Neurocomputing 453 (2021) 172–180

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Bayesian tensorized neural networks with automatic rank selection

Cole Hawkins [a], Zheng Zhang [b,*]

[a] Department of Mathematics, University of California, Santa Barbara, United States
[b] Department of Electrical and Computer Engineering, University of California, Santa Barbara, United States

ARTICLE INFO

Article history:
Received 12 November 2020
Revised 11 March 2021
Accepted 30 April 2021
Available online 4 May 2021
Communicated by Zidong Wang

Keywords:
Low-rank tensor
Tensor rank determination
Neural network compression

ABSTRACT

Tensor decomposition is an effective approach to compress over-parameterized neural networks and to enable their deployment on resource-constrained hardware platforms. However, directly applying tensor compression in the training process is a challenging task due to the difficulty of choosing a proper tensor rank. In order to address this challenge, this paper proposes a low-rank Bayesian tensorized neural network. Our Bayesian method performs automatic model compression via an adaptive tensor rank determination. We also present approaches for posterior density calculation and maximum a posteriori (MAP) estimation for the end-to-end training of our tensorized neural network. We provide experimental validation on a two-layer fully connected neural network, a 6-layer CNN and a 110-layer residual neural network where our work produces 7.4× to 137× more compact neural networks directly from the training while achieving high prediction accuracy.

1. Introduction

Despite their success in many engineering applications, deep neural networks are often over-parameterized, requiring extensive computing resources in their training and inference. In order to deploy neural networks on resource-constrained platforms such as IoT devices and smart phones, numerous techniques have been developed to build compact neural network models [1–4].

As a high-order generalization of matrix factorization, tensor decomposition has outperformed many existing compression algorithms by exploiting hidden low-rank structures in high dimensions. Given an over-parameterized neural network, such techniques have achieved state-of-the-art performance by tensorizing and compressing the fully connected layers or convolution kernels [4–7]. Tensorized neural networks achieve both memory footprint reduction and computational speedup [6], and recent work by [8] demonstrates their suitability for hardware acceleration. However, the train-then-compress framework cannot avoid the prohibitive computational costs in the training process.

This paper tries to directly train a low-rank tensorized neural network. A main challenge is to automatically determine the tensor rank (and thus the model complexity). Exactly determining a tensor rank is NP-hard [9]. Existing tensor rank surrogates such as tensor nuclear norms [10,11] are not suitable for regularizing the training process due to their high computational costs. Meanwhile, the unknown tensors describing weight matrices and convolution kernels are embedded in a nonlinear learning model. Although [6,12] presented some approaches to train tensorized neural networks, they assume that the tensor ranks are given, which is often infeasible in practice. Recently, some methods have been reported to control the rank of tensor trains in some optimization problems [13,14], but they often cause over-fitting because the tensor ranks can be arbitrarily increased in order to minimize the objective functions.

Contributions. Inspired by the recent Bayesian CP and Tucker tensor completion [15,16], we develop a novel low-rank Bayesian tensorized neural network. Our contribution is twofold. Firstly, we present a Bayesian model to compress the model parameters (e.g., weight matrices and convolution kernels) via tensor train decomposition [17]. We develop the first method for Bayesian tensor rank determination in nonlinear models such as neural networks. Our method employs a proper prior density to automatically determine the tensor ranks based on the given training data, which is beyond the capability of existing tensorized neural networks. Secondly, we develop training algorithms to estimate the full posterior density and the MAP point. To solve the large-scale Bayesian inference problem we approximate the posterior density by Stein variational gradient descent [18]. The proposed framework can generate much more compact neural networks. Our method can also provide uncertainty estimation, which is important for certain applications like decision making in autonomous driving [19] and robotics.

* Corresponding author.

https://doi.org/10.1016/j.neucom.2021.04.117
0925-2312/

1.1. Related work

Tensor Compression of Neural Networks. CP-format tensor decomposition [20,21] was first employed to compress the fully connected (FC) layers of pre-trained models by [4]. Following work then compressed both FC and convolutional layers by tensor train decomposition [5] and by Tucker decomposition [7], respectively. Work in [6,7] demonstrates that tensor-compression can reduce storage cost, inference and training times, and FLOP count. Further, tensor compressed layers can reduce the run-time through both tensor factor operations and hardware acceleration [6,8]. The results in [5,7] show that the FC layers can be compressed more significantly than the convolution layers. In order to avoid the expensive pre-training, [6,12] trained FC layers in low-rank tensor-train and Tucker formats, respectively, with the tensor ranks fixed in advance. In practice determining a proper tensor rank *a priori* is hard, and a bad rank estimation can result in low accuracy or high training cost. This challenge is the main motivation of our work.

Tensor Rank Determination. Two main approaches are used for rank determination in tensor completion: low-rank optimization and Bayesian inference. Optimization methods mainly rely on some generalization of the matrix nuclear norm [22] to tensors. The most popular approaches place a low-rank objective on tensor unfoldings [10,23,24], but the computation is expensive for high-order tensors. The Frobenius norms of CP factors are used as a regularization for 3-way tensor completion [25], but this does not generalize to high-order tensors either. Bayesian methods can directly infer the tensor rank in CP or Tucker tensor completion through low-rank priors [15,26,27,25,28]. In the CP and Tucker formats, the ranks of different tensor factors do not couple with each other. This is not true in the tensor-train format (see Section 3.2). Most importantly, in tensor completion the observed data is a linear mapping of a tensor, whereas the mapping is nonlinear in neural networks. Therefore, previous work using mean-field variational inference cannot generalize to tensorized neural networks.

Bayesian Neural Networks. Bayesian neural networks were developed to quantify the model uncertainty [29]. Both [30] and the work [29] explored sparsity-inducing shrinkage priors for relevance determination and weight removal in neural networks. Another popular prior for Bayesian neural network compression is the Minimum Description Length (MDL) framework for quantization [31]. MDL has seen modern implementations by [32,33], who also investigated Bayesian pruning. To our best knowledge, no Bayesian approaches have been reported to train a low-rank tensorized neural network. A main challenge of training a Bayesian neural network is the high computational cost caused by sampling-based posterior density estimations. We will address this issue by the recently developed Stein Variational Gradient Descent [18].

2. Background: Tensor Train (TT) Decomposition

Notation. We use bold lowercase letters (e.g., $\mathbf{a}$), uppercase letters (e.g., $\mathbf{A}$) and bold calligraphic letters (e.g., $\mathcal{A}$) to represent vectors, matrices and tensors, respectively. A tensor is a generalization of a matrix, or a multi-way data array [34]. An order-d tensor is a d-way data array $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_d}$, where I_k is the size of mode k. Given the integer $i_k \in [1, I_k]$ for each mode $k = 1 \cdots d$, an entry of the tensor $\mathcal{A}$ is denoted by $\mathcal{A}(i_1, \cdots, i_d)$. A **subtensor** is obtained by fixing a subset of the tensor indices. A **slice** is a two-dimensional section of a tensor, obtained by fixing all but two indices.

Definition 1. The **tensor-train** (TT) factorization [17] uses a compact multilinear format to express a tensor $\mathcal{A}$. Specifically, it expresses a d-way tensor $\mathcal{A}$ as a collection of matrix products:

$$\mathcal{A}(i_1, i_2, \ldots, i_d) = \mathcal{G}_1(:, i_1, :)\mathcal{G}_2(:, i_2, :) \ldots \mathcal{G}_d(:, i_d, :)$$

where $\mathcal{G}_k \in \mathbb{R}^{R_{k-1} \times I_k \times R_k}$ and $R_0 = R_d = 1$. The vector $\mathbf{R} = (R_0, R_1, R_2, \ldots, R_d)$ is the **TT-rank**. Each 3-way tensor $\mathcal{G}_i$ is a **TT core**.

The TT decomposition can also be applied to compress a matrix $\mathbf{W} \in \mathbb{R}^{M \times J}$. For the following definition we assume that the dimensions M and J can be factored as follows:

$$M = \prod_{k=1}^{d} M_k \quad J = \prod_{k=1}^{d} J_k.$$

Let μ, ν be the natural bijections from indices (m, j) of $\mathbf{W}$ to indices.

$(\mu_1(m), \nu_1(j), \ldots, \mu_d(m), \nu_d(j))$ of an order-$2d$ tensor $\mathcal{W}$. We identify the entries of $\mathbf{W}$ and $\mathcal{W}$ as:

$$\mathbf{W}(m, j) = \mathcal{W}(\mu_1(m), \nu_1(j), \ldots, \mu_d(m), \nu_d(j)). \tag{1}$$

Definition 2. The **TT-matrix** factorization expresses the matrix $\mathbf{W}$ as a series of matrix products:

$$\mathcal{W}(\mu_1(m), \nu_1(j), \ldots, \mu_d(m), \nu_d(j)) = \prod_{k=1}^{d} \mathcal{G}_k(:, \mu_k(m), \nu_k(j), :) \tag{2}$$

where each 4-way tensor $\mathcal{G}_k \in \mathbb{R}^{R_{k-1} \times M_k \times J_k \times R_k}$ is a TT core, and $R_0 = R_d = 1$. The vector $\mathbf{R} = (R_0, R_1, \ldots, R_d)$ is again the TT-rank.

For notational convenience we express a TT or TT-matrix parameterization of $\mathcal{W}$ as

$$\mathcal{W} = [[\mathcal{G}_1, \ldots, \mathcal{G}_d]]. \tag{3}$$

The TT-matrix format reduces the total number of parameters from $\prod_{k=1}^{d} M_k \prod_{k=1}^{d} J_k$ to $\sum_{k=1}^{d} R_{k-1} M_k J_k R_k$. This leads to massive parameter reductions when the TT-rank is low.

3. Proposed Bayesian Model

3.1. Low-Rank Bayesian Tensorized Neural Networks (LR-BTNN)

Let $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{N}$ be the training data, where $\mathbf{y}_i \in \mathbb{R}^S$ is an output label. We intend to train an L-layer tensorized neural network

$$\mathbf{y} \approx \mathbf{g}\left(\mathbf{x} | \left\{\mathcal{W}^{(l)}\right\}_{l=1}^{L}\right). \tag{4}$$

Here $\mathcal{W}^{(l)}$ is an unknown tensor describing the massive model parameters in the l-th layer. We consider two kinds of tensorized layers in this paper:

- **TT-FC Layer.** In a fully connected (FC) layer, a vector is mapped to a component-wise nonlinear activation function by a weight matrix $\mathbf{W}^{(l)}$. We tensorize $\mathbf{W}^{(l)}$ as $\mathcal{W}^{(l)}$ according to (1).
- **TT-Conv Layer.** A convolutional filter takes the form $\mathcal{K}^{(l)} \in \mathbb{R}^{t \times t \times C \times S}$ where $t \times t$ is the filter size, C and S are the numbers of input and output channels, respectively. The number of channels C and S are often larger than the filter size t, so we factorize $C = \prod_{i=1}^{d} c_i$, $S = \prod_{i=1}^{d} s_i$, reshape $\mathcal{K}^{(l)}$ into a $t^2 \times c_1 \times \ldots \times c_d \times s_1 \times \ldots \times s_d$ tensor, and denote the reshaped tensor as $\mathcal{W}^{(l)}$.

Our goal is to parameterize and compress each unknown $\mathcal{W}^{(l)}$ in TT-format in the training process. To simplify notations, we consider a single-layer neural network parametrized by a single tensor $\mathcal{W}$, but our results can be easily generalized to a general L-layer

network (see our result section). In order to build a Bayesian model, we assume the following likelihood function

$$p\left(\mathcal{D}|\{\mathcal{G}_k\}_{k=1}^d\right) = \prod_{i=1}^N p(\mathbf{y}_i, \mathbf{g}(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]])). \tag{5}$$

Here $\mathcal{G}_k \in \mathbb{R}^{R_{k-1}\times M_k\times J_k\times R_k}$ is an unknown TT core of $\mathcal{W}$, and the maximum TT-rank $\mathbf{R} = (1, R_1, \ldots, R_{d-1}, 1)$ sets an **upper bound** for the TT rank. The actual TT rank will be determined later. In this paper we focus on classification problems so we assume a multinomial likelihood. Let $y_{i,s}$ and g_s be the s-th component of $\mathbf{y}_i$ and $\mathbf{g}$, respectively, then we have

$$p(\mathbf{y}_i, \mathbf{g}(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]])) = \prod_{s=1}^S g_s(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]])^{y_{i,s}} \tag{6}$$

As a result, the negative log-likelihood is the standard cross-entropy loss

$$\mathcal{L}(\{\mathcal{G}_k\}) = -\sum_{i=1}^N\sum_{s=1}^S y_{i,s}\log g_s(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]]). \tag{7}$$

In order to infer the unknown TT cores $\{\mathcal{G}_k\}_{k=1}^d$ and to decide the actual ranks, we will further introduce some hidden variables $\left\{\boldsymbol{\lambda}^{(k)}\right\}_{k=1}^{d-1}$ to parameterize the prior density of $\{\mathcal{G}_k\}_{k=1}^d$ (which will be explained in Section 3.2). Let $\boldsymbol{\theta}$ denote all unknown variables

$$\boldsymbol{\theta} := \left\{\{\mathcal{G}_k\}_{k=1}^d, \left\{\boldsymbol{\lambda}^{(k)}\right\}_{k=1}^{d-1}\right\} \tag{8}$$

which is described with a prior density $p(\boldsymbol{\theta})$. Then we can build a tensorized neural network by estimating the MAP point or the full distribution of the following posterior density function:

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})} \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\mathcal{D},\boldsymbol{\theta}). \tag{9}$$

There exist two key challenges. Firstly, how shall we choose the prior density $p(\boldsymbol{\theta})$ to ensure desired model structures? Secondly, how can we efficiently solve the resulting large-scale Bayesian inference?.

3.2. Selection of prior density functions

In order to achieve automatic model compression in the training, the prior density function in (9) should be chosen such that: (1) the actual rank of $\mathcal{G}_k$ is very small; (2) manual rank tuning can be avoided.

Prior Density for $\mathcal{G}_k$. We specify the prior density on a TT-matrix core. The size of $\mathcal{G}_k$ is fixed as $R_{k-1}\times M_k\times J_k\times R_k$. In order to reduce the TT rank, we will enforce some rows and columns in the slice $\mathcal{G}_k(:, m_k, j_k, :)$ to zero. The main challenge is that the matrix products are coupled: the r_k-th column of $\mathcal{G}_{k-1}(:, m_{k-1}, j_{k-1}, :)$ and the r_k-th row of $\mathcal{G}_k(:, m_k, j_k, :)$ should simultaneously shrink to zero if a rank deficiency happens. Fig. 1 uses the slices of two adjacent 3-way TT cores to show this coupling in the TT decomposition.

In order to address the above challenge, we extend the sparsity-enforcing priors from [15] which were developed for CP and Tucker tensor completion but are not applicable to TT format. Specifically, we introduce the non-negative vector parameter $\boldsymbol{\lambda}^{(k)} = \left[\lambda_1^{(k)},\ldots,\lambda_{R_k}^{(k)}\right] \in \mathbb{R}^{R_k}$ to control the actual rank $\widehat{R}_k$ for each k with $1\leqslant k\leqslant d-1$. For each intermediate TT core $\mathcal{G}_k$, we place a normal prior on its entries:

$$p\left(\mathcal{G}_k \,|\, \boldsymbol{\lambda}^{(k-1)}, \boldsymbol{\lambda}^{(k)}\right) = \prod_{r_{k-1},m_k j_k,r_k} \mathcal{N}\left(\mathcal{G}_k(r_{k-1}, m_k, j_k, r_k)\,|\,0, \lambda_{r_{k-1}}^{(k-1)}\cdot\lambda_{r_k}^{(k)}\right) \tag{10}$$

where $2\leqslant k\leqslant d-1, 1\leqslant r_{k-1}\leqslant R_{k-1}, 1\leqslant r_k\leqslant R_k, 1\leqslant m_k\leqslant M_k$ and $1\leqslant j_k\leqslant J_k$.

Because $R_0 = R_d = 1$, the ranks of the first and last TT cores are controlled by only one vector. We place a similar normal prior on each:

$$\begin{aligned} p(\mathcal{G}_1 \,|\, \boldsymbol{\lambda}^{(1)}) &= \prod_{m_1 j_1, r_1} \mathcal{N}\left(\mathcal{G}_1(1, m_1, j_1, r_1)\,|\,0, \left(\lambda_{r_1}^{(1)}\right)^2\right) \\ p\left(\mathcal{G}_d \,|\, \boldsymbol{\lambda}^{(d-1)}\right) &= \prod_{r_{d-1},m_d j_d} \mathcal{N}\left(\mathcal{G}_d(r_{d-1}, m_d, j_d, 1)\,|\,0, \left(\lambda_{r_{d-1}}^{(d-1)}\right)^2\right). \end{aligned} \tag{11}$$

Here we use the squares $\left(\lambda_{r_1}^{(1)}\right)^2$ and $\left(\lambda_{r_{d-1}}^{(d-1)}\right)^2$ to ensure that the order of magnitude of the priors is consistent across all TT cores. To apply the same prior to the standard tensor train we remove the third index j_k of each TT core.

Prior Density for $\boldsymbol{\lambda}^{(k)}$. In order to avoid tuning $\left\{\boldsymbol{\lambda}^{(k)}\right\}$ and TT-ranks manually, we set each $\boldsymbol{\lambda}^{(k)}$ as a random vector and impose a gamma prior on its entries:

$$p\left(\boldsymbol{\lambda}^{(k)}\right) = \prod_{r_k=1}^{R_k} \mathrm{Ga}\left(\lambda_{r_k}^{(k)}|a_\lambda, b_\lambda\right). \tag{12}$$

The hyperparameters are set to $a_\lambda = 1, b_\lambda = 5$ to encourage sparsity. We chose the Gamma prior over other sparsity-inducing priors (i.e. Normal, Laplace, Horseshoe) due to better empirical compression ratios. We also experimented with shrinkage priors that were not coupled across adjacent tensor cores and found that they gave poor empirical compression.

Overall Prior for $\boldsymbol{\theta}$. Combining (10), (11) and (12), we have the overall prior density $p(\boldsymbol{\theta})$:

$$p(\boldsymbol{\theta}) = p(\mathcal{G}_1 \,|\, \boldsymbol{\lambda}^{(1)}) p\left(\mathcal{G}_d \,|\, \boldsymbol{\lambda}^{(d-1)}\right)\prod_{k=2}^{d-1} p\left(\mathcal{G}_k \,|\, \boldsymbol{\lambda}^{(k-1)}, \boldsymbol{\lambda}^{(k)}\right)\prod_{k=1}^{d-1} p\left(\boldsymbol{\lambda}^{(k)}\right). \tag{13}$$

Rank-Shrinkage Effect. We illustrate the rank-shrinkage of $\boldsymbol{\lambda}$ in our low-rank tensor prior with a close examination of the conditional prior density of the slices of the first core tensor $p\left(\mathcal{G}_1(:,:,:,r_k)|\lambda_{r_k}^{(1)}\right)$. Other tensor cores are similar. We observe that

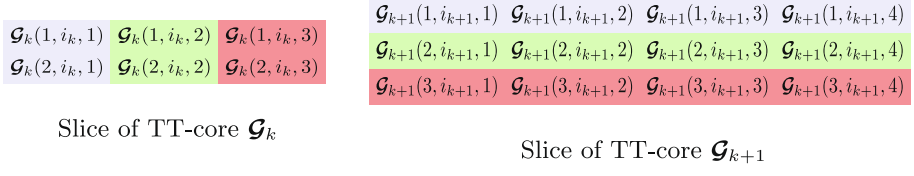


Fig. 1. Elements of 3-way $\mathcal{G}_k$ and $\mathcal{G}_{k+1}$. The elements controlled by the same entry of $\boldsymbol{\lambda}^{(k)}$ are marked with the same color. Here the upper bound of TT rank is set as $R_{k-1} = 2, R_k = 3, R_{k+1} = 4$.

$$
\begin{aligned}
p\left(\mathcal{G}_1(:,:,:,r_k)\,|\,\lambda_{r_k}^{(1)}\right) &= \prod_{m_1 j_1}\mathcal{N}\left(\mathcal{G}_1(1,m_1,j_1,r_k)\,|\,0,\left(\lambda_{r_k}^{(1)}\right)^2\right)\\
&= \prod_{m_1 j_1}\frac{1}{\lambda_{r_k}^{(1)}\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{\mathcal{G}_1(1,m_1 j_1 r_k)}{\lambda_{r_k}^{(1)}}\right)^2}.\\
&= \left(\frac{1}{\lambda_{r_k}^{(1)}\sqrt{2\pi}}\right)^{M_1 J_1} e^{-\frac{1}{2}\left(\frac{\|\mathcal{G}_1(:,:,:,r_k)\|_2}{\lambda_{r_k}^{(1)}}\right)^2}.
\end{aligned}
\tag{14}
$$

We will apply standard results for the Mahalanobis distance

$$
d^2 = \left(\frac{\|\mathcal{G}_1(:,:,:,r_k)\|_2}{\lambda_{r_k}^{(1)}}\right)^2 \tag{15}
$$

to demonstrate how shrinkage in λ_{r_k} leads to shrinkage in $\mathcal{G}_1(:,:,:,r_k)$. The Mahalanobis d distance follows a χ^2 distribution so

$$
p\left(d^2<\epsilon\right) = \frac{1}{2^{M_1 J_1}\Gamma(M_1 J_1/2)}\epsilon^{M_1 J_1/2-1}e^{-\frac{\epsilon}{2}}. \tag{16}
$$

Finally, we manipulate Eq. (16) by plugging in the value of d^2 from Eq. (15) to get

$$
p\left(\|\mathcal{G}_1(:,:,:,r_k)\|_2<\sqrt{\epsilon}\lambda_{r_k}^{(1)}\right) = \frac{1}{2^{M_1 J_1}\Gamma(M_1 J_1/2)}\epsilon^{M_1 J_1/2-1}e^{-\frac{\epsilon}{2}}. \tag{17}
$$

The right-hand side of Eq. (17) is constant and independent of $\lambda_{r_k}^{(1)}$. Therefore, as $\lambda_{r_k}^{(1)}\to 0$, with high probability the tensor slice norm $\|\mathcal{G}_1(:,:,:,r_k)\|_2\to 0$.

3.3. Complete probabilistic model

Now we are ready to obtain the full posterior density by combining (5), (9) and (13):

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathcal{D}) = \;& \frac{1}{p(\mathcal{D})}\prod_{i=1}^{N}p(\mathbf{y}_i,\mathbf{g}(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]]))p\left(\mathcal{G}_1\,|\,\boldsymbol{\lambda}^{(1)}\right)\times\\
& p\left(\mathcal{G}_d\,|\,\boldsymbol{\lambda}^{(d-1)}\right)\prod_{k=2}^{d-1}p\left(\mathcal{G}_k\,|\,\boldsymbol{\lambda}^{(k-1)},\boldsymbol{\lambda}^{(k)}\right)\prod_{k=1}^{d-1}p\left(\boldsymbol{\lambda}^{(k)}\right).
\end{aligned}
\tag{18}
$$

The associated probabilistic graphical model is shown in Fig. 2. The user-defined parameters a_λ and b_λ generate a Gamma distribution for $\boldsymbol{\lambda}^{(k)}$ which tunes the actual rank of each TT core $\mathcal{G}_k$ via a Gaussian distribution. The total number of parameters to be inferred for a single-layer tensorized neural network is

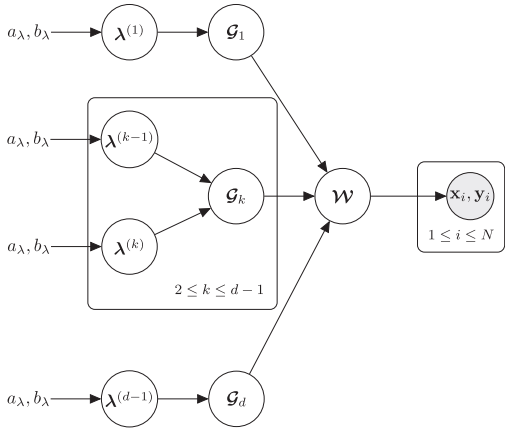


Fig. 2. Bayesian graphical model for a low-rank Bayesian tensorized neural network parametrized by a single low-rank tensor $\mathcal{W}$. There are N training samples.

$\sum_{k=1}^{d}M_k J_k R_{k-1}R_k + \sum_{k=1}^{d-1}R_k$. The extension to the an L-layer neural network is straightforward: one just needs to replicate the whole diagram except the training data by L times.

3.4. Automatic rank determination

The actual TT rank is determined by both the prior and training data. As shown in (10) and (11), each entry of $\boldsymbol{\lambda}^{(k)}$ directly controls one sub-tensor of $\mathcal{G}_k$ and one sub-tensor of $\mathcal{G}_{k+1}$. If the entry $\boldsymbol{\lambda}_{r_k}^{(k)}$ is large then elements of subtensors $\mathcal{G}_k(:,:,:,r_k)$ and $\mathcal{G}_{k+1}(r_k,:,:,:)$ can vary freely based on the training data. In contrast, if $\boldsymbol{\lambda}_{r_k}^{(k)}$ is close to zero, then the elements of $\mathcal{G}_k(:,:,:,r_k)$ and $\mathcal{G}_{k+1}(r_k,:,:,:)$ are more likely to vanish. Let $\bar{\boldsymbol{\lambda}}^{(k)}$ be the posterior mean of $\boldsymbol{\lambda}^{(k)}$ decided by both the prior and training data. Then the inferred TT-rank $\hat{\mathbf{R}} = \left[1,\widehat{R}_1,\widehat{R}_2,\ldots,\widehat{R}_{d-1},1\right]$ is estimated as the number of nonzero elements in $\bar{\boldsymbol{\lambda}}^{(k)}$:

$$
\widehat{R}_k = \mathrm{nnz}(\bar{\boldsymbol{\lambda}}^{(k)}) \text{ for } k=1,2,\ldots,d-1. \tag{19}
$$

In practice, an element of $\bar{\boldsymbol{\lambda}}^{(k)}$ is regarded as zero if it is below the threshold $1e-2$. Such an automatic rank tuning reduces the actual number of model parameters in a single layer to $\sum_{k=1}^{d}M_k J_k \widehat{R}_{k-1}\widehat{R}_k$.

4. Bayesian and MAP training

Now we describe how to train our low-rank Bayesian tensorized neural networks. We note here that prior work on Bayesian tensor rank determination [25,15,16,27] is only suitable for linear models, and cannot generalize to highly nonlinear Bayesian tensorized neural networks. We demonstrate how to overcome this difficulty.

4.1. Full Bayesian training

Existing Bayesian low-rank tensor methods either rely on mean-field variational inference or MCMC sampling. Mean-field approximations require that the tensor model is linear, and MCMC sampling is prohibitively expensive for large-scale or deep neural networks. Therefore, we employ the Stein variational gradient descent (SVGD) recently developed by [18] to approximate the posterior density $p(\boldsymbol{\theta}|\mathcal{D})$. SVGD combines the flexibility of MCMC with the speed of variational Bayesian inference to avoid both pitfalls.

The goal is to find a set of particles $\left\{\boldsymbol{\theta}^i\right\}_{i=1}^n$ such that $q(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n}k\left(\boldsymbol{\theta},\boldsymbol{\theta}^i\right)$ approximates the true posterior $p(\boldsymbol{\theta}|\mathcal{D})$. Here $k(\cdot,\cdot)$ is a positive definite kernel, and we use the radial basis function kernel here. The particles can be found by minimizing the Kullback–Leibler divergence between $q(\boldsymbol{\theta})$ and $p(\boldsymbol{\theta}|\mathcal{D})$. The optimal update $\phi(\cdot)$ is derived in [18] and takes the form

$$
\begin{aligned}
\boldsymbol{\theta}^k &\leftarrow \boldsymbol{\theta}^k + \epsilon\phi\left(\boldsymbol{\theta}^k\right)\\
\phi\left(\boldsymbol{\theta}^k\right) &= \frac{1}{n}\sum_{i=1}^{n}\left[k\left(\boldsymbol{\theta}^i,\boldsymbol{\theta}^k\right)\nabla_{\boldsymbol{\theta}^i}\log p\left(\boldsymbol{\theta}^i|\mathcal{D}\right) + \nabla_{\boldsymbol{\theta}^i}k\left(\boldsymbol{\theta}^i,\boldsymbol{\theta}^k\right)\right]
\end{aligned}
\tag{20}
$$

where ϵ is the step size. The gradient $\nabla_{\boldsymbol{\theta}}\log p(\boldsymbol{\theta}|\mathcal{D})$ is expressed as

$$
\begin{aligned}
\nabla_{\boldsymbol{\theta}}\log\, p(\boldsymbol{\theta}|\mathcal{D}) = \sum_{i=1}^{N}\sum_{s=1}^{S}y_{i,s}\frac{\nabla_{\boldsymbol{\theta}}[g_s(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]])]}{g_s(\mathbf{x}_i|[[\mathcal{G}_1,\ldots,\mathcal{G}_d]])} + \nabla_{\boldsymbol{\theta}}\\
\times\log p(\boldsymbol{\theta})
\end{aligned}
\tag{21}
$$

for classification. The first term is exactly the gradient of a maximum-likelihood tensorized training, and it is replaced by a stochastic gradient if N is large. The 2nd term caused by our low-rank prior is our only overhead over standard tensorized training

[6], and does not require forming the full tensor. Let $\mathbf{\Lambda}_k = \text{diag}\left(\boldsymbol{\lambda}^{(k-1)} \otimes \boldsymbol{\lambda}^{(k)}\right)$ be a diagonal matrix, and $\text{vec}(\mathcal{G}_k(:, m_k, j_k, :))$ be the column-major vectorization. The gradients of the log-prior are provided below:

$$\begin{aligned}
\frac{\partial \log p(\boldsymbol{\theta})}{\partial \text{vec}(\mathcal{G}_k(:,m_k,j_k,:))} &= -\mathbf{\Lambda}_k^{-1}\text{vec}(\mathcal{G}_k(:, m_k, j_k, :)) \\
\frac{\partial \log p(\boldsymbol{\theta})}{\partial \lambda_l^{(k)}} = & -\frac{1}{2}\sum_{m_{k+1} j_{k+1} r_{k+1}} \left\{ \frac{1}{\lambda_l^{(k)} \lambda_{r_{k+1}}^{(k+1)}} - \left(\frac{\mathcal{G}_{k+1}(l, m_{k+1} j_{k+1}, r_{k+1})}{\lambda_l^{(k)} \lambda_{r_{k+1}}^{(k+1)}} \right)^2 \right\} \\
& -\frac{1}{2}\sum_{r_{k-1} m_k j_k} \left\{ \frac{1}{\lambda_{r_{k-1}}^{(k-1)} \lambda_l^{(k)}} - \left(\frac{\mathcal{G}_k(r_{k-1}, m_k j_k, l)}{\lambda_{r_{k-1}}^{(k-1)} \lambda_l^{(k)}} \right)^2 \right\} \\
& + \frac{(a_\lambda - 1)}{\lambda_l^{(k)}} - b_\lambda.
\end{aligned} \tag{22}$$

4.2. MAP training

To obtain the MAP estimation we run stochastic gradient descent to minimize the negative log-posterior:

$$\begin{aligned}
-\log p(\boldsymbol{\theta}|\mathcal{D}) = & -\sum_{i=1}^{N}\sum_{s=1}^{S} y_{i,s} \log g_s(\mathbf{x}_i | [[\mathcal{G}_1, \ldots, \mathcal{G}_d]]) \\
& -\log p(\boldsymbol{\theta}) + \log p(\mathcal{D})
\end{aligned} \tag{23}$$

The local maximum achieved by MAP training provides a single non-Bayesian tensorized neural network that has been compressed by rank determination. This method is useful in order to quickly produce a single compressed model, but does not enable uncertainty quantification.

4.3. Initialization

Training deep neural networks requires careful initialization [35]. The same is true for tensorized neural networks [36]. A good empirically determined initialization distribution for the full tensor weights is $\mathcal{N}\left(0, \sqrt{\frac{2}{Q}}\right)$ where Q is the total number of parameters in the full tensor. In order to achieve variance $\sqrt{\frac{2}{Q}}$ for a low-rank TT or TT-matrix with ranks $R_1 = \ldots = R_{d-1} = R$, we initialize the TT core elements using a $\mathcal{N}(0, \sigma^2)$ with $\sigma^2 = \left(\frac{2}{Q}\right)^{1/2d} R^{1/d-1}$. This initialization provides a correction to the tensor core initialization described in [36] which contains an error.

5. Numerical experiments

5.1. Experimental setup

We validate our method using three network structures and three datasets. We use the Adam optimization algorithm [37] to initialize the first particle at the MAP point by minimizing Eq. (23) and then run 5000 iterations of SVGD. For all trainable weights except the low-rank tensors in our proposed model we apply a $\mathcal{N}(0, 100)$ prior which acts as a weak regularizer. We refer to our proposed low-rank Bayesian tensorized model as "**LR-BTNN**", a Bayesian tensorized neural network with a $\mathcal{N}(0, 100)$ prior on all weights and convolution kernels as "**BTNN**", and a Bayesian non-tensorized neural network with a $\mathcal{N}(0, 100)$ prior on all parameters as "**BNN**". We use the threshold 10^{-2} on all $\lambda_j^{(k)}$ to truncate a TT-rank. In all experiments the maximum rank R of the proposed LR-BTNN model is the same as the rank of the fixed-rank BTNN model.

- **Toy Model (2 FC Layers).** First we test on the MNIST and Fashion-MNIST datasets [38,39] using a network with two fully-connected layers. A similar tensorized neural network was studied in [12]. We compare the accuracy and rank determination ability of our approach as compared to the deterministic training approach from [12]. The first layer is size 784×625 with ReLU activation, and the second layer is size 625×10 with a softmax activation. Both layers contain a bias parameter. We convert the first layer into a TT-matrix with $(m_1, m_2, m_3, m_4) = (7, 4, 7, 4)$, $(j_1, j_2, j_3, j_4) = (5, 5, 5, 5)$ and the second layer into a TT-matrix with $(m_1, m_2) = (25, 25)$ and $(j_1, j_2) = (5, 2)$. Both layers have maximum TT-rank $R_1 = \ldots = R_{d-1} = 20$. We initialize by training for 100 epochs on the log-posterior. We use 50 particles to approximate the posterior density.
- **Baseline Six Layer CNN (4 Conv layers + 2 FC layers).** We test on the CIFAR-10 dataset [40] using a baseline tensorized convolutional model from [6]. This CNN model consists of (Conv-128, BN,ReLU), (Conv-128,BN,ReLU), max-pool 3×3, (Conv-256,BN, ReLU), (Conv-256,BN,ReLU), max-pool 5×5, fc-512, fc-10. All convolutions are 3×3 with stride 1. Following [5] we do not tensorize the first convolutional layer, which contains less than 1% of the parameters. We set the maximum TT-ranks of all layers to 30. We extend the original training 100 epoch training schedule from [5] to 120 epochs to account for the more complex log-posterior loss function. We use 20 particles for the SVGD fully Bayesian estimation.
- **ResNet-110 (109 Conv layers + 1 FC layer).** We further test the baseline Keras ResNetv1 structure [41,42] on the CIFAR-10 datasets. We do not tensorize the convolutions in the first ResBlock (first 36 layers) or the 1×1 convolutions. For all other convolutions we set the maximum TT-rank to 20. We use the standard 200 epoch training schedule to find the MAP point. As in the previous CNN experiment, we also use 20 particles for the SVGD fully Bayesian estimation.

5.2. Results

Table 1 shows the overall performances of our LR-BTNN with BNN and BTNN on the three examples. We evaluate test accuracy of the single-particle map estimate (MAP Acc.) and the uncertainty quality of the fully Bayesian SVGD model (test LL). From the table, we can compare the following performance metrics:

Model Size. The 3rd and 4th columns of Table 1 show the number of model parameters in the full posterior density estimations and in the MAP estimations, respectively. Because of the automatic tensor rank reduction, our LR-BTNN method generates much more compact neural networks. The compression ratio is very high when the networks have FC layers only (137× reduction over BNN on the MNIST example). The compression ratio becomes smaller when the network has more convolution layers (e.g., 27.3× on the CNN and 7.4× on ResNet-110). Our method outperforms [5,7] which compressed the convolution layers typically by 2× to 4×.

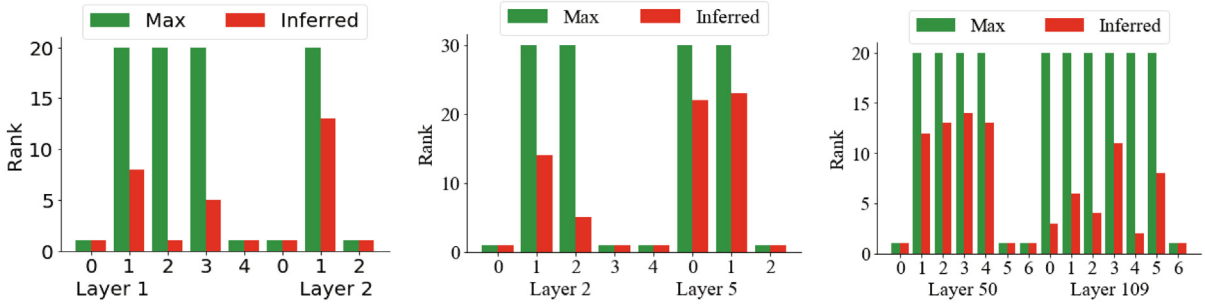
Prediction Accuracy. The 5th and 6th columns of Table 1 show the prediction accuracy of our fully Bayesian and MAP estimations, respectively. For the MAP estimation, the accuracy loss of our LR-BTNN model is very small, and our proposed model even achieves the best performance on the MNIST and Fashion-MNIST examples. For the fully Bayesian model, we measure the probabilistic model accuracy by computing the predictive log likelihood on held-out test data (denoted as "Test LL" in the table). Our LR-BTNN performs much better than other two methods on the MNIST and Fashion-MNIST examples, and all models are comparable on the CIFAR-10 tasks despite the fact that our model is much smaller.

Table 1
Results of a standard Bayesian neural network (BNN), a Bayesian tensorized neural network (BTNN) with Gaussian prior, and our low-rank Bayesian tensorized neural network (LR-BTNN).

Example	Model	SVGD Param #	MAP Param #	Test LL	MAP Acc.
Toy Model (2 FC, MNIST)	BNN	24,844,250	496,885	−0.193	97.6%
	BTNN	1,361,750	27,235	−0.188	97.7%
	LR-BTNN	**181,250 (137×)**	**3,625 (137×)**	**−0.106**	**97.8%**
Toy Model (2 FC, Fashion MNIST)	BNN	24,844,250	496,885	−0.619	87.1 %
	BTNN	1,361,750	27,235	−0.847	86.7%
	LR-BTNN	**642,750 (38.6×)**	**12,375 (40.1×)**	**−0.410**	**87.7%**
Baseline CNN (4 Conv+2 FC)	BNN	31,388,360	1,569,418	−0.497	**89.0%**
	BTNN	13,737,360	686,868	**−0.463**	88.8%
	LR-BTNN	**1,987,520 (15.8×)**	**99,376 (15.8×)**	-0.471	87.4%
ResNet-110 (109 Conv+1 FC)	BNN	34,855,240	1,742,762	**−0.506**	**92.6%**
	BTNN	12,895,800	644,790	−0.521	91.1%
	LR-BTNN	**4,733,028 (7.4×)**	**236,651 (7.4×)**	-0.515	90.4%

Automatic Rank Determination. The main advantage of our LR-BTNN method is its automatic rank determination and model compression in the traning process. Fig. 3 shows the rank determinations of some specific layers in all three examples. Fig. 4 shows the resulting layer-wise parameter reduction due to the automatic rank determination. The MNIST classification model has two overly parameterized FC layers, so the actual TT ranks are much lower than the maximum ranks. In fact, the TT-rank in Layer 1 reduces from $(1, 20, 20, 20, 1)$ to $(1, 8, 1, 5, 1)$, and the TT-rank in Layer 2 reduces from $(1, 20, 1)$ to $(1, 13, 1)$. This provides $8.5\times$ compression over the naive TNN and overall $138\times$ compression over the non-tensorized neural networks. A naive tensorization of the CNN model gives a compression ratio of $2.3\times$, and the rank determination gives a further parameter reduction of $6.9\times$, leading to an overall $15.8\times$ compression. Most layers in ResNet-110 are convolutions blocks with small numbers of filters, and there is only one small dense matrix. These facts make tensor compression less effective on this ResNet-110 example.

Uncertainty Quantification. Our LR-BTNN method is able to quantify the model uncertainty, which is critical for decision mak-

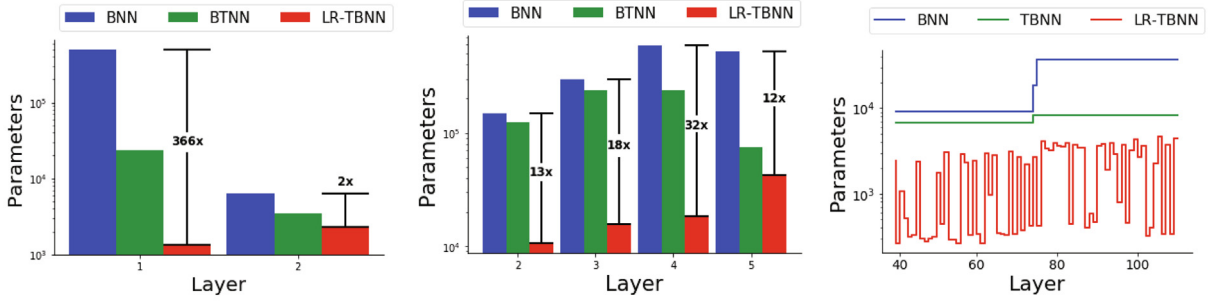


(a) Two FC layers in the MNIST example.

(b) Conv (left) and FC (right) layers in CNN.

(c) Two Conv layers in ResNet-110.

Fig. 3. Inferred TT-ranks at specified layers in the three examples.



(a) MNIST.

(b) CNN.

(c) ResNet-110.

Fig. 4. Compression of parameters at different layers due to the automatic rank determination.

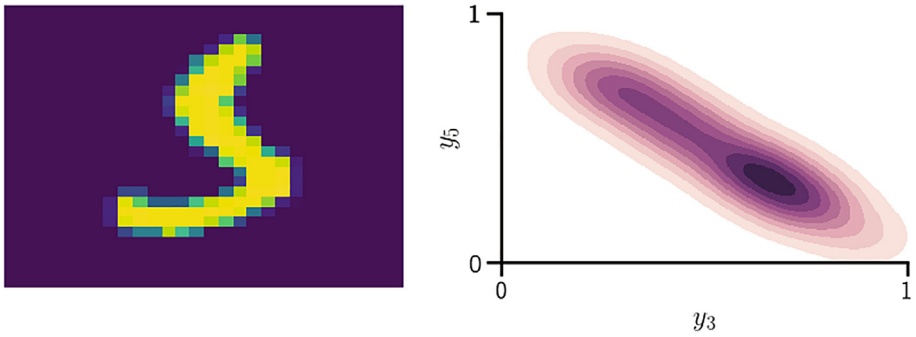


Fig. 5. Left: a challenging input image with true label 3. Right: the joint marginal density of softmax output 3 (x-axis) and softmax output 5 (y-axis).

Table 2
Rank parameter tuning experiment on the MNIST dataset. BTNN-*r* is a tensorized neural network with the same MNIST architecture as before and fixed tensor rank *r*.

Model	Param #	Accuracy
BTNN-5	3,160	96.89
BTNN-10	8,435	97.23
BTNN-15	16,460	97.44
BTNN-20	27,235	97.68
BTNN-25	40,760	97.66
BTNN-30	57,035	97.63
LR-BTNN	**3,625**	**97.78**

ing in uncertain environments. Specifically, based on the posterior density obtained from SVGD, we can easily estimate the probability density of an predicted output. As an example, Fig. 5 shows an image that is hard to classify. The predicted density shows that this image can be recognized as "3" with the highest probability, but it can also be recognized as "5" with a high probability. A possible pitfall of SVGD is that all particles collapse to the MAP point [43]. This does not occur for our model, as the MAP log-likelihood value consistently outperforms the SVGD log-likelihood value.

5.3. Cross-validation for rank selection

Cross-validation is a standard strategy for parameter tuning. However this approach is extremely time-consuming for selecting the tensor rank parameter because the rank parameter is discrete. Therefore combinatorial searches are required to select the best tensor ranks in the standard tensorized neural network training. In contrast, our model can determine the tensor rank in a single training run. We illustrate the effect of tuning the tensor rank in Table 2. We train the standard tensorized neural network (BTNN) on the MNIST task and vary the fixed tensor rank *r* of both layers to observe the parameter sensitivity. We observe that even after fine-tuning the tensor rank our model presents the most attractive parameter/accuracy tradeoff. This is because our model can automatically select non-uniform tensor ranks (i.e. (1,7,1,5,1)) without prohibitively expensive combinatorial search.

5.4. Particle number selection

For the MNIST model we found that existing SVGD approaches to non-tensorized training use $20-100$ particles for posterior approximation [18,44] so we selected an intermediate value of 50. In Fig. 6 we plot the particle/test log-likelihood tradeoff for each model (BNN,BTNN,LR-BTNN) on Fashion-MNIST. We observe that after approximately 40 particles the approximation quality is stable. For the larger CIFAR-10 experiments we selected the number of particles as 20 due to GPU computational constraints.

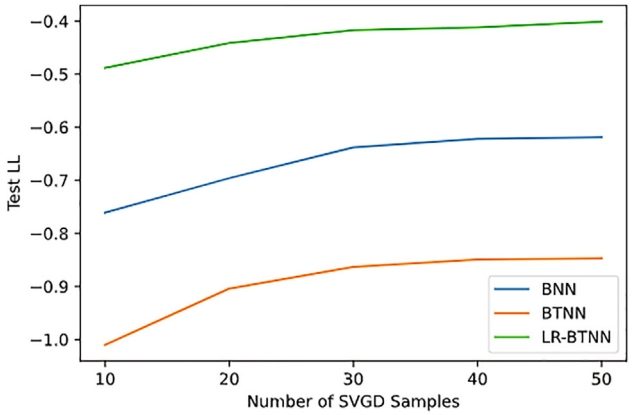


Fig. 6. Test log-likelihood sensitivity vs. number of particles used for the Fashion-MNIST task.

Table 3
Rank determination training overhead. All results are reported in seconds per epoch for MAP training. BTNN does not use rank determination. Our proposed LR-BTNN method uses rank determination.

Task	BTNN	LR-BTNN
Toy Model (2 FC, MNIST)	9.7	9.8
Baseline CNN (4 Conv+2 FC)	56.1	56.3
Resnet-110 (109 Conv+1 FC)	207.2	207.5

5.5. Rank determination overhead

To measure the overhead of our rank determination approach compared with the fixed-rank tensorized neural network training we measure the per-epoch timing overhead of the rank determination. The results are presented in Table 3. We observe that our proposed method (LR-BTNN) adds negligible overhead to standard fixed-rank training (BTNN).

6. Conclusion

We have proposed a low-rank Bayesian tensorized neural networks in the tensor train format. Our formulation provides an automatic rank determination and model compression in the end-to-end training. A Stein variational inference method has been employed to perform full Bayesian estimations, and the resulting model can predict output uncertainties. Our methods have shown excellent accuracy and model compression ratios on various neural

network structures with both fully connected and convolution layers for the MNIST, Fashion-MNIST, and CIFAR-10 data sets. Our method is shown to be more effective on FC layers than on convolution layers.

CRediT authorship contribution statement

Cole Hawkins: Investigation, Methodology, Software, Writing. **Zheng Zhang:** Conceptualization, Funding Acquisition, Supervision, Writing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

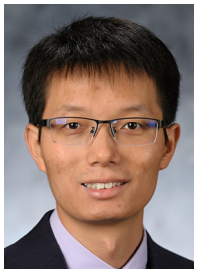
Acknowledgements

This work was supported by a UCSB Startup Grant, NSF Grant #1817037, and a hardware gift from NVIDIA.

References

[1] J.M. Alvarez, M. Salzmann, Compression-aware training of deep networks, in: Advances in Neural Information Processing Systems, 2017, pp. 856–867.
[2] S.J. Hanson, L.Y. Pratt, Comparing biases for minimal network construction with back-propagation, in: Advances in Neural Information Processing Systems, 1989, pp. 177–185. .
[3] Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in: Advances in Neural Information Processing Systems, 1990, pp. 598–605. .
[4] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, V. Lempitsky, Speeding-up convolutional neural networks using fine-tuned cp-decomposition, in: International Conference on Learning Representations, 2015. .
[5] T. Garipov, D. Podoprikhin, A. Novikov, D. Vetrov, Ultimate tensorization: compressing convolutional and FC layers alike, arXiv preprint arXiv:1611.03214. .
[6] A. Novikov, D. Podoprikhin, A. Osokin, D.P. Vetrov, Tensorizing neural networks, in: Advances in Neural Information Processing Systems, 2015, pp. 442–450.
[7] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, D. Shin, Compression of deep convolutional neural networks for fast and low power mobile applications, arXiv preprint arXiv:1511.06530. .
[8] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, B. Yuan, Tie: energy-efficient tensor train-based inference engine for deep neural network, in: Proc. Int. Symp. Computer Arch., 2019, pp. 264–278.
[9] C.J. Hillar, L.-H. Lim, Most tensor problems are NP-hard, Journal of the ACM (JACM) 60 (6) (2013) 45. .
[10] S. Gandy, B. Recht, I. Yamada, Tensor completion and low-n-rank tensor recovery via convex optimization, Inverse Problems 27 (2). .
[11] C. Lu, J. Feng, W. Liu, Z. Lin, S. Yan, et al., Tensor robust principal component analysis with a new tensor nuclear norm, IEEE Transactions on Pattern Analysis and Machine Intelligence. .
[12] G.G. Calvi, A. Moniri, M. Mahfouz, Z. Yu, Q. Zhao, D.P. Mandic, Tucker tensor layer in fully connected neural networks, arXiv preprint arXiv:1903.06133. .
[13] S.V. Dolgov, D.V. Savostyanov, Alternating minimal energy methods for linear systems in higher dimensions, SIAM Journal on Scientific Computing 36 (5) (2014) A2248–A2271.
[14] S. Holtz, T. Rohwedder, R. Schneider, The alternating linear scheme for tensor optimization in the tensor train format, SIAM Journal on Scientific Computing 34 (2) (2012) A683–A713.
[15] Q. Zhao, L. Zhang, A. Cichocki, Bayesian CP factorization of incomplete tensors with automatic rank determination, IEEE Transactions on Pattern Analysis and Machine Intelligence 37 (9) (2015) 1751–1763.
[16] Q. Zhao, G. Zhou, L. Zhang, A. Cichocki, S.-I. Amari, Bayesian robust tensor factorization for incomplete multiway data, IEEE Transactions on Neural Networks and Learning Systems 27 (4) (2016) 736–748.
[17] I.V. Oseledets, Tensor-train decomposition, SIAM Journal of Scientific Computing 33 (5) (2011) 2295–2317.
[18] Q. Liu, D. Wang, Stein variational gradient descent: A general purpose Bayesian inference algorithm, in: Proc. Advances In Neural Information Processing Systems, 2016, pp. 2378–2386. .
[19] A. Kendall, Y. Gal, What uncertainties do we need in Bayesian deep learning for computer vision?, in: Advances in Neural Information Processing Systems, 2017, pp. 5574–5584. .
[20] J.D. Carroll, J.-J. Chang, Analysis of individual differences in multidimensional scaling via an N-way generalization of Eckart-Young decomposition, Psychometrika 35 (3) (1970) 283–319.
[21] R.A. Harshman, M.E. Lundy, et al., PARAFAC: Parallel factor analysis, Computational Statistics and Data Analysis 18 (1) (1994) 39–72.
[22] B. Recht, M. Fazel, P.A. Parrilo, Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization, SIAM Review 52 (3) (2010) 471–501.
[23] J. Liu, P. Musialski, P. Wonka, J. Ye, Tensor completion for estimating missing values in visual data, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (1) (2013) 208–220.
[24] M. Imaizumi, T. Maehara, K. Hayashi, On tensor train rank minimization: Statistical efficiency and scalable algorithm, in: Advances in Neural Information Processing Systems, 2017, pp. 3930–3939.
[25] J.A. Bazerque, G. Mateos, G.B. Giannakis, Rank regularization and Bayesian inference for tensor completion and extrapolation, IEEE Transactions on Signal Processing 61 (22) (2013) 5689–5703.
[26] P. Rai, Y. Wang, S. Guo, G. Chen, D. Dunson, L. Carin, Scalable Bayesian low-rank decomposition of incomplete multiway tensors, in: International Conference on Machine Learning, 2014. .
[27] R. Guhaniyogi, S. Qamar, D.B. Dunson, Bayesian tensor regression, The Journal of Machine Learning Research 18 (1) (2017) 2733–2763.
[28] C. Hawkins, Z. Zhang, Robust factorization and completion of streaming tensor data via variational Bayesian inference, arXiv preprint arXiv:1809.01265. .
[29] R.M. Neal, Bayesian learning for neural networks, Ph.D. thesis, Citeseer, 1995. .
[30] D.J. MacKay, Bayesian methods for adaptive models, Ph.D. thesis, California Institute of Technology, 1992..
[31] G. Hinton, D. Van Camp, Keeping neural networks simple by minimizing the description length of the weights, in: Proc. ACM Conf. on Computational Learning Theory, Citeseer, 1993. .
[32] K. Ullrich, E. Meeds, M. Welling, Soft weight-sharing for neural network compression, in: International Conference on Learning Representations, 2017. .
[33] C. Louizos, K. Ullrich, M. Welling, Bayesian compression for deep learning, in: Advances in Neural Information Processing Systems, 2017, pp. 3288–3298.
[34] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Review 51 (3) (2009) 455–500.
[35] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Proc. International Conference on Artificial Intelligence and Statistics, 2010, pp. 249–256. .
[36] W. Wang, Y. Sun, B. Eriksson, W. Wang, V. Aggarwal, Wide compression: Tensor ring nets, in: Proc. IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 9329–9338. .
[37] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980. .
[38] Y. LeCun, The MNIST database of handwritten digits, http://yann. lecun. com/exdb/mnist/. .
[39] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747. .
[40] A. Krizhevsky, V. Nair, G. Hinton, The CIFAR-10 dataset, online: http://www.cs.toronto.edu/kriz/cifar.html. .
[41] F. Chollet, et al., Keras, URL:https://github.com/fchollet/keras (2015). .
[42] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[43] D. Wang, Z. Zeng, Q. Liu, Stein variational message passing for continuous graphical models, in: International Conference on Machine Learning, 2018, pp. 5206–5214. .
[44] M. Ye, T. Ren, Q. Liu, Stein self-repulsive dynamics: Benefits from past samples, in: H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, H. Lin (Eds.), Advances in Neural Information Processing Systems, vol. 33, Curran Associates Inc, 2020, pp. 241–252. URL: https://proceedings.neurips.cc/paper/2020/file/023d0a5671efd29e80b4deef8262e297-Paper.pdf. .

Cole Hawkins is a Ph.D student in the Mathematics department at University of California Santa Barbara (UCSB). His research interests include Bayesian inference and low-rank tensor computation. His primary application area is the design of compact tensorized neural networks for edge device machine learning.



Zheng Zhang received his Ph.D degree in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology (MIT), Cambridge, MA, in 2015. He is an Assistant Professor of Electrical and Computer Engineering with the University of California at Santa Barbara (UCSB), CA. His research interests include uncertainty quantification with applications to the design automation of multi-domain systems (e.g., nano-scale electronics, integrated photonics, and autonomous systems), and tensor computational methods for highdimensional data analytics. His industrial experiences include Coventor Inc. and Maxim-IC; academic visiting experiences include UC San Diego, Brown University and Politechnico di Milano; government lab experiences include Argonne National Labs. Dr. Zhang received the Best Paper Award of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems in 2014, the Best Paper Award of IEEE Transactions on Components, Packaging and Manufacturing Technology in 2018, two Best Paper Awards (IEEE EPEPS 2018 and IEEE SPI 2016) and three additional Best Paper Nominations (CICC 2014, ICCAD 2011 and ASP-DAC 2011) at international conferences. His PhD dissertation was recognized by the ACM SIGDA Outstanding Ph.D Dissertation Award in Electronic Design Automation in 2016, and by the Doctoral Dissertation Seminar Award (i.e., Best Thesis Award) from the Microsystems Technology Laboratory of MIT in 2015. He was a recipient of the Li Ka-Shing Prize from the University of Hong Kong in 2011.