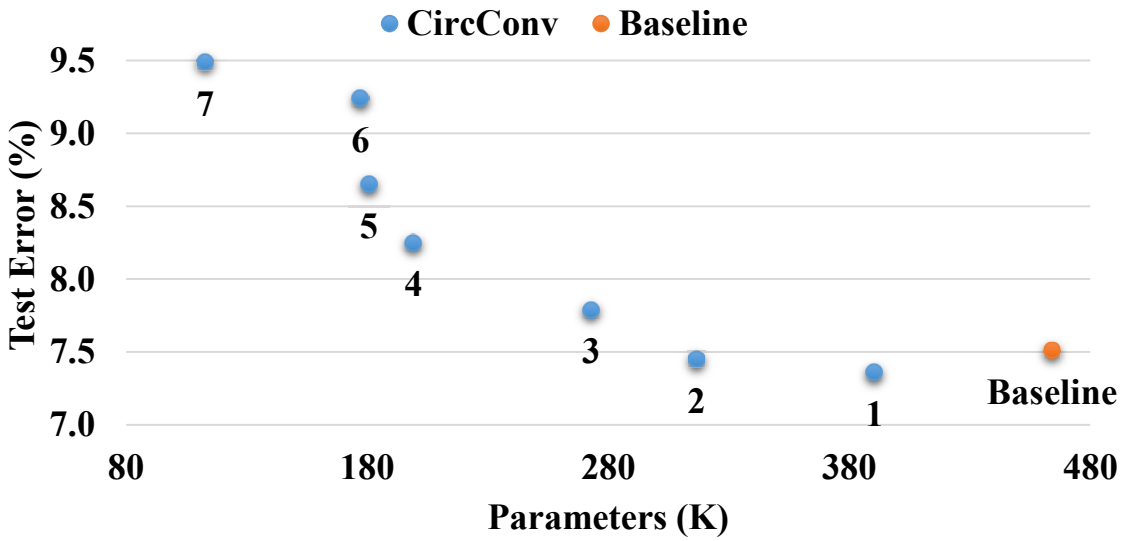


Figure 3.3: ResNet-32 Test Error and Model Size. Use of circulant convolutional layer can bring half of parameters reduction with negligible test error increase.

3.5.2 Wide ResNet on CIFAR-10

We also conduct the experiment on CIFAR-10 dataset using Wide ResNet [50], which has better performance than conventional ResNet in term of test accuracy. In this experiment, the compressed Wide ResNet models are trained using SGD with learning rate 0.01, momentum 0.9, batch size 64 and weight decay 0.0005.

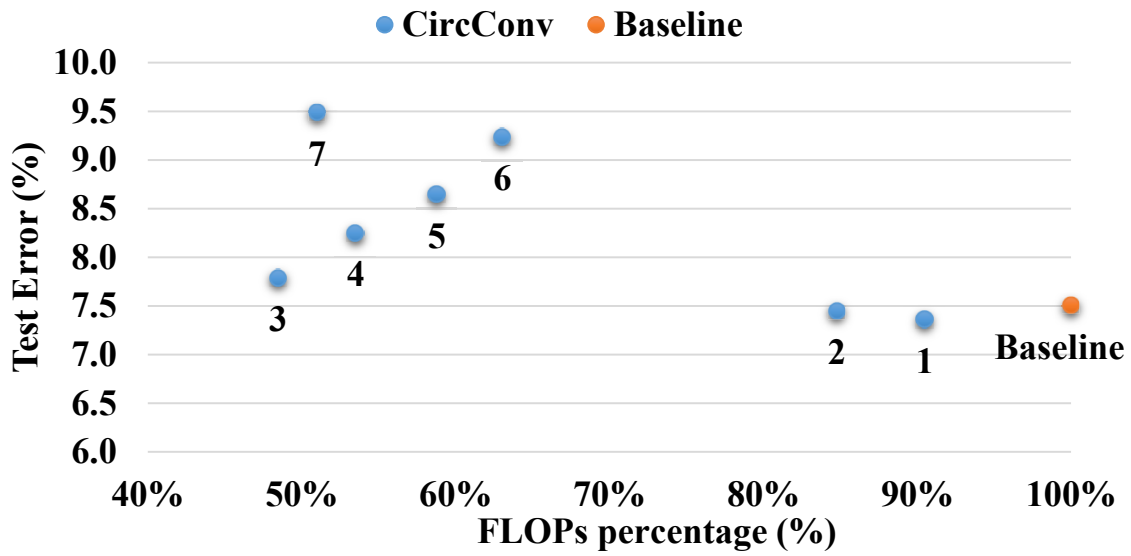


Figure 3.4: ResNet-32 Test Error and Model Size. Use of circulant convolutional layer can bring half of FLOPs reduction with negligible test error increase.

Model settings. To construct baseline Wide ResNet models, we take the same basic convolutional block structure in [50] and set different numbers of convolutional blocks and widening parameters for different models. To achieve better performance, we add two more blocks to the convolutional blocks that are wider than $16 \times k$, where k is the inherent widening parameter of each block. Different from the experiment in ResNet experiment section, this experiment on Wide ResNet adopts very aggressive compression strategy: For one convolutional layer, if the numbers of input channels (C_0) and output channels (C_2) are the same, then the compression ratio for that layer is $i = C_0 = C_2$; otherwise the convolutional layer is not compressed. We apply this compression strategy to five different Wide ResNet baselines and obtain five compressed Wide ResNet models. For each compressed model, it is labeled with two numbers ("*d*-*k*"), where d and k denote the number of convolutional layers ("depth") and widening parameter ("width"), respectively. These compressed models are compared with their corresponding baseline models as well as ResNet-110, which achieves the best performance on CIFAR-10 in [13].

Model size reduction. Figure 3.5 shows the number of parameters of Wide ResNet baselines and the corresponding compressed models after imposing circulant structure. It can be seen that the compressed models greatly reduce the model size. In particular,

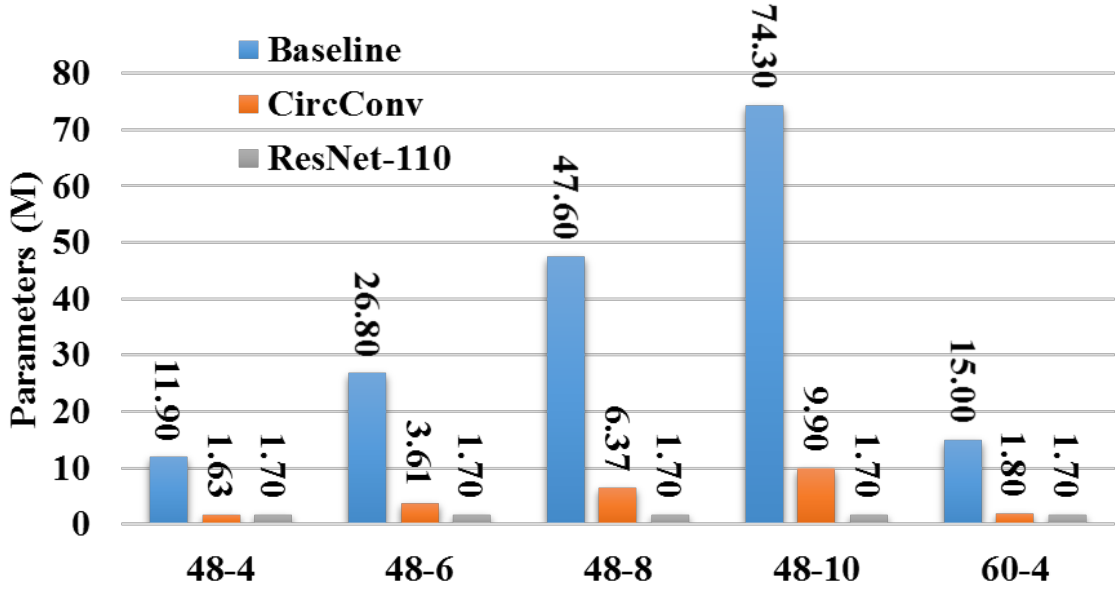


Figure 3.5: Wide ResNet Model Size Reduction. Compared with baseline models, compressed models achieve similar model size as ResNet-110. Compressed model named like "48-4" has 48 convolutional layers and widening parameter as 4.

model "60-4" can achieve 8.35 times reduction in the model size. Also, it can be seen that the numbers of parameters of Wide ResNet models are similar to the size of ResNet-110 after applying circulant convolutional layer. For instance, Model "48-4" has around 1.6M parameters which is less than 1.7M for ResNet-110.

Test error analysis. Figure 3.6 shows test errors of baseline Wide ResNet models and the corresponding compressed models using circulant convolutional layer. It can be seen that all compressed models have slightly test error increase less than 1%. In addition, compared with the state-of-the-art ResNet-110, all of the compressed models have around 1% test error decrease.

Comparison with ResNet-110. From Figure 3.6 we can see that all compressed Wide ResNet models have less test error than ResNet-110. Meanwhile, Figure 3.5 shows these compressed models have similar numbers of parameters as compared with ResNet-110, and model "48-4" has even fewer parameters. These results demonstrate that circulant structure-imposing approach can be useful in reducing model redundancy and holding less test error while maintaining similar model sizes.

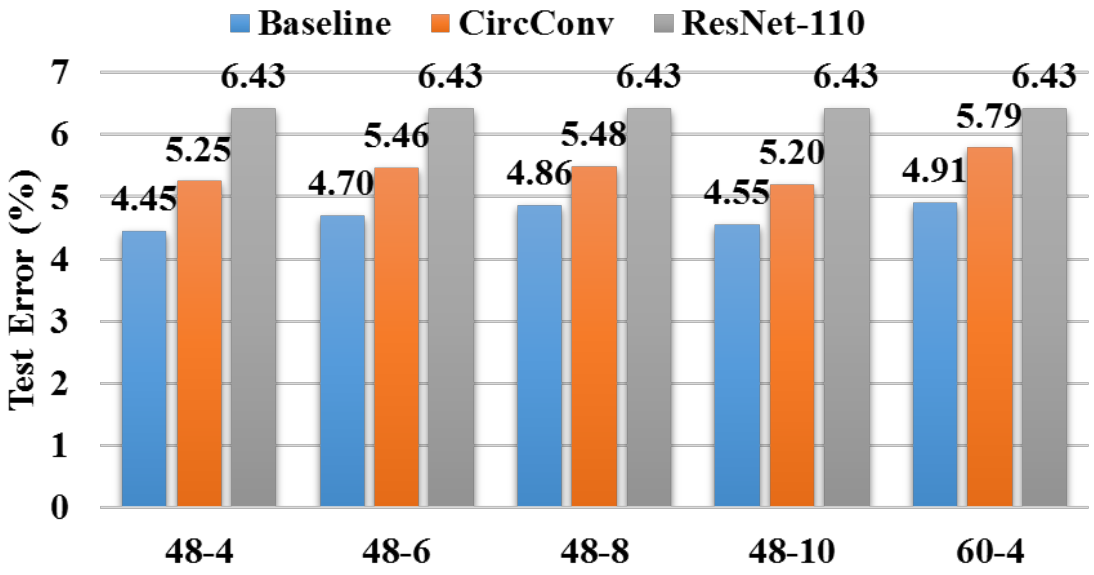


Figure 3.6: Wide ResNet Test Error. Baseline models are different original Wide ResNets and they are compared with the corresponding compressed models and ResNet-110.

FLOPs reduction. As shown in Figure 3.7, we measure the overall FLOPs reduction of Wide ResNet. It is found that the compressed Wide ResNet models can achieve significant reduction in FLOPs: all of them only require around 36% FLOPs as compared to the corresponding baseline models. In addition, the FLOPs reduction for the compressed blocks are very significant. From Figure 3.7 it can be seen that the FLOPs in the compressed blocks of all compressed models are only less than 6% of the corresponding uncompressed blocks in the original Wide ResNet baseline models.

3.5.3 AlexNet on ImageNet

To test the effectiveness of the proposed circulant-imposing approach on large-scale datasets, we evaluate the performance of circulant CNNs on ImageNet (ILSVRC2012). Here the baseline model is AlexNet [48]. All training images are randomly distorted as suggested in [51]. We train our AlexNet models using RMSprop [52] with learning rate 0.01, momentum 0.9, batch size 32 and decay 0.9.

Model settings. We explore three different compression configurations for the five convolutional layers in AlexNet. Table 3.3 listed the detailed compression configuration

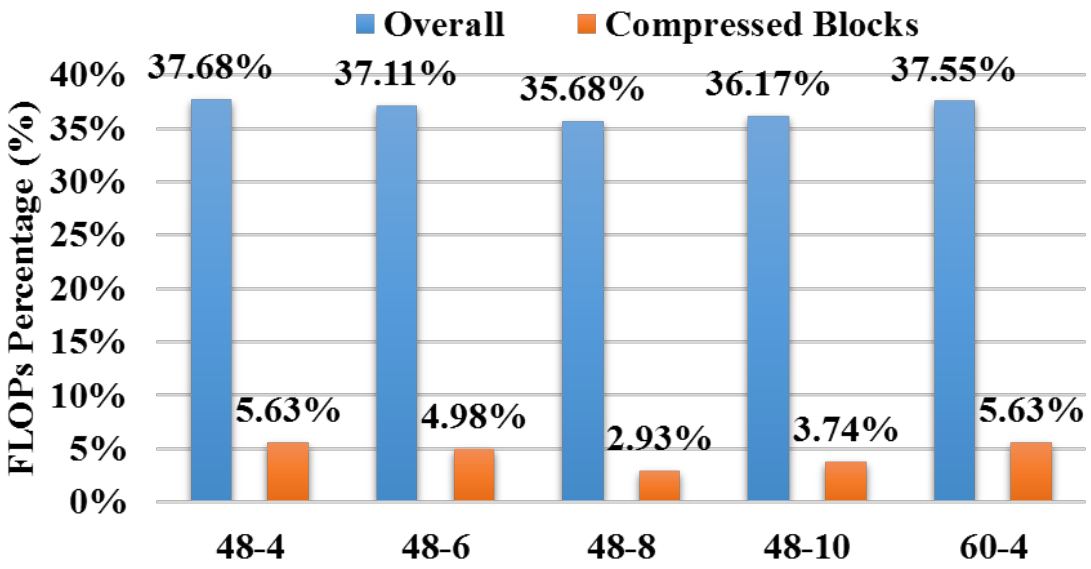


Figure 3.7: Wide ResNet FLOPs. The overall FLOPs measure the FLOPs percentage of compressed models over corresponding baselines. We also list FLOPs percentage of compressed convolutional blocks over original blocks.

schemes by using notation "*a-b-c-d-e*". For instance, "1-2-2-2-2" means the first convolutional layer is not compressed, and the rest four layers are compressed with the factor of 2. By using these configurations, three compressed AlexNet models are generated and compared with original AlexNet baseline model. Also, since SSL in [34] is the state-of-the-art work that explores the relationship between accuracy and compressed model size for AlexNet, we also compare our circulant convolutional layer-based compressed AlexNet models with three SSL regularization-based compressed AlexNet models in [34].

Test error analysis. Table 3.3 shows the test errors of compressed AlexNet models by using circulant structure-imposing and SSL approaches. It can be seen that both these two approaches can render the compressed models with the similar test errors to the original AlexNet model. Among them, the circulant model with "1-2-2-2-2" compression configuration achieves the least test error.

Model size reduction. Table 3.3 shows the percentage of number of parameters of each model over original AlexNet model. It can be seen that circulant convolution-based models have similar numbers of parameters to SSL-based models. Among them the circulant convolution-based model with "1-2-2-4-2" compression configuration has the least number

Table 3.3: Comparison among AlexNet models.

AlexNet Model	Compression Configuration	Test Error (%)	Parameters (%)	FLOPs(%)
Baseline	N/A	42.9	100	100
CircConv	1-2-2-2-2	**42.75**	50.36	**31.3**
CircConv	1-2-2-4-2	42.99	40.01	**31.3**
CircConv	1-2-4-2-2	43.13	45.19	**31.3**
[34]	N/A	42.75	51.20	39.0
[34]	N/A	43.00	44.40	43.0
[34]	N/A	43.25	42.30	45.0

of parameters.

FLOPs reduction. Table 3.3 shows the percentage of FLOPs of each model over the original AlexNet model. It can be found that all circulant convolution-based models require fewer FLOPs than the SSL regulation-based models. All the circulant convolution-based models have around 31% FLOPs of original uncompressed AlexNet baseline.

Overall comparison. As shown in Table 3.3 , circulant convolution-based models have similar accuracy to the state-of-the-art SSL models while maintaining similar number of parameters. Meanwhile, Table 3.3 shows that circulant convolution-based models requires less FLOPs than the SSL models when targeting to the similar accuracy. Therefore, imposing circulant structure to convolutional layer is a very promising accuracy-retained approach to reduce both the space and computational costs.

3.5.4 Speech Recognition

Table 3.4: Task performance with different compression ratio

Model	Block Size	WER	Overall Comp. Ratio
Uncompressed	-	18.08 [53]	1.00
Compressed-1	32	17.02	15.34
Compressed-2	64	18.12	28.76
Compressed-3	128	21.20	51.56

To evaluate the proposed Toeplitz structure, we perform experiment on long short-term memory (LSTM) for a speech recognition task. LSTM is a type of widely used DNN for

various sequence-involved tasks. In general, a LSTM takes a sequence of input $\mathbf{x}_1, \ldots, \mathbf{x}_T$ and generates a sequence of output $\mathbf{y}_1, \ldots, \mathbf{y}_T$, where T is number of time steps.

It is seen that LSTM consists of multiple weight matrices. Therefore, we impose block-Toeplitz structure on all weight matrices to compress model size. Specifically, the compression is performed on the model structure in the LSTM layers of model in [53]. The dataset is the AN4 audio data for speech recognition, where training utterances are 948, testing utterances are 130, and there are in total 29 unique spoken characters. During train process, we train 150 epochs using batch size 32, learning rate 0.0003 with annealing rate 1.01. The task performance is measured in word error rate (WER).

Table 3.4 summarizes the test results with different compression ratio. It is seen that the proposed approach leads to high compression ratio with negligible performance loss. Notice that here the block size does not equal to overall compression ratio. This is because after high compression on weight matrix the uncompressed bias vectors dominates the model size.

3.6 Conclusion

We have proposed to impose the low displacement rank structure to convolutional neural network. This structure-imposing approach leads to significant reduction in model size, FLOPs with negligible accuracy drop. Complexity analysis and experiments on different datasets and different network models demonstrate the effectiveness of the proposed approach.

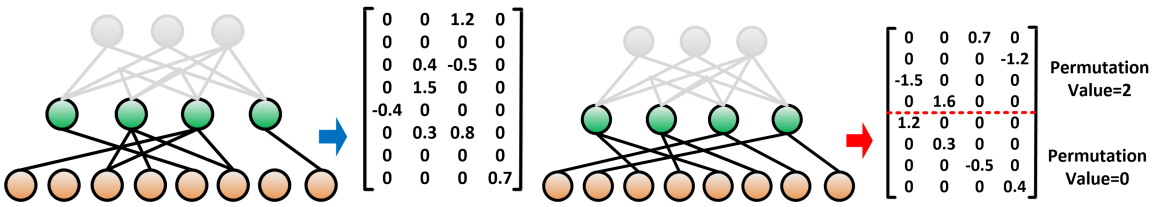
Chapter 4

Permuted Diagonal Matrix

To improve energy efficiency, efficient model compression has emerged as a very active topic in AI research community. Among different types of DNN compression techniques [54, 55, 56, 34], two approaches show the promising results. First, *network sparsification* is believed to be the most popular and state-of-the-art strategy because of its good balance between compression ratio and test accuracy. To date, many methods [54][55][34] have been proposed to perform efficient sparsification on different DNN models. Echoing the importance and popularity of this approach, several sparse model-oriented hardware architectures [57, 58, 59, 60] have been proposed .

However, the current network sparsification methods suffer from the inherent drawbacks of *irregularity*, *heuristic nature* and *indexing overhead*. Consequently, despite the encouraging compression ratio, the *unstructured* sparse DNN models cannot achieve optimal performance on the current computing platforms, especially DNN hardware accelerators. Essentially, the inefficient execution and hardware implementation are caused by the *non-hardware-friendly* sparse models.

In this chapter, we propose PermDNN, a novel approach that generates and executes hardware-friendly structured sparse DNN models using permuted diagonal matrices (PDM). As illustrated in Fig. 4.1(b), permuted diagonal matrix is a type of structured sparse matrix that places all the non-zero entries in the diagonal or permuted diagonal. When the weight matrices of DNNs can be represented in the format of multiple permuted diagonal matrices (so-called *block-permuted diagonal matrices*), their inherent strong structured sparsity leads to great benefits for practical deployment. Specifically, it eliminates indexing overhead, brings non-heuristic compression effects, and enables re-training-free model generation.



(a) Unstructured sparse weight matrix. (b) Block-permuted diagonal weight matrix.

Figure 4.1: Weight representation by using (a) conventional unstructured sparse matrix. (b) block-permuted diagonal matrix.

Based on PDM, we develop an end-to-end training scheme that can generate a high-accuracy permuted diagonal matrix-based DNN models from scratch. We also develop the corresponding low-complexity inference scheme that executes efficiently on the trained structured sparse DNN models. Experiment results on different datasets for different application tasks show that, enforced with the strong structure, the permuted diagonal matrix-based DNNs achieve high sparsity ratios with no or negligible accuracy loss.

4.1 Permuted Diagonal Fully Connected Layer

Let $\mathbf{W} \in \mathbb{R}^{m \times n}$ be the weight matrix which can be divided into small blocks in shape $k \times k$. In total, there will be $m/k \times n/k$ blocks. Let $\mathbf{Z}_s$ be a matrix generated by permuting a diagonal matrix and the permutation is shifting from the main diagonal to the s-th diagonal as follows:

$$\mathbf{Z}_s = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 & 1 \\ 1 & 0 & 0 & \dots & 0 & 0 \\ 0 & \ddots & & \ddots & \vdots & \vdots \\ \vdots & & \ddots & \ddots & 0 & 0 \\ 0 & \dots & & 0 & 1 & 0 \end{bmatrix}^{S}. \tag{4.1}$$

Each block $\mathbf{W}_{i,j}$ for $i = 1, \dots, \frac{m}{k}$ and $j = 1, \dots, \frac{n}{k}$ can then be formulated as following:

$$\mathbf{W}_{i,j} = \mathbf{Z}_{s_{i,j}} \times \text{diag}(\mathbf{w}_{i,j}), \tag{4.2}$$

where $s_{i,j}$ is the shifting parameters for the block and the $diag(\cdot)$ function outputs a diagonal matrix for given vector. $\mathbf{w}_{i,j}$ is a vector of k weight parameters inside the sub-matrix.

Therefore, the density of the block permuted diagonal matrix $\mathbf{W}$ is $\frac{1}{k}$. The sparsity $(1-\frac{1}{k})$ is controlled via the setting of value k. With larger k, the matrix is more sparse.

4.1.1 Forward Propagation

DNN computes output from the first input layer to the last output layer and this process is called forward propagation. For a layer defined with permuted diagonal matrix, the computation is mainly matrix multiplication. Let input $\mathbf{x} \in \mathbb{R}^n$ be equally divided into n/k sub-vectors. The multiplication can be written as following:

$$\begin{aligned}\mathbf{a} = \mathbf{W}\mathbf{x} &= \Big[\sum_{j=1}^{n/k} \mathbf{W}_{i,j}\mathbf{x}_j\Big]_{i=1}^{m/k} \\ &= \Big[\sum_{j=1}^{n/k} \mathbf{Z}_{s_{i,j}} \times \mathrm{diag}(\mathbf{w}_{i,j}) \times \mathbf{x}_j\Big]_{i=1}^{m/k} \\ &= \Big[\sum_{j=1}^{n/k} \mathbf{Z}_{s_{i,j}} \times \mathrm{diag}(\mathbf{w}_{i,j} \circ \mathbf{x}_j)\Big]_{i=1}^{m/k},\end{aligned} \tag{4.3}$$

where $\circ$ is the element-wise product. Note that $\mathbf{Z}_{s_{i,j}}$ is essentially shifting a vector which could be completed in linear time complexity. Overall, the forward propagation complexity can be easily implemented in the complexity of $O(mn/k)$.

4.1.2 Backward Propagation

Recall that DNN optimization algorithms mainly utilize the first order derivatives. The chain rule will expand the computation across layers, i.e., from the last layer to the very first layer. This process is called backward propagation. There are two gradients to calculate, $\frac{\partial L}{\partial \mathbf{w}_{i,j}}$ and $\frac{\partial L}{\partial \mathbf{x}}$. The $\frac{\partial L}{\partial \mathbf{w}_{i,j}}$ is used for updating weight parameters. The $\frac{\partial L}{\partial \mathbf{x}}$ is for computing gradients at next layer. Let $\mathbf{a}$ be equally divided into m/k parts and $\mathbf{a}_i$ be the i-th part. We first set up $\mathbf{b}_{i,j} \in \mathbb{R}^{k\times k}$ as following:

$$\begin{aligned}\mathbf{b}_{i,j} &= \mathrm{diag}(\mathbf{w}_{i,j} \circ \mathbf{x}_j) \\ \frac{\partial L}{\partial \mathbf{b}_{i,j}} &= \mathrm{diag}(\mathbf{Z}_{k-s_{i,j}} \times \frac{\partial L}{\partial \mathbf{a}_i}).\end{aligned} \tag{4.4}$$

The $\frac{\partial L}{\partial \mathbf{w}_{i,j}}$ and $\frac{\partial L}{\partial \mathbf{x}}$ can be computed by:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{w}_{i,j}} &= \text{diag}(\frac{\partial L}{\partial \mathbf{b}_{i,j}}) \circ \mathbf{x}_j \\ &= (\mathbf{Z}_{k-s_{i,j}} \times \frac{\partial L}{\partial \mathbf{a}_i}) \circ \mathbf{x}_j, \end{aligned} \tag{4.5}$$

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{x}_j} &= \text{diag}(\frac{\partial L}{\partial \mathbf{b}_{i,j}}) \circ \mathbf{w}_{i,j} \\ &= (\mathbf{Z}_{k-s_{i,j}} \times \frac{\partial L}{\partial \mathbf{a}_i}) \circ \mathbf{w}_{i,j}, \end{aligned} \tag{4.6}$$

where $diag(\cdot)$ is taking the diagonal line from the matrix. As a result, the computation complexity for backward propagation is also $O(mn/k)$ which is similar to the forward propagation and the matrix density.

4.1.3 Approximation

As mentioned in [61], it is possible to find the closest permuted diagonal matrix when given a matrix from pre-trained DNN model. Given shifting parameter s for $\mathbf{W} \in \mathbb{R}^{k \times k}$, the closest permuted diagonal matrix defined on $\mathbf{w} \in \mathbb{R}^k$ is the corresponding s-th diagonal:

$$\mathbf{w} = \text{diag}(\mathbf{Z}_{k-s} \times \mathbf{W}), \tag{4.7}$$

where $diag(\cdot)$ takes the diagonal line from the matrix. This method could be used to initialize training from a pre-trained DNN model. It may accelerate the convergence of training to some degree.

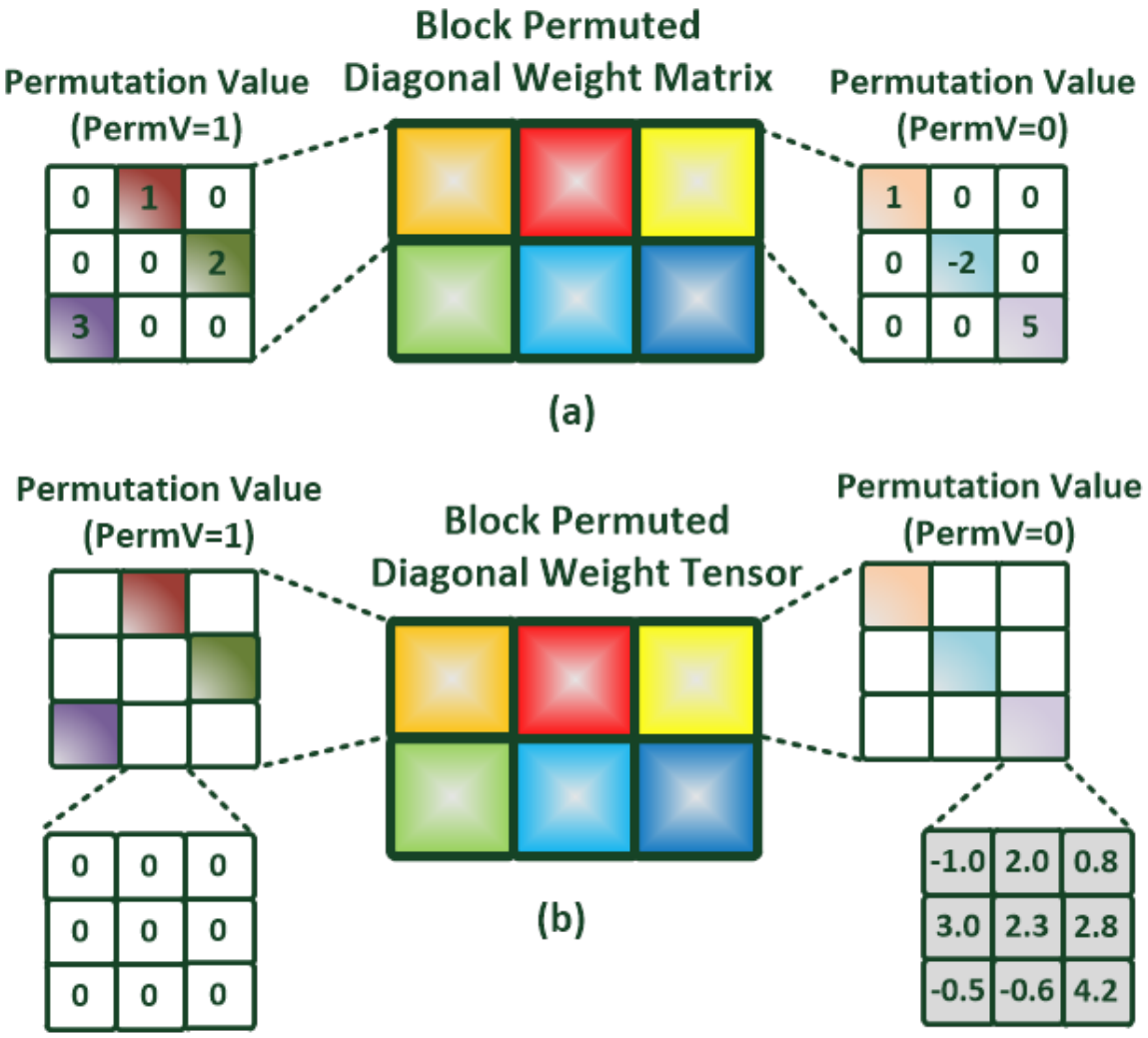


Figure 4.2: Example of imposing block-diagonal permuted structure on (a) weight matrix of FC layer and (b) weight tensor of CONV layer. Here the block size p, as the compression ratio, is 3. The entire weight matrix or tensor contains blocks of component weight matrices or tensors, and each component permuted diagonal matrix or tensor is affiliated with a permutation value (PermV). PermV is selected from 0,1,...,p-1. The non-zero weight values or weight filter kernels can only be placed in the main diagonal or permuted diagonal positions in the component weight matrices or tensors.

4.2 Permuted Diagonal Convolution Layer

In general, CONV layer is the most important component of a CNN model since it consumes the largest portion of the overall computation. Mathematically, when the inference is performed in a batch way, the computation of a CONV layer is essentially the 2D convolution between a 4D input tensor (a batch of 3D inputs) and 4D weight tensor (a group of 3D filters), and the output of CONV layer is also a 4D tensor. The detailed computation procedure is described as follows:

$$\mathcal{Y}(n,m,e,f) = \sum_{c=0}^{C-1}\sum_{r=0}^{R-1}\sum_{s=0}^{S-1} \mathcal{W}(m,c,r,s) \times \mathcal{X}(n,c,Ue+r,Uf+s), \tag{4.8}$$

where $\mathcal{X} \in \mathbb{R}^{N\times C\times H\times W}$, $\mathcal{Y} \in \mathbb{R}^{N\times M\times E\times F}$, $\mathcal{W} \in \mathbb{R}^{M\times C\times R\times S}$ are input tensor, output tensor, and weight tensor, respectively. And the meaning of parameters M, C, H/W, E/F, R/S, N, U are described in Table 4.1.

Table 4.1: The CNN parameters.

Parameter	Description
M	Number of 3D filters
C	Number of channels
H/W	Height/width of input
E/F	Height/width of output
R/S	Filter kernel height/width
N	Batch size of 3D input activations
U	Stride

4.2.1 Forward Propagation

As illustrated in Fig. 4.2, the key idea of designing permuted diagonal CONV layer is to impose such structure along the two dimensions that are defined by number of 3D filters and number of channels. Based on this mechanism, the non-zero kernels can only be placed in the main diagonal or permuted diagonal positions. Therefore, we only need to store non-zero kernels $\mathcal{W}' \in \mathbb{R}^{\frac{M\times C}{p}\times R\times S}$, and the mapping between original $\mathcal{W}$ and non-zero kernels $\mathcal{W}'$ is defined as following:

$$\mathcal{W}(m,c,r,s) = \begin{cases} \mathcal{W}'(k_l \times p + i, r, s) & (i + k_l) \equiv c \mod p \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

where p is the block size, $i \equiv m \mod p$, $l = \lfloor \frac{m}{p} \rfloor \times \frac{N}{p} + \lfloor \frac{c}{p} \rfloor$. The k_l is the permuted offset for diagonals, also denoted as PermV. Based on this mapping, the convolution is only performed for non-zero kernels. Overall, the forward propagation can be summarized as:

$$\begin{aligned} \mathcal{Y}(n,m,e,f) = \sum_{g=0}^{\frac{C}{p}-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathcal{X}(n, gp + (i + k_l \mod p), \\ Ue + r, Uf + s) \times \mathcal{W}'(k_l \times p + i, r, s). \end{aligned} \tag{4.10}$$

4.2.2 Backward Propagation

Besides forward propagation, the corresponding backward propagation is also developed to ensure the trained CONV layer exhibits permuted diagonal structure as follows:

$$\begin{aligned} \frac{\partial J}{\mathcal{W}(m,c,r,s)} = \sum_{n=0}^{N-1} \sum_{e=0}^{E-1} \sum_{f=0}^{F-1} X(n,c,Ue+r,Uf+s) \\ \times \frac{\partial J}{\partial Y(n,m,e,f)}, \forall\ \mathcal{W}(m,c,r,s) \neq 0 \end{aligned} \tag{4.11}$$

$$\begin{aligned} \frac{\partial J}{\partial \mathcal{X}(n,c,x,y)} = \sum_{m=1}^{M-1} \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} \mathcal{W}(n,m,r,s) \\ \times \frac{\partial J}{\partial \mathcal{Y}(n,m,,(x-r)/U,(y-s)/U)}. \end{aligned} \tag{4.12}$$

Notice that here Eqn. 4.12 is used to aid the gradient computation described in Eqn. 4.11 since $\mathcal{X}(n,c,x,y)$ in the current layer is the output of previous layer as $\mathcal{Y}(n,m,e,f)$.

4.2.3 Outline of Theoretical Proof on Universal Approximation

In CirCNN the *universal approximation property* of block-circulant matrix-based neural network was given to theoretically prove the effectiveness of using circulant matrices. In this work we discover that the PermDNN also exhibits the universal approximation property, thereby making the rigorous foundation for our proposed permuted diagonal structure-imposing method. The details of the proof will be provided in an individual technical

report and this subsection gives a brief outline of the proof as follows. First, we prove that the "connectedness" of PermDNN, – that means, thanks to the unique permuted diagonal structure of each block, when the k_l is not identical for all permuted diagonal matrices, the sparse connections between adjacent block-permuted diagonal layers do not block away information from any neuron in the previous layer. Based on this interesting property, we further prove that the function space achieved by the block-permuted diagonal networks is dense. Finally, with the Hahn-Banach Theorem we prove that there always exists a block-permuted diagonal neural network that can closely approximate any target continuous function defined on a compact region with any small approximation error, thereby showing the universal approximation property of block-permuted diagonal networks. Besides, we also derive that the error bound of this approximation error is in the order of $O(1/n)$, where n is the number of model parameters. Consequently, the existence of universal approximation property of PermDNN theoretically guarantees its effectiveness on different DNN types and applications.

4.2.4 Applicability on the Pre-trained Model

Besides training from scratch, the permuted diagonal matrix-based network model can also be obtained from a pre-trained dense model. Fig. 4.3 illustrates the corresponding procedure, which consists of two steps: permuted diagonal approximation and re-training/fine-tuning. First, the original dense weight matrices/tensors need to be converted to permuted diagonal matrices/tensors via *permuted diagonal approximation*. The mechanism of permuted diagonal approximation is to convert a non-permuted diagonal matrix/tensor to a permuted diagonal format by only keeping the entries in the desired permuted diagonal positions. **Mathematically, such approximation is the optimal approximation in term of l_2 norm measurement on the approximation error**. After that, the converted model already exhibits permuted diagonal structure and then can be further re-trained/fine-tuned by using Eqns. (4.11)-(4.12) to finally obtain a high-accuracy permuted diagonal network model. Such two-step generating approach is applied to all types of pre-trained models and can lead to high accuracy. For instance, for pre-trained dense LeNet-5 model on MNIST dataset, with $p = 4$ for CONV layer and $p = 100$ for FC layer, the finally

converted permuted-diagonal network after re-training achieves 99.06% test accuracy and overall 40× compression ratio without using quantization.

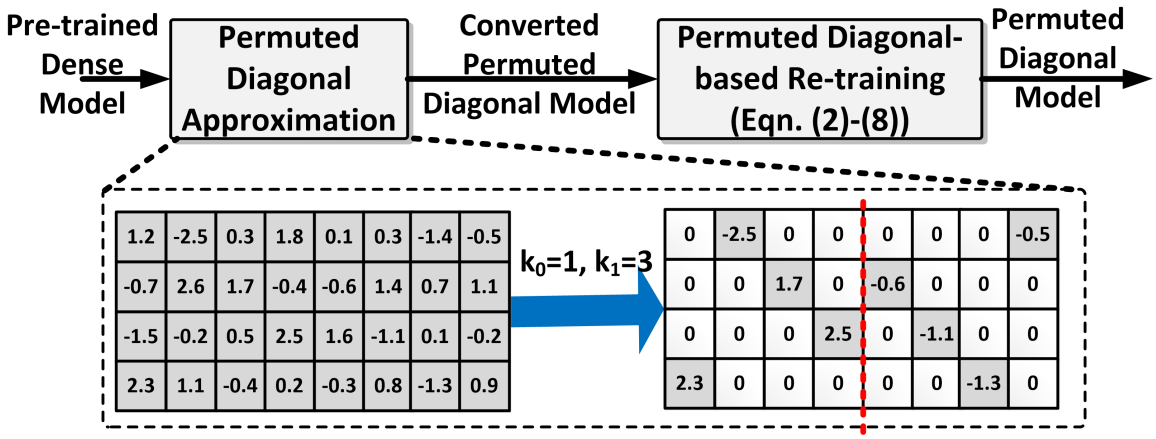


Figure 4.3: Train a PermDNN from a pre-trained dense model.

4.2.5 PermDNN vs Unstructured Sparse DNN

Compared to the existing network sparsification approaches, the proposed PermDNN enjoys several attractive advantages:

First, PermDNN *is a hardware-friendly model.* As illustrated in Fig. 4.4, due to the inherent regular structure of permuted diagonal matrix, the position of each non-zero entry can now be calculated using very simple modulo operation, thereby completely eliminating the needs of storing the indices of entries. From the perspective of hardware design, this elimination means that the permuted diagonal matrix-based PermDNN completely avoids the huge space/computation overhead incurred by the complicated weight indexing/addressing in the state-of-the-art sparse DNN accelerator (e.g. EIE), and hence achieves significant reduction in the space requirement and computational cost for DNN implementations.

Second, PermDNN *provides controllable and adjustable compression and acceleration schemes.* The reductions in the model sizes and number of arithmetic operations are no longer heuristic based and unpredictable. Instead it can now be precisely and deterministically controlled by adjusting the value of p. This further provides great benefits to explore the design space exploring the tradeoff between hardware performance and test accuracy.

Finally, PermDNN *enables direct end-to-end training while preserving high accuracy.* Since the structure of sparse weight matrices of PermDNN can now be pre-determined by the model designers at the initialization stage of training, with our training algorithm that preserves this fixed structure, the entire structured sparse network can be trained from scratch, completely avoiding the increasing complexity incurred by the extra iterative pruning and/or re-training process in the conventional unstructured sparse DNN training schemes.

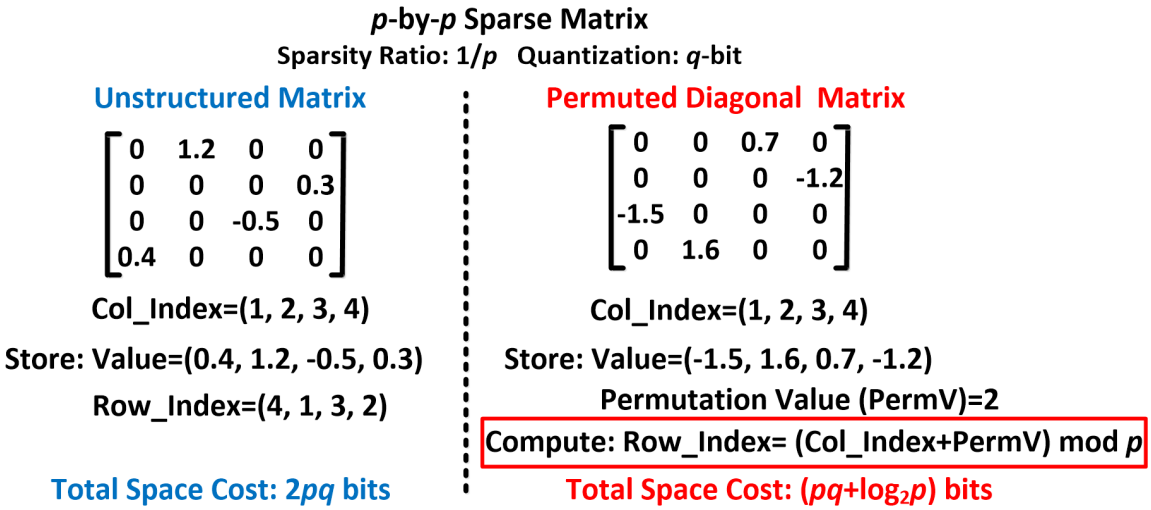


Figure 4.4: Storage requirement comparison.

4.3 Experiments

Table 4.2: AlexNet on ImageNet [2]. PD: Permuted Diagonal.

AlexNet	Block size (p) for PD weight matrix of FC6-FC7-FC8	Top-5 Acc.	Compression for overall FC layers
Original 32-bit float	1-1-1	80.20%	234.5MB(1×)
32-bit float with PD	10-10-4	80.00%	25.9MB(9.0×)
16-bit fixed with PD	10-10-4	79.90%	12.9MB(18.1×)

By leveraging the forward and backward propagation schemes, the PermDNN models can be trained from scratch and tested. Table 4.2 - Table 4.5 show the task performance and compression ratio of different PermDNN models on different types of datasets. Notice that for one FC/CONV layer with block size p for its permuted diagonal (PD) weight matrix/tensor, the compression ratio for that layer is p. The details of the experimental

Table 4.3: Stanford NMT (32-FC layer LSTMs) on IWSLT15 for English-Vietnamese Translation.

Stanford NMT (32-FC layer LSTMs)[1]	Block size (p) for PD weight matrix of ALL FC Layers	BLEU Points	Compression for overall FC layers
Original 32-bit float	1	23.3	419.4MB(1×)
32-bit float with PD	8	23.3	52.4MB(8×)
16-bit fixed with PD	8	23.2	26.2MB(16×)

Table 4.4: ResNet-20 on CIFAR-10[3].

ResNet-20	Block size (p) for PD weight tensor of CONV layers	Acc.	Compression for overall CONV Layers
Original 32-bit float	1	91.25%	1.09MB(1×)
32-bit float with PD	2 for most layers	90.85%	0.70MB(1.55×)
16-bit fixed with PD	2 for most layers	90.6%	0.35MB(3.10×)

setting are described as follows:

- **AlexNet [48]:** The block sizes (p) for the permuted diagonal weight matrices of three FC layers (FC6, FC7 and FC8) are set as different values (10, 10 and 4) for different FC layers.

- **Stanford Neural Machine Translation (NMT) [62]:** This is a stacked LSTM model containing 4 LSTMs with 8 FC weight matrices for each LSTM. In the experiment the value of p for all the FC layers is set as 8.

- **ResNet-20 [13]:** In this experiment for the group of CONV layers without 1x1 filter kernel, the value of p is set as 2. For the group of CONV layers with 1x1 filter kernel, p is set as 1.

- **Wide ResNet-48 [50]:** The widening parameter of this model is 8. For the group of CONV layers without 1x1 filter kernel, the value of p is set as 4. For the group of CONV layers with 1x1 filter kernel, p is set as 1.

- **Selection of Permutation value (k_l):** k_l can be selected via either natural indexing or random indexing. Our simulation results show no difference between task performance for these two setting methods. Table 4.2 - Table 4.5 are based on natural

Table 4.5: Wide ResNet-48 on CIFAR-10.

Wide ResNet-48	Block size (p) for PD weight tensor of CONV layers	Acc.	Compression for overall CONV layers
Original 32-bit float	1	95.14%	190.2MB(1×)
32-bit float with PD	4 for most layers	94.92%	61.9MB(3.07×)
16-bit fixed with PD	4 for most layers	94.76%	30.9MB(6.14×)

indexing. For instance, for a 4-by-16 block-permuted diagonal weight matrix with $p = 4$, $k_0 \sim k_3$ is set as $0 \sim 3$.

Table 4.2 - Table 4.5 show that imposing permuted diagonal structure to DNN models enables significant reduction in the weight storage requirement for FC or CONV layers. Meanwhile, the corresponding task performance, in terms of test accuracy (for computer vision) or BLEU scores [63] (for language translation), are still retained as the same or only exhibits negligible degradation. In short, PermDNN models can achieve high compression ratios in network size and strong spatial network structure, and simultaneously, preserve high task performance.

Chapter 5

Isotropic Iterative Quantization

Words are basic units in many natural language processing (NLP) applications, e.g., translation [64] and text classification [65]. Understanding words is crucial but can be very challenging. One difficulty lies in the large vocabulary commonly seen in applications. Moreover, their semantic permutations can be numerous, constituting rich expressions at the sentence and paragraph levels.

In statistical language models, word distributions are learned for unigrams, bigrams, and generally n-grams. A unigram distribution presents the probability for each word. The histogram is already sufficiently complex given a large vocabulary. Then, the complexity of bigram distributions is quadratic in the vocabulary size and that of n-gram ones is exponential. The combinatorial nature motivates researchers to develop alternative representations which otherwise explode.

Instead of word distributions, continuous representations with floating-point vectors are much more convenient to handle: they are differentiable, and their differences can be used to draw semantic analogy. A variety of algorithms were proposed over the years for learning these word vectors. Two representative ones are Word2Vec [66] and GloVe [67]. Word2Vec is a classical algorithm based on either skip grams or a bag of words, both of which are unsupervised and can directly learn word embeddings from a given corpus. GloVe is another embedding learning algorithm, which combines the advantage of a global factorization of the word co-occurrence matrix, as well as that of the local context. Both approaches are effective in many NLP applications, including word analogy and name entity recognition.

Neural networks with word embeddings are frequently used in solving NLP problems, such as sentiment analysis [68] and name entity recognition [69]. An advantage of word embeddings is that interactions between words may be modeled by using neural network

layers (e.g., attention architectures). Despite the success of these word embeddings, they often constitute a substantial portion of the overall model. For example, the pre-trained Word2Vec [70] contains 3M word vectors and the storage is approximately 3GB. This cost becomes a bottleneck in deployment on resource-constrained platforms.

Thus, much work studies the compression of word embeddings. [71] propose to represent word vectors by using multiple codebooks trained with Gumbel-softmax. [72] learn binary document emebddings via a bag-of-word-like process. The learned vectors are demonstrated to be effective for document retrieval. In information retrieval, iterative quantization (ITQ) [73] transforms vectors into binary ones, which are found to be successful in image retrieval. The method maximizes the bit variance meanwhile minimizing the quantization loss. It is theoretically sound and also computationally efficient. However, [72] find that directly applying ITQ in NLP tasks may not be effective. In [74], authors propose an alternate approach that improves the quality of word embeddings without incurring extra training. The main idea lies in the concept of isotropy used to explain the success of pointwise mutual information (PMI) based embeddings. The authors demonstrate that the isotropy could be improved through projecting embedding vectors toward weak directions.

Therefore, in this work we propose *isotropic iterative quantization* (IIQ), which leverages iterative quantization meanwhile satisfying the isotropic property. The main idea is to optimize a new objective function regarding the isotropy of word embeddings, rather than maximizing the bit variance. Maximizing the bit variance and maximizing isotropy are two opposite ideas, because the former performs projection toward large eigenvalues (dominant directions) while the latter projects toward the smallest ones (weak directions). Given prior success [74], it is argued that maximizing isotropy is more beneficial in NLP applications.

In information retrieval (where the proposed method is inspired), locality-sensitive hashing (LSH) is well studied and explored. The aim of LSH is to preserve the similarity between inputs after hashing. This aim is well aligned with that of embedding compression. For example, word similarity can be measured by the cosine distance of their embeddings. If LSH is applied, the hashed emebddings should maintain a similar distance as the original cosine distance but have much lower complexity in the meantime.

A well-known LSH method in image retrieval is ITQ [73]. However, its application in

NLP tasks such as document retrieval is not as successful [72]. Rather, the authors propose to learn binary paragraph embeddings via a bag-of-words-like model, which essentially computes a binary hash function for the real-valued embedding vectors.

On the other hand, [71] propose a compact structure for embeddings by using the gumble softmax. In this approach, each word vector is represented as the summation of a set of real-valued embeddings. This idea amounts to learning a low-rank representation of the embedding matrix.

Pre-trained embeddings may be directly used in deep neural networks (DNN) or serve as initialization [75]. There exist several compression techniques for DNNs, including pruning [15] and low-rank compression [76]. Most of these techniques requires retraining for specific tasks, thus challenges exist when applying them to unsupervised word embeddings (e.g., GloVe).

[77] successfully apply DNN compression techniques to unsupervised embeddings. The authors use pruning to sparsify embedding vectors, which however requires retraining after each pruning iteration. Although retraining is common when compressing DNNs, it often takes a long time to recover the model performance. Similarly, [78] uses low rank approximation to compress word embeddings, but they also face the same problem to fine-tune a supervised model.

5.1 Iterative Quantization and Embedding Isotropy

In this section, we briefly revisit the iterative quantization method by breaking it down into two steps. The first step is to maximize bit variance when transforming given vectors into binary representation. The second step is about minimizing the quantization loss while maintaining the maximum bit variance.

5.1.1 Maximize Bit Variance.

Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ be the embedding dictionary, where each row $\mathbf{x}_i^T \in \mathbb{R}^d$ denotes the embedding vector for the i-th word in the dictionary. Assuming that vectors are zero centered

($\sum_{i=1}^{n} \mathbf{x}_i = \mathbf{0}$), ITQ encodes vectors with a binary representation $\{-1, +1\}$ through maximizing the bit variance, which is achieved by solving the following optimization problem:

$$\begin{aligned} \max_{\mathbf{W}} \ & F(\mathbf{W}) = \frac{1}{n}\text{tr}(\mathbf{W}^T\mathbf{X}^T\mathbf{X}\mathbf{W}), \\ & \text{s.t. } \mathbf{W}^T\mathbf{W} = \mathbf{I} \text{ and } \mathbf{B} = \text{sgn}(\mathbf{X}\mathbf{W}), \end{aligned} \tag{5.1}$$

where $\mathbf{W} \in \mathbb{R}^{d\times c}$ and $c \leq d$ is the dimension of the encoded vectors. Here, $\mathbf{B}$ is the final binary representation of $\mathbf{X}$ and $\text{tr}(\cdot)$ and $\text{sgn}(\cdot)$ are the trace and the sign function, respectively. The problem is the same as that of Principal Component Analysis (PCA) and could be solved by selecting the top c right singular vectors of $\mathbf{X}$ as $\mathbf{W}$.

5.1.2 Minimize Quantization Loss.

Given a solution $\mathbf{W}$ to Equation (5.1), $\mathbf{U} = \mathbf{W}\mathbf{R}$ is also a solution for any orthogonal matrix $\mathbf{R} \in \mathbb{R}^{c\times c}$. Thus, we could minimize the quantization loss via adjusting the matrix $\mathbf{R}$ while maintaining the solution to (5.1). The quantization loss is defined as the difference between the vectors before and after the quantization:

$$Q(\mathbf{B}, \mathbf{R}) = ||\mathbf{B} - \mathbf{X}\mathbf{W}\mathbf{R}||_F^2, \tag{5.2}$$

where $||\cdot||_F$ is the Frobenius norm. Note that $\mathbf{B}$ must be binary. The proposed solution in ITQ is an iterative procedure that updates $\mathbf{B}$ and $\mathbf{R}$ in an alternating fashion until convergence. In practice, ITQ turns out able to achieve good performance with early stopping [73].

5.1.3 Isotropy of Word Embedding

In [79], isotropy is used to explain the success of PMI based word embedding algorithm, for example GloVe embedding. However, [74] find that existing word embeddings are not nearly isotropic but could be improved. The proposed solution is to project word embeddings toward the weak directions rather than the dominant directions, which seems counter-intuitive but in practice works well. The isotropy of word embedding $\mathbf{X}$ is defined as:

$$I(\mathbf{X}) = \frac{\min_{||\mathbf{e}||=1} Z(\mathbf{e})}{\max_{||\mathbf{e}||=1} Z(\mathbf{e})}, \tag{5.3}$$

where $Z(\cdot)$ is the partition function

$$Z(\mathbf{e}) = \sum_{\mathbf{x}_i \in \mathbf{X}} \exp(\mathbf{e}^T \mathbf{x}_i). \tag{5.4}$$

The value of $I(\mathbf{X}) \in [0, 1]$ is a measure of isotropy of the given embedding $\mathbf{X}$. A higher $I(\cdot)$ means more isotropic and a better quality of the embedding. It is found making the singular values close to each other can effectively improve embedding isotropy.

5.2 Proposed Quantization

The preceding section hints that maximizing the isotropy and maximizing the bit variance are opposite in action: The former intends to make the singular values close by removing the largest singular values, whereas the latter removes the smallest singular values and maintains the largest. Given the success of isotropy in NLP applications, we propose to minimize the quantization loss while improving the isotropy, rather than maximizing the bit variance. We call the proposed method *isotropic iterative quantization*, IIQ.

The key idea of ITQ is based on the observation that $\mathbf{U} = \mathbf{WR}$ is still a solution to the objective function of (5.1). In our approach IIQ, we show that the orthogonal transformation maintains the isotropy of the input embedding, so that we could apply a similar alternating procedure as in ITQ to minimize the quantization loss. As a result, our method is composed of three steps: maximizing isotropy, reducing dimension, and minimizing quantization loss.

5.2.1 Maximize Isotropy.

The isotropy measure $I(\mathbf{X})$ can be approximated as following [74] :

$$\hat{I}(\mathbf{X}) = \frac{|\mathbf{X}| - ||\mathbf{1}^T\mathbf{X}|| + \frac{1}{2}\sigma_{\min}^2}{|\mathbf{X}| + ||\mathbf{1}^T\mathbf{X}|| + \frac{1}{2}\sigma_{\max}^2}, \tag{5.5}$$

where $\sigma_{\min}$ and $\sigma_{\max}$ are the smallest and largest singular values of $\mathbf{X}$, respectively. For $\hat{I}(\mathbf{X})$ to be 1, the middle term $||\mathbf{1}^T\mathbf{X}||$ on both the numerator and the denominator must be zero and additionally $\sigma_{\min} = \sigma_{\max}$. The former requirement can be easily satisfied by the zero-centering given embeddings:

$$\begin{aligned} \mathbf{u} &= \frac{1}{n}\mathbf{1}^T \cdot \mathbf{X} \\ \bar{\mathbf{X}} &= \mathbf{X} - \mathbf{1} \cdot \mathbf{u}^T, \end{aligned} \tag{5.6}$$

where $||\mathbf{1}^T\bar{\mathbf{X}}|| = 0$. The latter may be approximately achieved by removing the large singular values such that the rest of the singular values are close to each other. A reason why removing the large singular values makes the rest close, is that often the large singular values have substantial gaps while the rest are clustered. However, removing singular components does not change its dimension. We denote the maximized result as $\hat{\mathbf{X}}$.

5.2.2 Dimension Reduction.

To make our method more flexible, we perform a dimension reduction afterward by using PCA. This step essentially removes the smallest singular values so that the clustering of the singular values may be further tightened. Note that PCA won't affect the maximized isotropy of given embeddings, since it only works on the singular values that are already closed to each other after previous step. One can treat the dimension as a hyperparameter, tailored for each data set.

5.2.3 Minimize Quantization Loss.

Given a solution $\hat{\mathbf{X}}$ to the maximization of (5.5), we prove that multiplying $\hat{\mathbf{X}}$ with an orthogonal matrix $\mathbf{R}$ results in the same $\hat{I}(\mathbf{X})$. In other words, we could minimize the quantization loss (5.2) while maintaining the isotropy.

Proposition 8. *If* $\hat{\mathbf{X}} \in \mathbb{R}^{n\times d}$ *is isotropic and* $\mathbf{R} \in \mathbb{R}^{d\times d}$ *is orthogonal, then* $\mathbf{U} = \hat{\mathbf{X}}\mathbf{R}$ *admits* $\hat{I}(\mathbf{U}) = \hat{I}(\hat{\mathbf{X}})$.

Proof. Given that $\mathbf{R}$ is orthogonal, we first prove that $\mathbf{U}$ has the same singular values as does $\hat{\mathbf{X}}$. Let $\hat{\mathbf{X}}$ have the singular value decomposition (SVD)

$$\hat{\mathbf{X}} = \mathbf{P}\text{diag}(\sigma_{\max}, \ldots, \sigma_{\min})\mathbf{Q}, \tag{5.7}$$

where $\mathbf{P} \in \mathbb{R}^{n\times d}$ and orthogonal matrix $\mathbf{Q} \in \mathbb{R}^{d\times d}$. Let $\mathbf{Q}' = \mathbf{Q}\mathbf{R}$. Then, we have

$$\mathbf{U} = \mathbf{P}\text{diag}(\sigma_{\max}, \ldots, \sigma_{\min})\mathbf{Q}'. \tag{5.8}$$

Since $\mathbf{Q}'$ is also orthogonal, Equation (5.8) gives the SVD of $\mathbf{U}$. Therefore, $\mathbf{U}$ has the same singular values as does $\hat{\mathbf{X}}$.

Moreover, $||\mathbf{1}^T\mathbf{U}|| = ||\mathbf{1}^T\hat{\mathbf{X}}\mathbf{R}|| = 0$, thus $\mathbf{U}$ is also zero-centered. By Equation (5.5), we conclude $\hat{I}(\mathbf{U}) = \hat{I}(\hat{\mathbf{X}})$. □

With the given proof, we can always use an orthogonal matrix $\mathbf{R}$ to reduce the quantization loss. The iterative optimization strategy as in ITQ [73] is adopted to minimize the quantization loss. Two alternating steps lead to a local minimum. First, compute $\mathbf{B}$ given $\mathbf{R}$:

$$\mathbf{B} = \text{sgn}(\hat{\mathbf{X}} \cdot \mathbf{R}). \tag{5.9}$$

Second, update $\mathbf{R}$ given $\mathbf{B}$. The update minimizes the quantization loss, which essentially solves the orthogonal Procrustes problem. The solution is given by

$$\begin{aligned} \mathbf{S} \cdot \mathbf{\Omega} \cdot \hat{\mathbf{S}}^T &= \text{SVD}(\mathbf{B}^T \cdot \hat{\mathbf{X}} \cdot \mathbf{R}) \\ \mathbf{R} &= \hat{\mathbf{S}} \cdot \mathbf{S}^T, \end{aligned} \tag{5.10}$$

where $\text{SVD}(\cdot)$ is the singular value decomposition function and $\mathbf{\Omega}$ is the diagonal matrix of singular values.

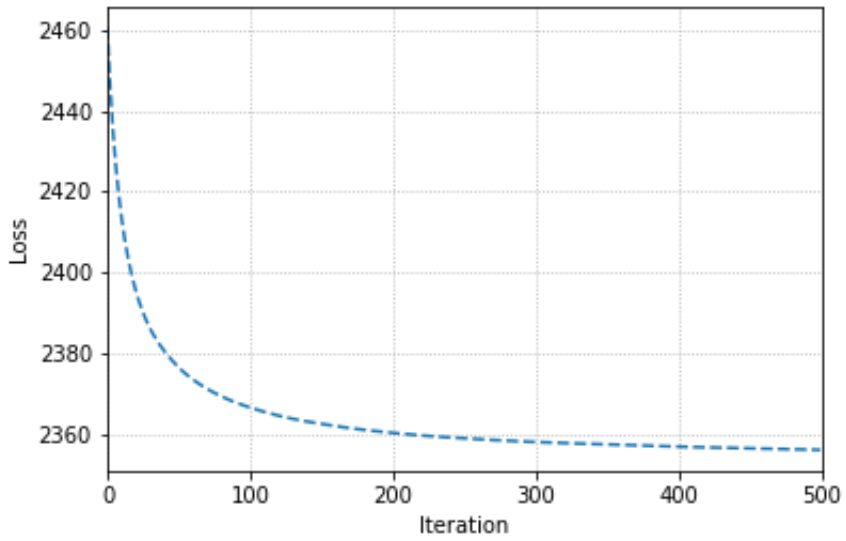


Figure 5.1: Quantization loss curve of 50000 embedding vectors from a pre-trained CNN model.

This iterative updating strategy runs until a local optimal solution is found. Fig. 5.1 shows an example of the quantization loss curve. This result is similar to the behavior of

Algorithm 4: Isotropic Iterative Quantization

Input: $\mathbf{X} \in \mathbb{R}^{n \times d}, D, T, O$
Output: $\mathbf{B}$

1 $\mathbf{u} \leftarrow \frac{1}{n}\mathbf{1}^T \cdot \mathbf{X}$;
2 $\bar{\mathbf{X}} \leftarrow \mathbf{X} - \mathbf{1} \cdot \mathbf{u}^T$;
3 $\mathbf{S} \cdot \mathbf{\Omega} \cdot \hat{\mathbf{S}}^T \leftarrow \text{SVD}(\bar{\mathbf{X}})$;
4 Set top D singular values in $\mathbf{\Omega}$ as 0;
5 $\hat{\mathbf{X}} \leftarrow \mathbf{S} \cdot \mathbf{\Omega} \cdot \hat{\mathbf{S}}^T$;
6 **if** $O < d$ **then**
7 $\quad \hat{\mathbf{X}} \leftarrow \text{PCA}(\hat{\mathbf{X}}, O)$
8 **end**
9 Randomly initialize an orthogonal matrix $\mathbf{R}$;
10 **for** $i \leftarrow 1$ *to* T **do**
11 $\quad \mathbf{U} \leftarrow \hat{\mathbf{X}} \cdot \mathbf{R}$;
12 $\quad \mathbf{B} \leftarrow \text{sgn}(\mathbf{U})$;
13 $\quad \mathbf{S} \cdot \mathbf{\Omega} \cdot \hat{\mathbf{S}}^T \leftarrow \text{SVD}(\mathbf{B}^T \cdot \mathbf{U})$;
14 $\quad \mathbf{R} \leftarrow \hat{\mathbf{S}} \cdot \mathbf{S}^T$;
15 **end**
16 **return** $sgn(\hat{\mathbf{X}} \cdot \mathbf{R})$;

ITQ, the authors of which proposed using early stopping to terminate iteration in practice. We follow the guidance and run only 50 iterations in our experiments. Our method is an unsupervised approach, which does not require any label supervision. Therefore, it can be applied independently of downstream tasks and no fine tuning is needed. This advantage benefits many problems where embeddings often slow down the learning process because of the high space and computation complexity.

We present the pseudocode of the proposed IIQ method in Algorithm 4. The input D denotes the number of top singular values to be removed, T denotes the number of iterations for minimizing the quantization loss, and O denotes the dimension of the output binary vectors. The first two lines make zero-centered embedding. Lines 3 to 5 maximize the isotropy. Lines 6 to 8 reduce the embedding dimension. Lines 9 to 15 minimize the quantization loss. Within the iteration loop, lines 11 to 12 update $\mathbf{B}$ based on the most recent $\mathbf{R}$, whereas lines 13 to 14 update $\mathbf{R}$ given the updated $\mathbf{B}$. The last line uses the final transformation $\mathbf{R}$ to return the binary embeddings as output.

5.3 Experimental Results

We run the proposed method on pre-trained embedding vectors and evaluate the compressed embedding in various NLP tasks. For some tasks, the evaluation is directly conducted over the embedding (e.g., measuring the cosine similarity between word vectors); whereas for others, a classifier is trained with the embedding. We conduct all experiments in Python by using Numpy and Keras. The environment is Ubuntu 16.04 with Intel(R) Xeon(R) CPU E5-2698.

Pre-trained Embedding. We perform experiments with the GloVe embedding [67] and the HDC embedding [80]. The GloVe embedding is trained from 42B tokens of Common Crawl data. The HDC embedding is trained from public Wikipedia. It has a better quality than GloVe because the training process considers both syntagmatic and paradigmatic relations. All embedding vectors are used in the experiment without vocabulary truncation or post-processing.

In addition, we evaluate embedding compression on a CNN model pre-trained with the IMDB data set. Different from the prior case, the embedding from CNN is trained with supervised class labels. We compress the embedding and retrain the model to evaluate performance. This way enables us to compare with other compression methods fairly.

Configuration. We compare IIQ with the traditional ITQ method [73], the pruning method [77], deep compositional code learning (DCCL) [81] and a recent method [82] we name as NLB. The pruning method is set to prune 95% of the words for a similar compression ratio. The DCCL method is similarly configured. We run NLB with its default setting. We train the DCCL method for 200 epochs and set the batch size to be 1024 for GloVe and 64 for HDC. For our method, we set the iteration number T to be 50 since early stopping works sufficiently well. We set the same iteration number for ITQ. We also set the parameter D to be 2 for HDC, and 14 for Glove embedding. Note that we perform all vector operations in real domain on the platform [83] and [84].

Table 5.1 lists the experiment configurations with method name, dimension, embedding value type, and compression ratio. The baseline means the original embedding. Our method starts with "IIQ," followed by the compression ratio. The "dimension" column gives the

Table 5.1: Experiment Configurations.

	Method	Dimension	Comp. Ratio
GloVe	Baseline	1917494×300	1
	Prune	1917494×300	20
	DCCL	$M = 32, K = 256$	32
	NLB	1917494×300	32
	ITQ	1917494×300	32
	IIQ-32	1917494×300	32
	IIQ-64	1917494×150	64
	IIQ-128	1917494×75	128
HDC	Baseline	388723×300	1
	Prune	388723×300	20
	DCCL	$M = 32, K = 128$	29
	NLB	388723×300	32
	ITQ	388723×300	32
	IIQ-32	388723×300	32
	IIQ-64	388723×150	64
	IIQ-128	388723×75	128

number of vectors and the vector dimension. For DCCL, we list the parameters M and K that determine the compression ratio. Note that we use single precision for real values. The last column shows the compression ratio, which is the the size of the original embedding over that of the compressed one. Thus, the compression from real value to binary is 32. Moreover, we also apply dimension reduction in IIQ so that higher compression ratio is possible.

5.3.1 Word Similarity

The task measures Spearman's rank correlation between word vector similarity and human rated similarity. A higher correlation means a better quality of the word embedding. The similarity between two words is computed as the cosine of the corresponding vectors, i.e., $\cos(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}/(||\mathbf{x}|| \cdot ||\mathbf{y}||)$, where $\mathbf{x}$ and $\mathbf{y}$ are two word vectors. Out-of-vocabulary (OOV) words are replaced by the mean vector.

In this experiment, seven data sets are used, including MEN [85] with 3000 pairs of words obtained from Amazon crowdsourcing; MTurk [86] with 287 pairs, focusing on word semantic relatedness; RG65 [87] with 65 pairs, an early published dataset; RW [88] with 2034 pairs of rare words selected based on frequencies; SimLex999 [89] with 999 pairs, aimed at

Table 5.2: Word Similarity Results.

	Method	MEN	MTurk	RG65	RW	SimLex999	TR9856	WS353
GloVe	Baseline	73.62	64.50	81.71	37.43	37.38	9.67	69.07
	Prune	17.97	22.09	39.66	12.45	-0.37	8.31	14.52
	DCCL	54.46	50.46	63.89	28.04	25.48	7.91	54.55
	NLB	73.99	**64.98**	72.07	40.86	40.52	**14.00**	66.09
	ITQ	57.37	52.93	72.08	25.10	26.23	8.98	55.00
	IIQ-32	**76.43**	63.33	**78.16**	**41.35**	**41.87**	9.80	**72.22**
	IIQ-64	71.55	58.37	74.94	37.61	38.80	12.81	67.99
	IIQ-128	59.25	50.42	62.39	28.71	33.25	12.31	53.56
HDC	Baseline	76.03	65.77	80.58	46.34	40.68	20.71	76.81
	Prune	46.83	41.49	56.14	29.84	26.27	15.27	52.06
	DCCL	68.82	55.78	72.23	**39.33**	35.02	**18.41**	66.09
	NLB	72.06	61.57	72.58	35.45	38.50	11.71	67.20
	ITQ	72.31	61.68	74.70	37.01	37.40	9.69	72.32
	IIQ-32	**74.37**	**66.71**	**78.04**	38.75	**39.35**	9.63	**75.32**
	IIQ-64	66.32	56.73	65.77	35.63	36.22	11.33	72.70
	IIQ-128	55.83	51.33	45.76	32.03	29.45	12.61	58.54

genuine similarity estimation; TR9856 [90] with 9856 pairs, containing many acronyms and name entities; and WS353 [91] with 353 pairs of mostly verbs and nouns. The experiment is conducted on the platform [83].

Table 5.2 summarizes the results. The performance of IIQ degrades as the compression ratio increases. This is expected, since a higher compression ratio leads to more loss of information. In addition, our IIQ method consistently achieves better results than ITQ, DCCL, NLB and the pruning method. Particularly, one sees that on the Men data set, IIQ even outperforms the baseline embedding Glove. Another observation is that on TR9856, a higher compression ratio surprisingly yields better results for IIQ. We speculate that the cause is the multi-word term relations unique to TR9856. Interestingly, the pruning method results in negative correlation in SimLex999 for the GloVe embedding. This means that pruning too many small values inside word embedding can drastically destroy the embedding quality.

5.3.2 Categorization

The task is to cluster words into different categories. The performance is measured by purity, which is defined as the fraction of correctly classified words. We run the experiment

using agglomerative clustering and k-means clustering, and select the highest purity as the final result for each embedding. This experiment is conducted on the platform [83] where OOV words are replaced by the mean vector.

Four data sets are used in this experiment: Almuhareb-Poesio (AP) [92] with 402 words in 21 categories; BLESS [93] with 200 nouns (animate or inanimate) in 17 categories; Battig [94] with 5231 words in 56 taxonomic categories; and ESSLI2008 Workshop [95] with 45 verbs in 9 semantic categories.

Table 5.3 lists evaluation results for GloVe and HDC embeddings. One sees that the proposed IIQ method works better than ITQ, DCCL, and the pruning method on all data sets. But NLB sometimes achieves the best result for example on Battig. For ESSLI, IIQ even outperforms the original GloVe and HDC embedding.

Table 5.3: Categorization Results.

	Method	**AP**	**BLESS**	**Battig**	**ESSLI**
GloVe	Baseline	62.94	78.50	45.13	57.78
	Prune	38.56	46.00	23.42	42.22
	DCCL	52.24	75.00	36.09	48.89
	NLB	59.45	78.50	**43.39**	**66.67**
	ITQ	58.71	76.50	40.76	48.89
	IIQ-32	**64.18**	**80.00**	41.98	60.00
	IIQ-64	56.22	76.50	37.49	51.11
	IIQ-128	45.02	69.00	31.43	44.44
HDC	Baseline	65.42	81.50	43.18	60.00
	Prune	34.33	48.00	23.28	51.11
	DCCL	55.97	74.50	40.16	53.33
	NLB	59.20	75.50	**41.88**	62.22
	ITQ	57.21	77.50	41.04	55.56
	IIQ-32	**61.69**	**78.00**	41.29	**62.22**
	IIQ-64	48.51	72.50	35.90	53.33
	IIQ-128	43.03	57.50	28.50	62.22

5.3.3 Topic Classification

In this experiment, we perform topic classification by using sentence embedding. The embedding is computed as the average of the corresponding word vectors. The average of binary embedding is fed to the classifier in single precision. Missing words are treated as zero and so are OOV words. In this task, we train a Multi-Layer Perceptron (MLP) as the

classifier for each method. Due to the different size of embeddings, we train 10 epochs for all Glove embeddings and 4 epochs for all HDC embedding. Five-fold cross validation is used to report classification accuracy.

Four data sets are selected from [96], including movie review (MR), customer review (CR), opinion-polarity (MPQA), and subjectivity (SUBJ). Similar performance is achieved by using the original embedding. The experiment is conducted on the platform of [84].

Table 5.4 shows the results for each method. Similar to the previous tasks, the proposed IIQ method consistently performs better than ITQ, pruning, and DCCL. The only exception is that for MPQA and SUBJ, DCCL and NLB achieves the best result for the GloVe embedding respectively. As the compression ratio increases, IIQ encounters performance degrade.

Table 5.4: Topic Classification Results.

	Method	**CR**	**MPQA**	**MR**	**SUBJ**
GloVe	Baseline	78.78	87.16	76.42	91.29
	Prune	73.48	81.93	71.97	87.19
	DCCL	77.27	**85.6**	74.74	89.56
	NLB	75.36	85.77	73.01	**89.92**
	ITQ	71.79	84.11	73.18	89.55
	IIQ-32	**77.7**	85.15	**74.96**	89.87
	IIQ-64	75.07	83.02	73.17	88.14
	IIQ-128	72.56	80.55	69.93	84.29
HDC	Baseline	76.40	86.61	75.71	90.86
	Prune	70.97	78.84	67.56	83.58
	DCCL	74.68	84.2	73.32	89.43
	NLB	70.89	84.51	73.18	89.48
	ITQ	73.57	84.44	72.3	89.46
	IIQ-32	**76.32**	**84.77**	**73.51**	**89.91**
	IIQ-64	72.18	82.07	70.32	87.41
	IIQ-128	70.83	77.62	67.89	84.62

5.3.4 Sentiment Analysis

In this experiment, we evaluate over the embedding input to a pre-trained Convolutional Neural Network (CNN) model on the IMDB data set [97]. The CNN model follows the Keras tutorial [98]. We train 50,000 embedding vectors in 300 dimensions. The model is composed of an embedding layer, followed by a dropout layer with probability 0.2, a 1D

convolution layer with 250 filters and kernel size 3, a 1D max pooling layer, a fully connected layer with hidden dimension 250, a dropout layer with probability 0.2, a ReLU activation layer, and a single output fully connected layer with sigmoid activation. Moreover, we use adam optimizer with learning rate 0.0001, sentence length 400, batch size 128, and train for 20 epochs. Input embedding fed into CNN is kept fixed (not trainable).

Table 5.5: Configurations for IMDB Classification.

Method	**Dimension**	**Comp. Ratio**
Baseline	50000×300	1
Prune	50000×300	20
DCCL	$M = 32, K = 32$	27
NLB	50000×300	32
ITQ	50000×300	32
IIQ-32	50000×300	32
IIQ-64	50000×150	64
IIQ-128	50000×75	128

The data set contains 25,000 movie reviews for training and another 25,000 for testing. We randomly separate 5,000 reviews from the training set as validation data. The model with the best performance on the validation set is kept as the final model for measuring test accuracy. Moreover, all results are averaged from 10 runs for each embedding. The baseline model is the pre-trained CNN model with 87.89% accuracy. Table 5.5 summarizes the configurations for this experiment. All configurations are similar to the previous experiments. The DCCL method is now configured with $M = 32$ and $K = 32$ to achieve a similar compression ratio.

We present in Fig. 5.2 the result of each embedding. The histogram shows the average accuracy of 10 runs experiments for each method and the error bar shows the standard deviation. One sees that among all compression methods, IIQ achieves the least performance degrade. IIQ with compression ratio 64 is the best.

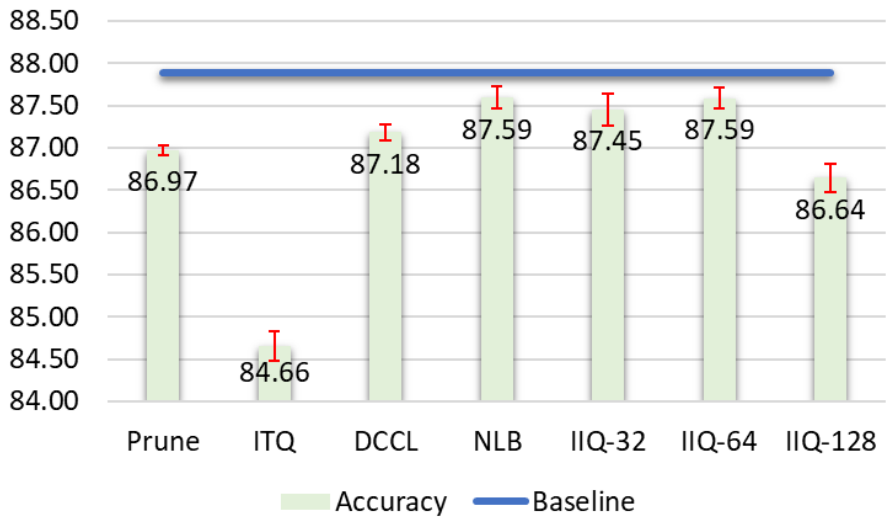
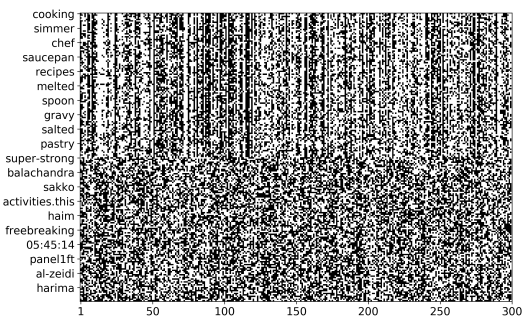
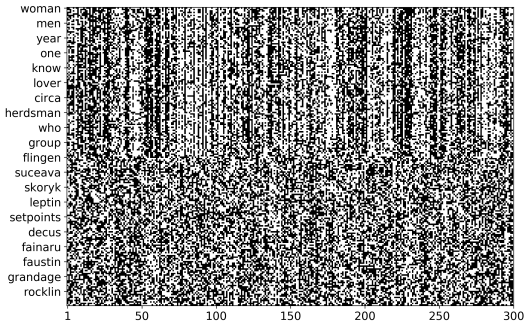


Figure 5.2: IMDB CNN Test Accuracy Results.



(a) Nearest and furthest 100 words of "cook" in IIQ-GloVe.



(b) Nearest and furthest 100 words of "man" in IIQ-HDC.

Figure 5.3: Visualizing Binary IIQ Word Embedding.

5.3.5 Visualization

We visualize the binary IIQ embedding in Fig. 5.3 The nearest and furthest 100 word vectors are shown. The distance is calculated by the dot product. Fig. 5.3(a) shows the IIQ-compressed GloVe embedding and Fig. 5.3(b) shows the IIQ-compressed HDC embedding. The y axis lists every 10 words and the x axis is the dimension of the embedding. One sees that similar word vectors have similar patterns in many dimensions. A white column means that the dimension is zero for all words. A black column means one. Moreover, there is obvious difference between nearest and furthest words.

5.4 Conclusion

We present an isotropic iterative quantization (IIQ) method for compressing word embeddings. While it is based on the ITQ method in image retrieval, it also maintains the embedding isotropy. We evaluate the proposed method on GloVe and HDC embeddings and show that it is effective for word similarity, categorization, and several other downstream tasks. For pre-trained embeddings that are less isotropic (e.g., GloVe), IIQ performs better than ITQ owing to the improvement on isotropy. These findings are based on a 32-fold (and higher) compression ratio. The results point to promising deployment of trained neural network models with word embeddings on resource constrained platforms in real life.

Chapter 6

Conclusions and Future Work

This thesis aims at developing a structure matrix based DNN compression framework. It mostly focuses on the application into computer vision, natural language processing models. Both the forward and backward propagation are derived and demonstrate the effectiveness of structure in terms of space and time complexity. We have studied the use of low displacement rank, particularly the circulant structure in DNN models. In addition, we presented the approximation algorithm converting a non-structured model into our structured model. It is found that structure matrix based compression is useful in compressing models as large as AlexNet and VGG.

Moreover, we also studied the word embedding compression algorithm. The concept of isotropy is a special property for word embedding. We integrated this property into the locality sensitive hashing methods and developed our isotropic iterative quantizaiton. Its effectiveness has been shown over different NLP tasks when compared with traditional methods and also the state-of-the-art algorithms.

In the future, we'd like to investigate the rank learning under the low displacement rank framework, and also the structured sparsity learning in the permuted diagonal matrices. More specifically, we want to develop from the optimization perspective such as proposing a new optimizer for structure learning.

Lastly, we also want to study the application of our compression algorithms into different areas, such as image segmentation, language modeling and even generative adversarial networks. In summary, the essential goal of this and its future research is providing an efficient and low cost training and inference solution for the machine learning community.

References

[1] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[2] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[3] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[4] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[5] Siyu Liao, Yi Xie, Xue Lin, Yanzhi Wang, Min Zhang, and Bo Yuan. Reduced-complexity deep neural networks design using multi-level compression. *IEEE Transactions on Sustainable Computing*, 2017.

[6] Liang Zhao, Siyu Liao, Yanzhi Wang, Zhe Li, Jian Tang, and Bo Yuan. Theoretical properties for neural networks with weight matrices of low displacement rank. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 4082–4090. JMLR.org, 2017.

[7] Siyu Liao, Zhe Li, Liang Zhao, Qinru Qiu, Yanzhi Wang, and Bo Yuan. Circconv: A structured convolution with low complexity. In *AAAI*, 2019.

[8] Guosheng Hu, Yongxin Yang, Dong Yi, Josef Kittler, William Christmas, Stan Z Li, and Timothy Hospedales. When face recognition meets with deep learning: an evaluation of convolutional neural networks for face recognition. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 142–150, 2015.

[9] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[10] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[11] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[14] Song Han. Efficient methods and hardware for deep learning. 2017.

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

[16] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[17] Misha Denil, Babak Shakibi, Laurent Dinh, Marc'Aurelio Ranzato, and Nando De Freitas. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.

[18] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[19] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[20] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wangchun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[21] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[22] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[23] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a backpropagation network. In *Advances in neural information processing systems*, pages 396–404, 1990.

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

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[26] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

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

[28] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[29] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.

[30] Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9329–9338, 2018.

[31] Jinmian Ye, Linnan Wang, Guangxi Li, Di Chen, Shandian Zhe, Xinqi Chu, and Zenglin Xu. Learning compact recurrent neural networks with block-term tensor decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9378–9387, 2018.

[32] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[33] Zhangjie Cao, Mingsheng Long, Jianmin Wang, and Philip S Yu. Hashnet: Deep learning to hash by continuation. In *Proceedings of the IEEE international conference on computer vision*, pages 5608–5617, 2017.

[34] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2074–2082, 2016.

[35] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[36] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

[37] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[38] Victor Pan. *Structured matrices and polynomials: unified superfast algorithms*. Springer Science & Business Media, 2001.

[39] Yu Cheng, Felix X Yu, Rogerio S Feris, Sanjiv Kumar, Alok Choudhary, and Shi-Fu Chang. An exploration of parameter redundancy in deep networks with circulant projections. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2857–2865, 2015.

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

[41] Dario Bini, Victor Pan, and Wayne Eberly. Polynomial and matrix computations volume 1: Fundamental algorithms. *SIAM Review*, 38(1):161–164, 1996.

[42] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[43] Siyu Liao, Zhe Li, Xue Lin, Qinru Qiu, Yanzhi Wang, and Bo Yuan. Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 458–465. IEEE, 2017.

[44] Siyu Liao, Ashkan Samiee, Chunhua Deng, Yu Bai, and Bo Yuan. Compressing deep neural networks using toeplitz matrix: Algorithm design and fpga implementation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1443–1447. IEEE, 2019.

[45] Mansoor Rezghi and Lars Eldén. Diagonalization of tensors with circulant structure. *Linear Algebra and its Applications*, 435(3):422–447, 2011.

[46] Victor Pan. *Structured matrices and polynomials: unified superfast algorithms.* Springer Science & Business Media, 2012.

[47] Moody T Chu and Robert J Plemmons. Real-valued low rank circulant approximation. *SIAM Journal on Matrix Analysis and Applications*, 24(3):645–659, 2003.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[50] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[51] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[52] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[53] Zhisheng Wang, Jun Lin, and Zhongfeng Wang. Accelerating recurrent neural networks: A memory-efficient approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2763–2775, 2017.

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

[55] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

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

[57] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 367–379. IEEE Press, 2016.

[58] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally. Scnn: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 27–40. ACM, 2017.

[59] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ACM SIGARCH Computer Architecture News*, volume 44, pages 1–13. IEEE Press, 2016.

[60] Jiecao Yu, Andrew Lukefahr, David Palframan, Ganesh Dasika, Reetuparna Das, and Scott Mahlke. Scalpel: Customizing dnn pruning to the underlying hardware parallelism. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 548–560. ACM, 2017.

[61] Chunhua Deng, Siyu Liao, Yi Xie, Keshab K Parhi, Xuehai Qian, and Bo Yuan. Permdnn: Efficient compressed dnn architecture with permuted diagonal matrices. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 189–202. IEEE, 2018.

[62] Minh-Thang Luong and Christopher D Manning. Stanford neural machine translation systems for spoken language domains. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 76–79, 2015.

[63] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[64] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[65] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[66] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[67] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[68] Cicero dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[69] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[70] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[71] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. *arXiv preprint arXiv:1711.01068*, 2017.

[72] Karol Grzegorczyk and Marcin Kurdziel. Binary paragraph vectors. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, pages 121–130, 2017.

[73] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, 2013.

[74] Jiaqi Mu, Suma Bhat, and Pramod Viswanath. All-but-the-top: Simple and effective postprocessing for word representations. *arXiv preprint arXiv:1702.01417*, 2017.

[75] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[76] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 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*, pages 6655–6659. IEEE, 2013.

[77] Abigail See, Minh-Thang Luong, and Christopher D Manning. Compression of neural machine translation models via pruning. *arXiv preprint arXiv:1606.09274*, 2016.

[78] Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6196–6203, 2019.

[79] Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, and Andrej Risteski. A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4:385–399, 2016.

[80] Fei Sun, Jiafeng Guo, Yanyan Lan, Jun Xu, and Xueqi Cheng. Learning word representations by jointly modeling syntagmatic and paradigmatic relations. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 136–145, 2015.

[81] Raphael Shu and Hideki Nakayama. Compressing word embeddings via deep compositional code learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[82] Julien Tissier, Christophe Gravier, and Amaury Habrard. Near-lossless binarization of word embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7104–7111, 2019.

[83] Stanisław Jastrzebski, Damian Leśniak, and Wojciech Marian Czarnecki. word-embeddings-benchmarks. `https://github.com/kudkudak/word-embeddings-benchmarks`, 2015.

[84] Alexis Conneau and Douwe Kiela. Senteval: An evaluation toolkit for universal sentence representations. *arXiv preprint arXiv:1803.05449*, 2018.

[85] Elia Bruni, Nam-Khanh Tran, and Marco Baroni. Multimodal distributional semantics. *Journal of Artificial Intelligence Research*, 49:1–47, 2014.

[86] Kira Radinsky, Eugene Agichtein, Evgeniy Gabrilovich, and Shaul Markovitch. A word at a time: computing word relatedness using temporal semantic analysis. In *Proceedings of the 20th international conference on World wide web*, pages 337–346. ACM, 2011.

[87] Herbert Rubenstein and John B Goodenough. Contextual correlates of synonymy. *Communications of the ACM*, 8(10):627–633, 1965.

[88] Thang Luong, Richard Socher, and Christopher Manning. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113, 2013.

[89] Felix Hill, Roi Reichart, and Anna Korhonen. Simlex-999: Evaluating semantic models with (genuine) similarity estimation. *Computational Linguistics*, 41(4):665–695, 2015.

[90] Ran Levy, Liat Ein-Dor, Shay Hummel, Ruty Rinott, and Noam Slonim. Tr9856: A multi-word term relatedness benchmark. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 419–424, 2015.

[91] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics, 2009.

[92] Abdulrahman Almuhareb and Massimo Poesio. Concept learning and categorization from the web. In *proceedings of the annual meeting of the Cognitive Science society*, 2005.

[93] Marco Baroni and Alessandro Lenci. How we blessed distributional semantic evaluation. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, pages 1–10. Association for Computational Linguistics, 2011.

[94] William F Battig and William E Montague. Category norms of verbal items in 56 categories a replication and extension of the connecticut category norms. *Journal of experimental Psychology*, 80(3p2):1, 1969.

[95] S Evert M Baroni and A Lenci. Bridging the gap between semantic theory and computational simulations: Proceedings of the esslli workshop on distributional lexical semantics. In *Proceedings of the esslli workshop on distributional lexical semantics*, 2008.

[96] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th annual meeting of the association for computational linguistics: Short papers-volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[97] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[98] Francois Chollet et al. Keras documentation, convolution1d for text classification. `https://keras.io/examples/imdb_cnn/`. Accessed: 2019-08.