Learning Low-Rank Tensor Cores with Probabilistic ℓ_0-Regularized Rank Selection for Model Compression

Tianxiao Cao[1] , **Lu Sun**[1] , **Canh Hao Nguyen**[2] and **Hiroshi Mamitsuka**[2]
[1]ShanghaiTech University
[2]Kyoto University
{caotx,sunlu1}@shanghaitech.edu.cn, {canhhao,mami}@kuicr.kyoto-u.ac.jp

Abstract

Compressing deep neural networks is of great importance for real-world applications on resource-constrained devices. Tensor decomposition is one promising answer that retains the functionality and most of the expressive power of the original deep models by replacing the weights with their decomposed cores. Decomposition with optimal ranks can achieve a good compression-accuracy trade-off, but it is expensive to optimize due to its discrete and combinatorial nature. A common practice is to set all ranks equal and tune one hyperparameter, but it may significantly harm the flexibility and generalization. In this paper, we propose a novel automatic rank selection method for deep model compression that allows learning model weights and decomposition ranks simultaneously. We propose to penalize the ℓ_0 (quasi-)norm of the slices of decomposed tensor cores during model training. To avoid combinatorial optimization, we develop a probabilistic formulation and apply an approximate Bernoulli gate to each of the slices of tensor cores, which can be implemented in an end-to-end and scalable framework via gradient descent. It enables the automatic rank selection to be incorporated with arbitrary tensor decompositions and neural network layers such as linear layers, convolutional layers, and embedding layers. Comprehensive experiments on various tasks, including image classification, text sentiment classification, and neural machine translation, demonstrate the superior effectiveness of the proposed method over baselines.

1 Introduction

In the last decade, deep neural networks have achieved excellent results on a variety of ill-posed machine learning tasks in various fields, including image classification and object detection [He *et al.*, 2016] in computer vision (CV), neural machine translation (NMT) and language modeling [Vaswani *et al.*, 2017] in the field of natural language processing (NLP), and music source separation [Rouard *et al.*, 2023]. Many pioneering models have a large number of parameters, which are considered to enhance the feasibility of learning optimization and the expressive power of the model in later studies [Arora *et al.*, 2018]. For example, a ResNet-50 has 25.6M parameters and needs 97.5MB memory storage. The large storage and computation overhead associated with a huge number of model parameters makes it difficult to apply deep neural networks on resource-constrained devices, such as smartphones, wearables, and IoT devices. Several works [Ba and Caruana, 2014] have shown that modern neural networks are highly redundant because most parameters are attributed to a small subset of the learned parameters. This has given rise to the study of various types of model compression techniques, including low-rank approximation [Yaguchi *et al.*, 2019], knowledge distillation [Hinton *et al.*, 2015], pruning, and quantization [Han *et al.*, 2015].

As a promising low-rank technique, tensor decomposition efficiently represents high-order arrays using decomposed tensor cores. It has been an attractive tool in quantum system modeling [Orús, 2019], data mining [Kwon *et al.*, 2023], and applied mathematics applications [Dolgov *et al.*, 2021], and has been applied to compress neural networks [Novikov *et al.*, 2015]. A compact model in the decomposed form can be used to replace the original model with little performance loss. Thanks to the good expressive power of tensor decomposition, tensor decomposition-based compression methods usually result in large compression rates. However, they usually suffer from the rank selection problem, which is particularly serious in the successful chain-structured Tensor-Train [Novikov *et al.*, 2015] and Tensor-Ring [Wang *et al.*, 2018] methods due to their large rank search spaces.

In the data completion problem, it has been extensively studied that ranks are essential influence factors to the expressive power of decomposition [Hashemizadeh *et al.*, 2020]. A similar phenomenon [Li *et al.*, 2021] holds for deep compression since tensor ranks determine the shapes of decomposed cores and thus control the trade-off between model accuracy and compression rate. Early works [Yin *et al.*, 2022] typically select ranks manually for each decomposed layer, where all ranks are set to equal and treated as one hyperparameter. However, it is intuitively over-simplified and usually leads to sub-optimal performance. Several automated non-uniform rank selection methods are proposed, turning out to be better than the simple uniform scheme. However, existing methods rely on iterative large-scale low-rank tensor ap-

proximations [Gusak *et al.*, 2019], retraining [Li *et al.*, 2021], Bayesian network design [Hawkins and Zhang, 2021], or integer approximate searching algorithms such as genetic algorithms [Li *et al.*, 2021] and reinforcement learning [Cheng *et al.*, 2020], which complicate the optimization and require significantly additional training overhead and memory consumption.

To overcome these limitations, in this paper, we propose to apply an approximate Bernoulli gate to each of the slices of tensor cores and apply probabilistic ℓ_0 regularization on the ranks of decomposed tensor cores during training. The proposed ℓ_0-regularized scheme requires only a little extra overhead than training a vanilla tensorized neural network and manages to find a superior set of ranks than baselines. We also introduce an initialization scheme. The technical contributions are summarized as follows.

- We propose a novel gated form of tensor contractions (illustrated in Fig. 2) and a probabilistic ℓ_0-regularized rank selection method that allows simultaneous parameter learning and rank selection, as well as flexible combination with arbitrary tensor decompositions.
- We introduce the scheme of optimization using an efficient and scalable gradient-based optimizer with the help of the Gaussian-based Bernoulli continuous relaxation.
- Our method has been successfully applied for tensorized fully-connected layers, convolutional layers, and embedding layers, and comprehensive experiments on various CV and NLP tasks show its efficiency.

2 Related Work

Tensor Methods for Deep Compression. Tensor decomposition has been successfully applied to deep model compression. Various types of decompositions, *e.g.* Tucker [Gusak *et al.*, 2019], CP [Lebedev *et al.*, 2014], Tensor-Train (TT) [Novikov *et al.*, 2015; Yin *et al.*, 2022], Tensor-Ring (TR) [Wang *et al.*, 2018], and Tensor Network [Hayashi *et al.*, 2019], are used in designing significantly compact neural network layers. Early works focus on fully-connected layers [Novikov *et al.*, 2015; Yang *et al.*, 2017] and convolutional layers [Garipov *et al.*, 2016; Wang *et al.*, 2018], while more recent works generalize to tensorized word embedding [Hrinchuk *et al.*, 2020] and self-attention [Ma *et al.*, 2019]. Our proposed method can be well combined with existing techniques and is not limited to specific types of decomposition, layers, and tasks, as will be shown in the methodology and experiments.

Tensor Rank Selection. A major problem of tensor-based deep compression is how to determine the ranks of the compact tensorized representation of layers. Determining the tensor rank is NP-hard on CP decomposition [Hillar and Lim, 2013], and it is likewise non-trivial on other decompositions, which is commonly circumvented by setting all ranks equal in model compression. This uniform rank selection scheme sacrifices more performance than non-uniform ones [Kodryan *et al.*, 2023]. To achieve non-uniform rank selections for a better compression-performance trade-off, MUSCO [Gusak *et al.*, 2019] iteratively performs low-rank approximation with ranks determined using variational Bayesian matrix factorization [Nakajima *et al.*, 2010] and fine-tuning. [Dai *et al.*, 2023] jointly learns full and low-rank Tucker models and utilizes Tucker decomposition at each iteration to reduce ranks. [Cheng *et al.*, 2020] treats ranks of decomposition as actions and performance plus compression ratio as rewards, and then searches TR ranks using a reinforcement learning agent. [Li *et al.*, 2021] proposes to progressively search TR ranks using neural architecture search techniques with genetic algorithms. Bayesian approaches [Hawkins and Zhang, 2021; Kodryan *et al.*, 2023] model the distributions of TT ranks and learn the proper ranks via smart prior choices. Current approaches typically treat the rank selection problem as combinatorial optimization and apply approximate search algorithms to solve it, resulting in significantly additional computational costs for searching and re-training. In contrast, our proposed method requires only a little additional overhead than training a vanilla tensorized neural network, without sacrificing prediction performance.

Regularization for Sparsity. Prior works in the context of feature selection and neural network pruning have extensively investigated embedded regularization approaches. To mention a few relevant, LASSO [Tibshirani, 1996] selects feature subsets with convex ℓ_1 constraints. Grouped ℓ_1 regularization allows structured neural network pruning [Wen *et al.*, 2016]. Non-differentiable ℓ_0 is introduced by its relaxed alternatives, such as Concrete [Maddison *et al.*, 2016], Hard Concrete [Louizos *et al.*, 2018; Zhen *et al.*, 2022] and stochastic gate approximation [Yamada *et al.*, 2020]. Despite the difficulty of non-convex optimization, ℓ_0-based approaches outperform the convex ℓ_1 in both feature selection [Yamada *et al.*, 2020] and model compression [Louizos *et al.*, 2018]. Therefore, it is promising to impose ℓ_0 regularization to learn the appropriate tensor rank of the weight parameters and prune tensorized neural networks.

3 Preliminary

Notations. A tensor, in machine learning literature, refers to a multi-dimensional array. Throughout the paper, 'x', '$\boldsymbol{X}$', and '$\boldsymbol{\mathcal{X}}$' denote scalar, matrix, and tensor respectively. The $(i_1, \ldots, i_N)$-th entry of N-order tensor $\boldsymbol{\mathcal{A}}$ is denoted as $\boldsymbol{\mathcal{A}}(i_1, \ldots, i_N)$. We use the notation $[n] = \{1, \ldots, n\}$, where n is a positive integer. The ℓ_0 (quasi-)norm $\|\cdot\|_0$ refers to the number of non-negative entries of a matrix.

Contraction. Tensor contraction is a mathematical operation that extends the concept of matrix multiplication to higher-dimensional tensors. It involves summing the products of corresponding elements along matched indices, resulting in a new tensor with reduced dimensions. A formal definition can be found in [Kodryan *et al.*, 2023]. For instance, given two tensor $\boldsymbol{\mathcal{G}}_1 \in \mathbb{R}^{r_1 \times d_1 \times r_2}$ and $\boldsymbol{\mathcal{G}}_2 \in \mathbb{R}^{r_2 \times d_2 \times r_1}$, the output tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{d_1 \times d_2}$ is obtained through tensor contractions along two matched indices as follow:

$$\begin{aligned}\boldsymbol{\mathcal{A}}(a_1, a_2) &= \left\{\boldsymbol{\mathcal{G}}_1 \times_{3,1}^{1,3} \boldsymbol{\mathcal{G}}_2\right\}(a_1, a_2) \\ &= \sum_{i,j=1}^{r_1, r_2} \boldsymbol{\mathcal{G}}_1(i, a_1, j)\boldsymbol{\mathcal{G}}_2(j, a_2, i).\end{aligned} \tag{1}$$

Tensor Decomposition. In this paper, we adopt TR [Wang *et al.*, 2018] as the main decomposition method for compression, and the same idea applies to other methods as well. A TR decomposition [Zhao *et al.*, 2016] represents $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ with cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^d$ as follows:

$$\boldsymbol{\mathcal{X}}(i_1, \ldots, i_d) = \sum_{r_1, \ldots, r_d} \boldsymbol{\mathcal{G}}_1(r_d, i_1, r_1) \cdots \boldsymbol{\mathcal{G}}_d(r_{d-1}, i_d, r_d), \tag{2}$$

where $\boldsymbol{\mathcal{G}}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}$ for $i \in [d]$ and $R_0 = R_d$. The intermediate dimensions $\{R_i\}_{i=1}^d$ are the TR ranks. Constraining $R_0 = R_d = 1$ yields a TT decomposition model [Oseledets, 2011]. With the rest intermediate ranks equal, a TR decomposition can be seen as a linear combination of TT decompositions and thus has stronger expressive power.

4 Methodology

Suppose that X is the random variable in the input domain $\mathbb{R}^I$ with corresponding variable Y in the output domain $\mathbb{R}^O$, let $f_{\boldsymbol{\theta}}(\cdot)$ be the L-layer neural network predictor parameterized by weights $\boldsymbol{\theta} = \{\boldsymbol{\mathcal{W}}_l\}_{l=1}^L$. Given a loss function $\mathcal{L}$, the expected risk minimization can be formulated as:

$$\min_{\boldsymbol{\theta}} \quad \mathbb{E}_{X,Y} \mathcal{L}(f_{\boldsymbol{\theta}}(X), Y), \tag{3}$$

where $\mathbb{E}_X[\cdot]$ is the expectation over the distribution $\mathbb{P}(X)$.

4.1 Tensorized Neural Network Layers

We start by parameterizing the predictor using decomposed cores. To compress the most common layers involving linear transformations (including *fully-connected layers*[1] and *embedding layers*), following [Novikov *et al.*, 2015; Wang *et al.*, 2018], the weight matrix $\boldsymbol{W} \in \mathbb{R}^{O \times I}$ is reshaped to a high-order tensor $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{O_1 \times \cdots \times O_n \times I_1 \times \cdots \times I_m}$, where $O = \prod_{i=1}^n O_i$ and $I = \prod_{i=1}^m I_i$. Then $\boldsymbol{\mathcal{W}}$ can be represented by TR cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{m+n}$ of ranks $\{R_i\}_{i=0}^{m+n}$. An arbitrary input sample $\boldsymbol{x} \in \mathbb{R}^I$ is correspondingly reshaped to $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_m}$. Thus, the reshaped output $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{O_1 \times \cdots \times O_n}$ is calculated through contractions between the cores and $\boldsymbol{\mathcal{X}}$:

$$\boldsymbol{\mathcal{Y}}(j_1, \ldots, j_m) = \sum_{i_1, \ldots, i_n} \boldsymbol{\mathcal{W}}(i_1, \ldots, i_n, j_1, \ldots, j_m) \boldsymbol{\mathcal{X}}(i_1, \ldots, i_n), \tag{4}$$

where

$$\boldsymbol{\mathcal{W}}(i_1, \ldots, i_n, j_1, \ldots, j_m) = \sum_{r_1, \ldots, r_{m+n}} \boldsymbol{\mathcal{G}}_1(r_{m+n}, i_1, r_1) \cdots \boldsymbol{\mathcal{G}}_{m+n}(r_{m+n-1}, j_m, r_{m+n}). \tag{5}$$

Once $\boldsymbol{\mathcal{Y}}$ is obtained, it can be flattened to $\boldsymbol{y} \in \mathbb{R}^O$.

The original *convolutional layer* has the weight of a kernel tensor $\boldsymbol{\mathcal{K}} \in \mathbb{R}^{O \times I \times K \times K}$ and deals with the input $\boldsymbol{x}$ of shape $H \times W \times I$, where O, I, and K are the number of output channels, the number of input channels, and the kernel size, respectively. Similar to (4), $\boldsymbol{\mathcal{K}}$ and $\boldsymbol{\mathcal{X}}$ are reshaped to $\mathbb{R}^{O_1 \times \cdots \times O_m \times I_1 \times \cdots \times I_m \times K \times K}$ and $\mathbb{R}^{H \times W \times I_1 \times \cdots \times I_m}$, respectively, where $O = \prod_{i=1}^n O_i$ and $I = \prod_{i=1}^m I_i$. Then,

[1] Biases are omitted in this part to simplify notation.

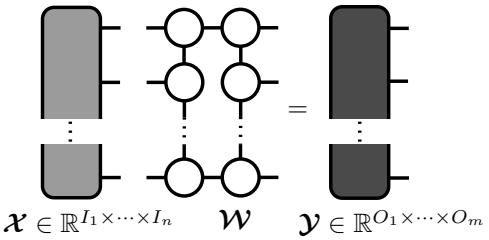

Figure 1: Tensor diagram of a TR linear layer. The output $\boldsymbol{\mathcal{Y}}$ is calculated through contractions between $\boldsymbol{\mathcal{X}}$ and tensor cores.

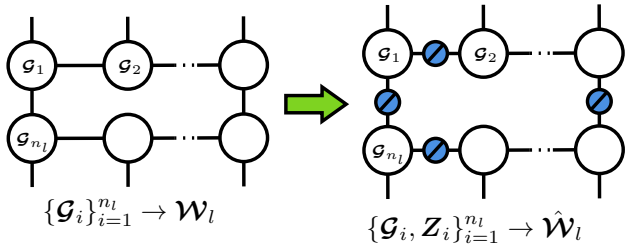

Figure 2: Schematic diagram of the gated tensor contraction for TR decomposition. The l-th layer parameterized with the original TR cores $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{n_l}$ is modified to be parameterized with $\{\boldsymbol{\mathcal{G}}_i, \boldsymbol{\mathcal{Z}}_i\}_{i=1}^{n_l}$.

$\boldsymbol{\mathcal{K}}$ is represented with TR cores $\{\boldsymbol{\mathcal{U}}_i\}_{i=1}^n \cup \{\boldsymbol{\mathcal{G}}\} \cup \{\boldsymbol{\mathcal{V}}_j\}_{j=1}^m$, where $\boldsymbol{\mathcal{U}}_i \in \mathbb{R}^{R_{i-1} \times I_i \times R_i}$, $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_n \times K \times K \times R_{n+1}}$, and $\boldsymbol{\mathcal{V}}_j \in \mathbb{R}^{R_{j+n} \times O_j \times R_{j+n+1}}$. Then, the original convolutional operation, formulated by

$$\boldsymbol{Y}(h, w, o) = \sum_{d_1, d_2=1}^{K} \sum_{i=1}^{I} \boldsymbol{\mathcal{X}}(h', w', i) \boldsymbol{\mathcal{K}}(d_1, d_2, i, o), \tag{6}$$

where $h = (h'-1)S + d_1 - P$ and $w = (w'-1)S + d_2 - P$, with $S \in \mathbb{N}$ being the stride, and $P \in \mathbb{N}$ being the padding size, can be reformulated by a three-step alternative:

$$\boldsymbol{\mathcal{P}}(h', w', r_0, r_n) = \sum_{\substack{i_1, \ldots, i_n, \\ r_1, \ldots, r_{n-1}}} \boldsymbol{\mathcal{X}}(h', w', i_1, \ldots, i_n) \prod_{j=1}^{n} \boldsymbol{\mathcal{U}}_j(r_{j-1}, i_j, r_j), \tag{7}$$

$$\boldsymbol{\mathcal{Q}}(h, w, r_0, r_{n+1}) = \sum_{d_1, d_2, r_n} \boldsymbol{\mathcal{P}}(h', w', r_0, r_n) \boldsymbol{\mathcal{G}}(r_n, d_1, d_2, r_{n+1}), \tag{8}$$

$$\boldsymbol{\mathcal{Y}}(h, w, o) = \sum_{\substack{r_0, r_{n+1}, \\ \ldots, r_{n+m}}} \boldsymbol{\mathcal{Q}}(h, w, r_0, r_{n+1}) \prod_{j=1}^{m} \boldsymbol{\mathcal{V}}_j(r_{n+j}, o_j, r_{n+j+1}). \tag{9}$$

Note that in (5), (7), and (9), TR ranks determine the number of parameters and the expressive power of the model, whose search space could be potentially large with a large number of cores. Setting all ranks equal could be over-simplified while manually tuning each of the ranks is impractical. Therefore, we consider an automatic rank selection method for tensorized neural networks.

4.2 Automatic Rank Selection via Probabilistic ℓ_0 Regularization

In this section, we propose an ℓ_0-based rank selection formulation through approximate Bernoulli gates, to remove the

rank redundancy without performance loss in (5), (7), and (9). Following the decomposition scheme in Sec. 4.1, $\boldsymbol{\theta}$ in (3) is parameterized with cores $\{\{\boldsymbol{\mathcal{G}}_i^{(l)}\}_{i=1}^{n_l}\}_{l=1}^{L}$, where n_l is the number of cores of the l-th layer. Based on (3), we define the following low-rank constrained optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \mathbb{E}_{X,Y}\mathcal{L}(f_{\boldsymbol{\theta}}(X), Y) \\ \text{s.t.} \quad & \text{rank}(\boldsymbol{\mathcal{G}}_i^{(l)}) \coloneqq R_i^{(l)} \leq Q, \quad i \in [n_l], l \in [L], \end{aligned} \tag{10}$$

where Q is some target rank.

Gated Tensor Contraction with ℓ_0 Regularization. Since (10) is unsolvable with gradient methods due to the non-differentiable nature of the rank function, one may replace the rank minimization with trace norm penalty [Tomioka *et al.*, 2010]. However, penalizing on the trace norm is SVD-based and thus computationally intractable in large-scale neural networks. To overcome the limitation, we introduce Bernoulli gates applied to slices of tensor cores to shrink their ranks and propose the gated tensor contraction. Specifically, we propose to insert a diagonal matrix $\boldsymbol{Z}_i \in \mathbb{R}^{R_i \times R_i}$ between each pair of adjacent factors $(\boldsymbol{\mathcal{G}}_i, \boldsymbol{\mathcal{G}}_{i+1})$ to be contracted, as shown in Fig. 2. Diagonal entries of $\boldsymbol{Z}_i$ are independent random variables and satisfy $\boldsymbol{Z}_i(j,j) \sim \text{Bern}(\pi_{ij})$. Then the contraction between $\boldsymbol{\mathcal{G}}_i$ and $\boldsymbol{\mathcal{G}}_{i+1}$ is replaced by:

$$\begin{aligned} &\{\boldsymbol{\mathcal{G}}_i \times_3 \boldsymbol{Z}_i \times_{3,1} \boldsymbol{\mathcal{G}}_{i+1}\}(q, a_1, a_2, p) \\ =&\sum_{j=1}^{R_i} \boldsymbol{\mathcal{G}}_i(q, a_1, j)\boldsymbol{Z}_i(j,j)\boldsymbol{\mathcal{G}}_{i+1}(j, a_2, p), \end{aligned} \tag{11}$$

where $\boldsymbol{Z}_i$ acts as gates on the ranks of cores $\boldsymbol{\mathcal{G}}_i$ and $\boldsymbol{\mathcal{G}}_{i+1}$. The learning problem of (10) is then transformed to an ℓ_0-regularized optimization problem:

$$\min_{\boldsymbol{\theta}'} \quad \mathbb{E}_{\boldsymbol{Z}}\mathbb{E}_{X,Y}\left[\mathcal{L}(f_{\boldsymbol{\theta}'}(X), Y) + \lambda \sum_{l=1}^{L}\sum_{i=1}^{n_l} \|\boldsymbol{Z}_i^{(l)}\|_0\right], \tag{12}$$

where f is parameterized by $\boldsymbol{\theta}' = \{\{\boldsymbol{\mathcal{G}}_i^{(l)}, \{\pi_{ij}^{(l)}\}_{j=1}^{R_i^{(l)}}\}_{i=1}^{n_l}\}_{l=1}^{L}$ and λ is the non-negative hyperparameter. Note that if $\boldsymbol{Z}_i(j,j) = 0$, the slices $\boldsymbol{\mathcal{G}}_i(:,:,j)$ and $\boldsymbol{\mathcal{G}}_{i+1}(j,:,:)$ can be zeroed out and pruned without changing the result of tensor contractions. Therefore, (12) can be seen as a structured pruning on tensorized neural networks that brings an actual reduction in both parameters and computations. A simple and conventional way is to relax ℓ_0 to ℓ_1. However, practically, introducing an ℓ_1 penalty into gradient methods does not result in sparse solutions [Ziyin and Wang, 2023] and requires post-training thresholding.

Approximate Bernoulli Gates for ℓ_0 Regularization. The optimization of (12) involves sampling from discrete distribution and suffers from high variance[2]. Inspired by stochastic gates [Yamada *et al.*, 2020], we propose to use the Gaussian-based Bernoulli continuous relaxation, which makes the rank selection and the end-to-end learning of a tensorized neural network stable and scalable. For simplicity of notations, we consider one layer with n tensor cores of the neural network and omit the index l. An approximate Bernoulli gate is defined by $\boldsymbol{Z}_i(j,j) \coloneqq z_{i,j} = \max(0, \min(1, \mu_{i,j} + \sigma\epsilon_{i,j}))$, where $\epsilon_{i,j} \sim \mathcal{N}(0,1)$ and σ is fixed during training. Thus, its gradient with respect to $\mu_{i,j}$ can be computed using standard backpropagation, and the expected ℓ_0 regularizer of $\boldsymbol{Z}_i$ becomes

$$\mathcal{R}(\{\mu_{i,j}\}_{j=1}^{R_i}) \coloneqq \mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}\|\boldsymbol{Z}_i\|_0 = \sum_{j=1}^{R_i} \Phi\left(\frac{\mu_{i,j}}{\sigma}\right), \tag{13}$$

where $\Phi(\cdot)$ is the standard Gaussian CDF. Given the training data $\{\boldsymbol{x}_k, \boldsymbol{y}_k\}_{k=1}^{K}$, the Monte Carlo (MC) approximation of the regularized empirical risk minimization of (12) is

$$\begin{aligned} \min_{\boldsymbol{\theta}'} \quad & \frac{1}{M}\sum_{m=1}^{M}\left(\frac{1}{K}\sum_{k=1}^{K}\mathcal{L}(f_{\{\boldsymbol{\mathcal{G}}_i, \boldsymbol{Z}_i^{[m]}\}_{i=1}^{n}}(\boldsymbol{x}_k), \boldsymbol{y}_k)\right) \\ & + \lambda\sum_{i=1}^{n}\sum_{j=1}^{R_i}\Phi\left(\frac{\mu_{i,j}}{\sigma}\right), \end{aligned} \tag{14}$$

where the set of variables is $\boldsymbol{\theta}' = \{\{\boldsymbol{\mathcal{G}}_i^{(l)}, \{\mu_{ij}^{(l)}\}_{j=1}^{R_i^{(l)}}\}_{i=1}^{n_l}\}_{l=1}^{L}$, $z_{i,j}^{[m]} = \max(0, \min(1, \mu_{i,j} + \sigma\epsilon_{i,j}^{[m]}))$ is the m-th MC sample, $\epsilon_{i,j}^{[m]} \sim \mathcal{N}(0,1)$, and M is the number of samples. The gradient of the objective of (14) w.r.t. $\mu_{i,j}$ is estimated as

$$\begin{aligned} &\nabla_{\mu_{i,j}}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,1)}[\mathcal{L}(f_{\boldsymbol{\theta}'}(X), Y) + \lambda\|\boldsymbol{Z}_i\|_0] \\ &\approx \frac{1}{M}\sum_{m=1}^{M}\frac{\partial\mathcal{L}}{\partial\hat{\boldsymbol{\mathcal{G}}}_i}\frac{\partial\hat{\boldsymbol{\mathcal{G}}}_i}{\partial z_{i,j}^{[m]}}\frac{\partial z_{i,j}^{[m]}}{\partial\mu_{i,j}} + \lambda\frac{\partial}{\partial\mu_{i,j}}\Phi\left(\frac{\mu_{i,j}}{\sigma}\right), \end{aligned} \tag{15}$$

where $\hat{\boldsymbol{\mathcal{G}}}_i = \boldsymbol{\mathcal{G}}_i \times_3 \boldsymbol{Z}_i$. The optimization incorporated with the ℓ_0 norm can be solved using ubiquitous stochastic gradient methods through the approximations. Compared to a vanilla tensorized neural network parameterized with tensor cores, our formulation introduces minor extra parameters with their number linear to the sum of tensor ranks.

Learned Ranks and Pruning. During the evaluation phase, the injected noises $\epsilon_{i,j}$ are removed for deterministic prediction, leading to $z_{i,j} = \max(0, \min(1, \mu_{i,j}))$. There is no need to round the learned gates after training since (1) most of the gates converge to 0 and 1, and (2) rounding of the gates scales some slices of the cores, resulting in output distortion and possible performance degradation. The trained gates $\{\boldsymbol{Z}_i\}_{i=1}^{n}$ in (14) indicate the learned ranks, because the slices $\boldsymbol{\mathcal{G}}_i(:,:,j)$ and $\boldsymbol{\mathcal{G}}_{i+1}(j,:,:)$ can be zeroed out and pruned if $\boldsymbol{Z}_i(j,j) = 0$. The learned rank $\hat{R}_i$ is defined as follows:

$$\hat{R}_i = \sum_{j=1}^{R_i}\mathbb{I}\left(\boldsymbol{Z}_i(j,j) > 0\right), \tag{16}$$

where $\mathbb{I}(\cdot)$ is the indicator function. It is worth noting that sparsity does not imply instant compression. To this end, we create a compact representation (denoted by $\boldsymbol{\mathcal{G}}_i^*$) of the core tensor $\hat{\boldsymbol{\mathcal{G}}}_i \in \mathbb{R}^{R_{i-1}\times d\times R_i}$ by excluding the pruned slices, leading to size reduced to $\hat{R}_{i-1} \times d \times \hat{R}_i$.

[2] An ablation study is done in the supplementary material. Optimizing discrete distribution is feasible but unstable and inefficient. Code and supplementary materials are available at https://github.com/ctxGou/Tensor-L0-Compression.

4.3 Initialization

Training a tensorized neural network from scratch requires proper weight initialization. We propose to initialize all μ_{ij}'s by some real numbers larger than 1, so all gates are one at the beginning, making the initialization methods on original tensorized neural networks applicable. For the sake of convergence and stable training, model weights are designed to maintain variance during forward propagation and during backward propagation. Specifically, consider a TR fully-connected layer or embedding layer with weights $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{n+m}$, and ranks $\{R_i\}_{i=1}^{n+m}$, the input and output are denoted by $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \ldots \times I_m}$ and $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{O_1 \times \ldots \times O_n}$, respectively. We make an assumption on $\boldsymbol{\mathcal{X}}$ and $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{n+m}$ that their entries are *i.i.d.*, and thus the following equation holds:

$$\sigma^2(\boldsymbol{\mathcal{Y}}) = \sigma^2(\boldsymbol{\mathcal{X}}) \prod_{i=1}^{n+m} \sigma^2(\boldsymbol{\mathcal{G}}_i) R_i I, \tag{17}$$

following Proposition 3.3 of [Pan *et al.*, 2022], where $I = \prod_{i=1}^{m} I_i$. Similar to Kaiming Initialization [2016], we consider the impact of ReLU activation and seek $\sigma^2(\boldsymbol{\mathcal{Y}}) = 2\sigma^2(\boldsymbol{\mathcal{X}})$ to avoid exponentially reducing or magnifying the magnitude of inputs. Additionally, we simply assume $\sigma(\boldsymbol{\mathcal{G}}_1) = \cdots = \sigma(\boldsymbol{\mathcal{G}}_{n+m})$. Therefore, (17) becomes

$$\sigma^2(\boldsymbol{\mathcal{G}}_i) = \sqrt[n+m]{\frac{2}{I \prod_{k=1}^{n+m} R_k}}, \quad i \in [n+m]. \tag{18}$$

In a similar way, the variance of tensorized convolutional layer TR cores with $\{\boldsymbol{\mathcal{G}}_i\}_{i=1}^{n+m+1}$ and ranks $\{R_i\}_{i=1}^{n+m+1}$ is derived as follows:

$$\sigma^2(\boldsymbol{\mathcal{G}}_i) = \sqrt[n+m+1]{\frac{2}{K^2 I \prod_{k=1}^{n+m+1} R_k}}, \quad i \in [n+m+1], \tag{19}$$

where K is the kernel size.

Tensor cores are initialized to uniform distribution following the variance suggested as (18) and (19). In some rare cases, we observe that the learned rank $\hat{R}_i$ drops to 0 with all gates turning to 0's, for which the whole layer is zeroed out and the data flows are destroyed. Such glitches can be circumvented by stopping updating the diagonal entries of $\boldsymbol{Z}_i$ once their values are lower than a pre-defined minimal rank.

5 Experiments

The proposed approach applies to various layers that can be parameterized with factorized tensor formats. Therefore, in this section, we evaluate the proposed method on various learning models for a wide range of applications. To evaluate the fundamental availability of our method on fully-connected layers, we initiate the experimental study with a basic **2-layer MLP** on the MNIST dataset [LeCun *et al.*, 1998], following the structure of [Hawkins and Zhang, 2021] (Sec. 5.1). Then we extend to convolutional layers based on **LeNet-5**, which is used in [Wang *et al.*, 2018] (Sec. 5.2). To show the robustness and scalability of the proposed approach on deeper networks, we further investigate the performance of **ResNet**-32/110 [He *et al.*, 2016; Wang *et al.*, 2018] on the CIFAR-10 dataset [Krizhevsky *et al.*, 2009] (Sec. 5.3), which is a particularly common setting in deep compression literature. Moreover, we evaluate our approach to embedding layers of **LSTMs** [Hochreiter and Schmidhuber, 1997] and **Transformers** [Vaswani *et al.*, 2017] on NLP tasks, including text sentiment classification and neural machine translation (Sec. 5.4).

Method	Accuracy (%)	Compression
Original	97.58±0.11	1×
TT	97.78±0.03	18.25×
TR	**98.47**±0.01	17.60×
LR-BTNN [2021]	97.8	137×
MARS (soft)	98.08±0.12	167.68±13.32×
MARS (hard)	97.53±0.41	196.55 ± 18.38×
Ours (TT)	97.50±0.27	**240.86**±60.46×
Ours (TR)	97.95±0.29	219.21±70.60×
	98.05±0.30	199.32±31.95×

Table 1: Comparison results of 2-layer MLP on MNIST. Results show the mean and standard deviation of five runs from different random seeds. The result of the Low-rank Bayesian Tensorized NN (LR-BTNN) is cited from [Hawkins and Zhang, 2021]. In this table and the following tables to appear, *Compression* refers to compression ratio, *i.e.*, the number of parameters of the original model divided by the number of parameters of the compact model.

Methods for Comparison. We compare our proposed method with tensor decomposition baselines with uniform ranks [Wang *et al.*, 2018], and cutting-edge rank selection approaches [Cheng *et al.*, 2020; Gao *et al.*, 2020; Li *et al.*, 2021; Kodryan *et al.*, 2023]. We implemented Tensor-Ring Net [Wang *et al.*, 2018] and conducted the experiments of MARS [Kodryan *et al.*, 2023] in Sec. 5.1 by their released code.

5.1 Tiny Example: 2-Layer MLP

In the first experiment, we evaluate our method on a toy 2-layer MLP on the MNIST image classification task using 60k/10k train/test samples. We follow the same settings of structures in [Hawkins and Zhang, 2021; Kodryan *et al.*, 2023], which train the network and select ranks in an end-to-end fashion. A tensorized neural network with two fully-connected layers [Novikov *et al.*, 2015] of 784×625 and 625×10 is parameterized with two TT-matrix representations of sizes $(7, 4, 7, 4) \times (5, 5, 5, 5)$ and $(25, 25) \times (5, 2)$, respectively. Initial ranks are set to 20, as recommended in [Kodryan *et al.*, 2023]. We also extend the 2-layer network to TR decomposition of shapes $(7, 4, 7, 4, 5, 5, 5, 5)$ and $(5, 5, 5, 2)$ for two layers, respectively. We set $\sigma = 1$ and the MC sampling number $M = 1$.

Results are shown in Table 1. We test our method and reproduce the results in [Kodryan *et al.*, 2023] (denoted as MARS) by running the provided codes, where *soft* and *hard* represent different hyperparameter choices. Both models are trained for 300 epochs using Adam with decaying learning rates from $1e-2$. The results indicate that with a comparable performance loss, ours learns a tensorized neural network of much lower ranks, suggesting the effectiveness of the ℓ_0-based regularization.

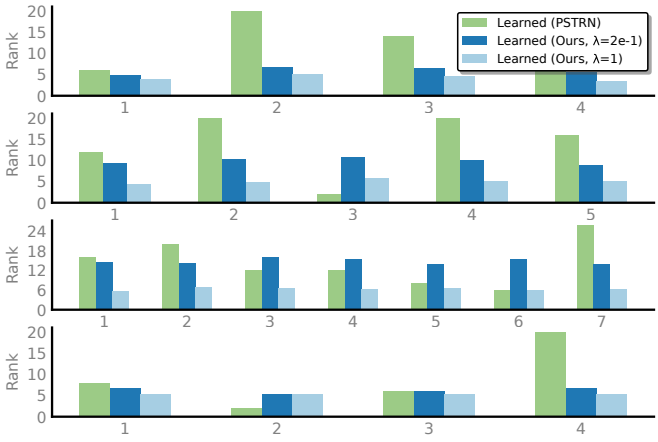

Figure 3: The learned ranks of ours and PSTRN for LeNet-5 on MNIST. *Ours* with darker blue and the one lighter correspond to the second and the first row of *Ours (TR)* in Table 2, respectively. The horizontal axis is the index of the rank of the cores. ↓ Layer 1-4

The two results of *Ours (TR)* are obtained by using different values of λ's ($\lambda = 0.003$ and $\lambda = 0.001$). The hyperparameter λ controls the strength of regularization on the sparsity of the diagonal entries of $\boldsymbol{Z}_i$. It helps to dramatically decrease the size of the search space, compared to manual tuning. As we will further see in the following experiments in Sec. 5.3, the search of λ can be further averted using a budget-aware scheme.

5.2 LeNet-5 on MNIST

We follow the network architecture described in [Wang *et al.*, 2018]. The model is a modified LeNet-5-like CNN, which incorporates two convolutional layers with 20 and 50 output channels, respectively, and two linear layers with 320 and 10 neurons, respectively, resulting in a cumulative parameter count of 429K in an uncompressed state. To embed TR decomposition to the network and present a fair comparison, we follow the decomposition settings of [Wang *et al.*, 2018]. We train the nets using Adam with decaying learning rates from $5e-3$. In this experiment, we set [3]$\sigma = 0.5$, $M = 1$, and initial ranks as 25.

Table 2 reports the prediction performance and compression rates of ours and other rank selection approaches. Our method achieves a competitive accuracy-compression trade-off. It is worth noting that PSTRN [Li *et al.*, 2021] utilizes a genetic algorithm search scheme, and therefore requires an evolutionary phase in which thousands of models are trained to search the appropriate sets of ranks. [Li *et al.*, 2021] reports that the search processes cost about 5 GPU days on NVIDIA V100s, which is much more time-consuming than our end-to-end rank selection approach.

Figure 3 further shows the learned ranks of the proposed method and PSTRN. PSTRN learns very small ranks in some cores but large ranks in other cores, while the learned ranks within a layer of ours are closer, resulting in a larger compression rate. Whether the optimal rank should be more even or vice versa, which could be relevant to the function of each tensor core in tensorized neural networks, is still an open problem, and we leave it for future research.

[3]A hyperparameter sensitivity analysis and experiments on KMNIST and FMNIST are presented in the supplementary material.

Method	Accuracy (%)	Compression
LeNet-5	**99.37**±0.06	1×
TRN ($r = 10$)	99.03±0.15	26×
TRN ($r = 15$)	99.16±0.06	11.71×
TRN ($r = 20$)	99.20±0.08	6.62×
Tucker [2018]	99.15	2×
MPO (TT) [2020]	99.17	20×
PSTRN [2021]	**99.4**	16.5×
MARS [2023]	99.0	10±0.8×
Ours (TR)	99.19±0.06	**61.39**±2.84×
	99.37±0.03	20.53±0.81×

Table 2: LeNet-5 on MNIST. Results show the mean and standard deviation of five runs. Compared methods include: Tensor Ring Nets (TRN) [Wang *et al.*, 2018], Tucker (reproduced in [Wang *et al.*, 2018]), Matrix Product Operator (MPO) [Gao *et al.*, 2020], Heuristic rank selection (PSTRN) [Li *et al.*, 2021], and Masked automatic rank selection (MARS) [Kodryan *et al.*, 2023].

5.3 ResNets on CIFAR-10

The combinations of ResNets [He *et al.*, 2016] and CIFAR-10 [Krizhevsky *et al.*, 2009] are popular settings in the literature on deep compression. CIFAR-10 [Krizhevsky *et al.*, 2009] consists of 50k images for training and 10k images for testing. We utilize SGD optimizer with a decaying learning rate initiating from 0.1. The models are trained for 200 epochs with batch size 128. The stochasticity introduced by the gates $\boldsymbol{Z}_i$'s makes the ℓ_0 approximation theoretically and empirically practical, but it also brings difficulties to the training of tensor core parameters due to the perturbation during the training phase. Thus, on the relative deep network ResNet, we set the variance of sampled noises $\sigma = 10^{-3}$, $M = 1$, and initial ranks as 15. The choice of λ directly determines the final compression rate, which brings inconvenience in practice since one may need to do cross-validation to meet a specific trade-off between compression and accuracy. We circumvent the problem by stopping updating the parameters of $\boldsymbol{Z}_i$'s and removing their stochasticity after the target compression ratio is reached. For a fair comparison, we follow [Wang *et al.*, 2018; Cheng *et al.*, 2020; Li *et al.*, 2021] by using the same TR decomposition (the same shapes and orders of core tensors) of ResNet-32 proposed in [Wang *et al.*, 2018]. We also conduct an experiment with ResNet-110 on CIFAR-10.

Results are shown in Tables 3 and 4. Our method outperforms the uniform rank baselines and rank selection approaches at a compression ratio of around 5×. Specifically, ours has a 0.5% accuracy advantage over Tensor-Ring Net baselines with a uniform rank of 10 at a similar compression rate, which again validates the superiority of the non-uniform rank scheme. While it is reported in [Li *et al.*, 2021] that the search processes for ResNet-32 of PSTRN cost about 3.2 GPU days on NVIDIA V100, ours requires about $\sim$ 0.11 GPU days on NVIDIA 2080Ti, which is $\sim$ 1.18× time of training a TR Net with a uniform rank of 15 and achieves

Method	Accuracy (%)	Compression
ResNet-32	**92.47**±0.17	1×
TRN ($r = 15$)	91.33±0.05	2.28×
TRN ($r = 10$)	90.45±0.10	4.95×
TT ($r = 13$) [2018]	88.3	4.8×
Tucker [2020]	87.7	5×
TR-RL [2020]	88.1	**15**×
PSTRN-M [2021]	90.6	5.1×
Ours (TR)	90.93±0.05	5×

Table 3: Results of ResNet-32 on CIFAR-10 image classification. Results with means and standard deviations are repeated three times, and others are cited from existing works. Unfortunately, Tensor-Ring Reinforcement Learning (TR-RL) [Cheng *et al.*, 2020] did not report the performance on $\sim 5\times$ compression rate.

Method	Accuracy (%)	Compression
ResNet-110	**93.13**±0.18	1×
TRN ($r = 20$)	92.31±0.18	1.36×
LR-BTNN [2021]	90.4	**7.4**×
MARS [2023]	91.1	5.5×
Ours (TR)	91.09±0.30	5.6×

Table 4: ResNet-110 on CIFAR-10 image classification. Results with means and standard deviations are obtained from three runs.

a 0.33% accuracy gain under $5\times$ compression rate. The search process of PSTRN is done with genetic algorithms and therefore becomes extremely time-consuming when the search space is large. In addition, PSTRN adopts a coarse-grained layer-wise rank selection scheme. In contrast, our method can search the rank of every tensor core with little extra overheads.

5.4 Embedding Layers

Word embedding is one of the core components of neural-based NLP models, which encapsulates linguistic information into vectors. However, as the vocabulary and embedding size increase, the number of its parameters can become large. For instance, neural machine translation models typically employ embeddings with a vocabulary size of ∼30k and an embedding size of ∼1024, totaling around 3×10^7 parameters.

A compact TT representation of embedded layers is proposed in [Hrinchuk *et al.*, 2020], replacing the original huge layers and drastically reducing the number of parameters. The implementation of this method is the same as the TT-format linear layer without bias and can be easily extended to TR. We first compare our probabilistic ℓ_0-based rank selection scheme with [Hrinchuk *et al.*, 2020; Kodryan *et al.*, 2023] on TT-embeddings in a text sentiment classification task, and then validate its effectiveness on TR-embeddings [Hrinchuk *et al.*, 2020] in a NMT task.

Text Sentiment Classification. We follow [Hrinchuk *et al.*, 2020; Kodryan *et al.*, 2023] by using the fine-grained five-class Stanford Sentiment Treebank (SST) dataset[4]. The dataset consists of 8.54k texts for training and 2.21k texts for testing. We take the most frequently appearing 17200 words and embed them into vectors of 256 dimensions using embedding layers of the original shape 17200×256. Embedding layers are followed by a bidirectional 2-layer LSTM with a hidden size of 128 and a dropout rate of 0.5.

[4]Details at https://nlp.stanford.edu/sentiment/.

Method	Accuracy	Comp. (Emb.)	Param.
Original Emb.	37.4%	1×	5.20M
TT ($r = 16$)	41.1%	182×	0.82M
MARS [2023]	**42.4**%	340×	0.81M
Ours (TT)	42.2%	**449**×	**0.80M**

Table 5: Text sentiment classification results, BiLSTM on SST-5. We report the classification accuracy, the compression rate in embedding layers, and the number of parameters.

Method	SacreBLEU	Comp. (Emb.)	Params.
Original Emb.	**32.36**	1×	65.13M
TR ($r = 32$)	32.14	21.33×	49.14M
TR ($r = 27$)	31.76	29.97×	48.91M
Ours (TR)	31.94	**30.78**×	**48.90M**

Table 6: Transformer on IWSLT'14 De-En. SacreBLEU [2018] (higher is better) with tokenizer MTEVAL-V14.PL is reported on the results of testing hypotheses vs. ground truth sentences. Note that the number of parameters excluding embedding layers is 48.35M.

In our experiment, initial ranks are set as 16. To compare with [Kodryan *et al.*, 2023; Hrinchuk *et al.*, 2020], we use TT-embeddings of the same shape $(10, 10, 12, 15) \times (4, 4, 4, 4)$. As shown in Table 5, our method achieves similar accuracy but a higher compression ratio than MARS and outperforms baselines in both metrics.

Neural Machine Translation. In this experiment, we use a standard Transformer [Vaswani *et al.*, 2017] and different embedding layers to conduct NMT on the IWSLT'14 De-En [Cettolo *et al.*, 2014] dataset, which consists of 170k German-English sentence pairs for training and 6.75k pairs for testing. We use a vocabulary of 32768 tokens via a joint source and target Byte-Pair Encoding (BPE). A full hyperparameter list can be found in the supplementary material.

Results are shown in Table 6. We use a TR-embedding of shape $(32, 32, 32) \times (8, 8, 8)$ for *TR* and *Ours* and search the ranks initiated from 32. On this challenging task, our method learns a more compact layer with a preferable BLEU score compared to the TR baselines with uniform ranks.

6 Conclusion

In this paper, we present an ℓ_0-regularized formulation for jointly learning model weights and decomposition ranks of tensorized neural networks in an end-to-end manner and propose to efficiently solve this by probabilistic approximations of ℓ_0 (quasi-)norm. Comprehensive experiments are done to validate that the proposed scheme learns more compact models. Our future work includes applying the method to other layers, *e.g.* tensorized self-attention, and combining existing orthogonal techniques, such as knowledge distillation [Wang *et al.*, 2021; Li *et al.*, 2022; Gu *et al.*, 2022].

Acknowledgments

Canh Hao Nguyen was supported by MEXT KAKENHI Grant Number 22K12150. Hiroshi Mamitsuka was supported by MEXT KAKENHI Grant Numbers 21H05027 and 22H03645.

References

[Arora *et al.*, 2018] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *International Conference on Machine Learning*, pages 244–253. PMLR, 2018.

[Ba and Caruana, 2014] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? *Advances in neural information processing systems*, 27, 2014.

[Cettolo *et al.*, 2014] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign. In *Proceedings of the 11th International Workshop on Spoken Language Translation: Evaluation Campaign*, pages 2–17, 2014.

[Cheng *et al.*, 2020] Zhiyu Cheng, Baopu Li, Yanwen Fan, and Yingze Bao. A novel rank selection scheme in tensor ring decomposition based on reinforcement learning for deep neural networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3292–3296. IEEE, 2020.

[Dai *et al.*, 2023] Wei Dai, Jicong Fan, Yiming Miao, and Kai Hwang. Deep learning model compression with rank reduction in tensor decomposition. *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[Dolgov *et al.*, 2021] Sergey Dolgov, Dante Kalise, and Karl K Kunisch. Tensor decomposition methods for high-dimensional hamilton–jacobi–bellman equations. *SIAM Journal on Scientific Computing*, 43(3):A1625–A1650, 2021.

[Gao *et al.*, 2020] Ze-Feng Gao, Song Cheng, Rong-Qiang He, Zhi-Yuan Xie, Hui-Hai Zhao, Zhong-Yi Lu, and Tao Xiang. Compressing deep neural networks by matrix product operators. *Physical Review Research*, 2(2):023300, 2020.

[Garipov *et al.*, 2016] Timur Garipov, Dmitry Podoprikhin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.

[Gu *et al.*, 2022] Jiaqi Gu, Ben Keller, Jean Kossaifi, Anima Anandkumar, Brucek Khailany, and David Z Pan. Heat: Hardware-efficient automatic tensor decomposition for transformer compression. *arXiv preprint arXiv:2211.16749*, 2022.

[Gusak *et al.*, 2019] Julia Gusak, Maksym Kholiavchenko, Evgeny Ponomarev, Larisa Markeeva, Philip Blagoveschensky, Andrzej Cichocki, and Ivan Oseledets. Automated multi-stage compression of neural networks. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 2501–2508. IEEE Computer Society, 2019.

[Han *et al.*, 2015] 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.

[Hashemizadeh *et al.*, 2020] Meraj Hashemizadeh, Michelle Liu, Jacob Miller, and Guillaume Rabusseau. Adaptive tensor learning with tensor networks. 2020.

[Hawkins and Zhang, 2021] Cole Hawkins and Zheng Zhang. Bayesian tensorized neural networks with automatic rank selection. *Neurocomputing*, 453:172–180, 2021.

[Hayashi *et al.*, 2019] Kohei Hayashi, Taiki Yamaguchi, Yohei Sugawara, and Shin-ichi Maeda. Exploring unexplored tensor network decompositions for convolutional neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[He *et al.*, 2016] 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.

[Hillar and Lim, 2013] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):1–39, 2013.

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

[Hochreiter and Schmidhuber, 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Hrinchuk *et al.*, 2020] Oleksii Hrinchuk, Valentin Khrulkov, Leyla Mirvakhabova, Elena Orlova, and Ivan Oseledets. Tensorized embedding layers. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4847–4860, 2020.

[Kodryan *et al.*, 2023] Maxim Kodryan, Dmitry Kropotov, and Dmitry Vetrov. Mars: Masked automatic ranks selection in tensor decompositions. In *International Conference on Artificial Intelligence and Statistics*, pages 3718–3732. PMLR, 2023.

[Krizhevsky *et al.*, 2009] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[Kwon *et al.*, 2023] Taehyung Kwon, Jihoon Ko, Jinhong Jung, and Kijung Shin. Tensorcodec: Compact lossy compression of tensors without strong data assumptions. *arXiv preprint arXiv:2309.10310*, 2023.

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

[LeCun *et al.*, 1998] 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.

[Li *et al.*, 2021] Nannan Li, Yu Pan, Yaran Chen, Zixiang Ding, Dongbin Zhao, and Zenglin Xu. Heuristic rank selection with progressively searching tensor ring network. *Complex & Intelligent Systems*, pages 1–15, 2021.

[Li *et al.*, 2022] Sunzhu Li, Peng Zhang, Guobing Gan, Xiuqing Lv, Benyou Wang, Junqiu Wei, and Xin Jiang. Hypoformer: Hybrid decomposition transformer for edge-friendly neural machine translation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 7056–7068, 2022.

[Louizos *et al.*, 2018] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through l_0 regularization. In *International Conference on Learning Representations*, 2018.

[Ma *et al.*, 2019] Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. A tensorized transformer for language modeling. *Advances in neural information processing systems*, 32, 2019.

[Maddison *et al.*, 2016] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *International Conference on Learning Representations*, 2016.

[Nakajima *et al.*, 2010] Shinichi Nakajima, Masashi Sugiyama, and Ryota Tomioka. Global analytic solution for variational bayesian matrix factorization. *Advances in Neural Information Processing Systems*, 23, 2010.

[Novikov *et al.*, 2015] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. *Advances in neural information processing systems*, 28, 2015.

[Orús, 2019] Román Orús. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1(9):538–550, 2019.

[Oseledets, 2011] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[Pan *et al.*, 2022] Yu Pan, Zeyong Su, Ao Liu, Wang Jingquan, Nannan Li, and Zenglin Xu. A unified weight initialization paradigm for tensorial convolutional neural networks. In *International Conference on Machine Learning*, pages 17238–17257. PMLR, 2022.

[Post, 2018] Matt Post. A call for clarity in reporting bleu scores. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, page 186. Association for Computational Linguistics, 2018.

[Rouard *et al.*, 2023] Simon Rouard, Francisco Massa, and Alexandre Défossez. Hybrid transformers for music source separation. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[Tibshirani, 1996] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[Tomioka *et al.*, 2010] Ryota Tomioka, Kohei Hayashi, and Hisashi Kashima. Estimation of low-rank tensors via convex optimization. *arXiv preprint arXiv:1010.0789*, 2010.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wang *et al.*, 2018] 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.

[Wang *et al.*, 2021] Benyou Wang, Yuxin Ren, Lifeng Shang, Xin Jiang, and Qun Liu. Exploring extreme parameter compression for pre-trained language models. In *International Conference on Learning Representations*, 2021.

[Wen *et al.*, 2016] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016.

[Yaguchi *et al.*, 2019] Atsushi Yaguchi, Taiji Suzuki, Shuhei Nitta, Yukinobu Sakata, and Akiyuki Tanizawa. Decomposable-net: Scalable low-rank compression for neural networks. *arXiv preprint arXiv:1910.13141*, 2019.

[Yamada *et al.*, 2020] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In *International Conference on Machine Learning*, pages 10648–10659. PMLR, 2020.

[Yang *et al.*, 2017] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *International Conference on Machine Learning*, pages 3891–3900. PMLR, 2017.

[Yin *et al.*, 2022] Miao Yin, Yang Sui, Wanzhao Yang, Xiao Zang, Yu Gong, and Bo Yuan. Hodec: Towards efficient high-order decomposed convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12299–12308, 2022.

[Zhao *et al.*, 2016] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016.

[Zhen *et al.*, 2022] Peining Zhen, Ziyang Gao, Tianshu Hou, Yuan Cheng, and Hai-Bao Chen. Deeply tensor compressed transformers for end-to-end object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4716–4724, 2022.

[Ziyin and Wang, 2023] Liu Ziyin and Zihao Wang. spred: Solving l1 penalty with sgd. In *International Conference on Machine Learning*, pages 43407–43422. PMLR, 2023.