SIAM J. MATH. DATA SCI.
Vol. 4, No. 1, pp. 46–71

Towards Compact Neural Networks via End-to-End Training: A Bayesian Tensor Approach with Automatic Rank Determination[*]

Cole Hawkins[†], Xing Liu[‡], and Zheng Zhang[§]

Abstract. Post-training model compression can reduce the inference costs of deep neural networks, but uncompressed training still consumes enormous hardware resources and energy. To enable low-energy training on edge devices, it is highly desirable to directly train a compact neural network from scratch with a low memory cost. Low-rank tensor decomposition is an effective approach to reduce the memory and computing costs of large neural networks. However, directly training low-rank tensorized neural networks is a very challenging task because it is hard to determine a proper tensor rank a priori, and the tensor rank controls both model complexity and accuracy. This paper presents a novel end-to-end framework for low-rank tensorized training. We first develop a Bayesian model that supports various low-rank tensor formats (e.g., CANDECOMP/PARAFAC, Tucker, tensor-train, and tensor-train matrix) and reduces neural network parameters with automatic rank determination during training. Then we develop a customized Bayesian solver to train large-scale tensorized neural networks. Our training methods shows orders-of-magnitude parameter reduction and little accuracy loss (or even better accuracy) in the experiments. On a very large deep learning recommendation system with over 4.2×10^9 model parameters, our method can reduce the parameter number to 1.6×10^5 automatically in the training process (i.e., by 2.6×10^4 times) while achieving almost the same accuracy. Code is available at https://github.com/colehawkins/bayesian-tensor-rank-determination.

Key words. compressed training, low-rank tensor, tensor rank determination, compact neural networks

AMS subject classifications. 68T01, 68T10

DOI. 10.1137/21M1391444

1. Introduction.

Despite their success in many applications, deep neural networks are often overparameterized, requiring extensive computing resources in their training and inference. For instance, the VGG-19 network requires 500M memory [44] for image recognition, and realistic deep learning recommendation model (DLRM) [38] has billions of parameters. It has been a common practice to reduce the size of neural networks before deploying them in various scenarios ranging from cloud services to embedded systems to mobile applications. To reduce hardware cost, numerous techniques have been developed to build *compact* models [1, 16, 34] after training. Representative approaches include pruning [34, 40], quantization [15, 58], knowledge distillation [21], and low-rank factorization [27, 33, 43, 52]. Among

[*] Received by the editors January 12, 2021; accepted for publication (in revised form) September 23, 2021; published electronically January 13, 2022.
https://doi.org/10.1137/21M1391444
Funding: This work is supported by a Facebook Research Award. The first and third authors are also supported by NSF CCF-1817037.

[†] Department of Mathematics, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (colehawkins@math.ucsb.edu).
[‡] Facebook AI Systems Hardware/Software Co-design, Menlo Park, CA 94025 USA (xingl@fb.com).
[§] Department of Electrical and Computer Engineering, University of California at Santa Barbara, Santa Barbara, CA 93106 USA (zhengzhang@ece.ucsb.edu).

these techniques, low-rank tensor compression [6, 9, 20, 27, 33, 36] has achieved possibly the most significant compression, leading to promising reduction of FLOPS and hardware cost [7, 27]. The recent progress of algorithm/hardware co-design [7, 54] of tensor operations can further reduce the run-time and boost the energy efficiency of tensorized models on edge devices (e.g., on field-programmable gate arrays (FPGAs) and application-specific circuits). While post-training compression techniques can reduce the cost of deploying a deep neural network, they cannot reduce the training cost.

Training consumes far more money, run-time, energy, and hardware resources than inference [45]. Meanwhile, the increasing concerns about data privacy have become a driving force for training on resource-constrained edge devices [48]. The high costs and hardware constraints associated with neural network training motivate us to ask the following question: *"Is it possible to train a compact neural network model from scratch?"* Both computing and hardware costs may be significantly reduced on various platforms if we can avoid the full-size uncompressed training. While pruning techniques can also be used in training [40, 51], they do not necessarily reduce the number of training variables. Low-precision arithmetic [13, 24, 31, 46] can reduce the cost per parameter during training and inference, but the memory cost reduction is limited to a single order of magnitude even in ultralow-precision 4-bit training [46].

1.1. Contributions.

This paper will present a rank-adaptive end-to-end tensorized training method to generate ultracompact neural networks directly from scratch. As shown in Figure 1(a), our method avoids the expensive full-size training in contrast with existing post-training tensor compression methods [6, 9, 27, 33]. Our method can reduce the training and inference variables by several orders of magnitude, and may achieve further reductions if combined with low-precision numerical operations [13, 24, 31, 46]. This work can make a great practical impact: it may enable energy-efficient training of medium- or large-size neural networks on edge devices (e.g., embedded graphics processing units and FPGA), which is impossible to achieve at this moment with existing training methods. Some recent works have studied low-rank tensorized training [3, 26, 41], but they fix the tensor ranks before training.

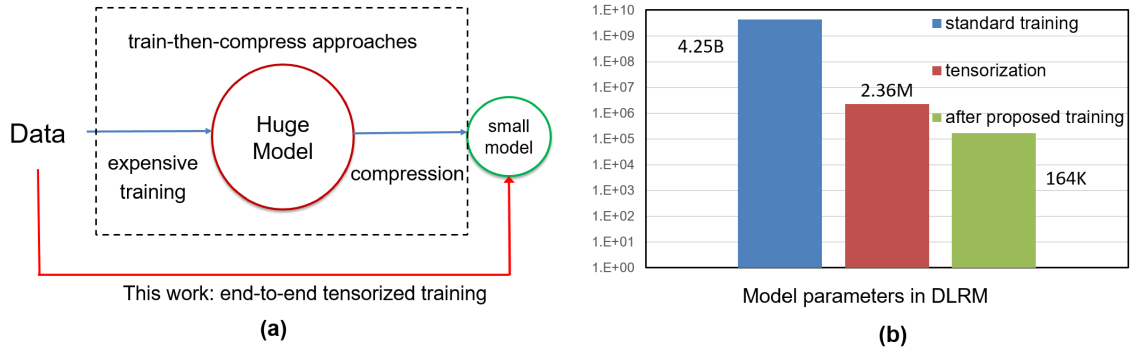


Figure 1. (a) *Key idea of this work. Conventional train-then-compress approaches have high training costs. In constrast, the proposed end-to-end tensorized training can reduce the training variables significantly and directly produce ultracompact neural networks.* (b) *Effectiveness of this approach on a realistic DLRM benchmark. Standard methods train* 4.25 *billion variables. Our proposed method only trains* 2.36 *million variables, which are further reduced to* 164*K in the training process due to the automatic tensor rank determination.*

It is hard to decide a proper tensor rank parameter a priori in practice; therefore, one often has to perform extensive combinatorial searches and many training runs until a good rank parameter is found.

We make the following contributions to achieve efficient one-shot tensorized training:

- **A general-purpose rank-adaptive Bayesian tensorized model.** The training cost and model performance are controlled by tensor ranks, which are unknown a priori. In order to avoid expensive manual search for tensor ranks required by recent works [3, 26, 41], we develop a novel Bayesian model to determine both tensor ranks and factors automatically. Existing tensor-based modeling methods are problem-specific and focus on a single tensor format [18, 19, 55, 56]. In contrast our work includes all four low-rank tensor formats in common use (CANDECOMP/PARAFAC (CP), Tucker, tensor-train, and tensor-train matrix) and makes general advances in low-rank tensor-based modeling. This paper focuses on neural networks, but our method can easily be applied to other tensor problems (e.g., tensor completion, tensor regression, and multitask tensor learning).
- **A scalable stochastic variational inference Bayesian solver for the proposed tensorized neural networks.** Training Bayesian tensorized neural networks is expensive, and existing approaches incur high memory and compute requirements. This is because particle-based Bayesian methods require multiple model copies and multiple forward propagations for every training and inference step [19]. Existing mean-field Bayesian tensor completion solvers [18, 55, 56] do not work for tensorized neural networks because of the highly nonlinear forward propagation model in our case. In this work we improve the approximate Bayesian inference method [22]. Specifically, we observe that directly employing the solver in [22] causes large gradient variance in our tensorized model. Therefore, we simplify the posterior density of some rank-controlling hyperparameters and develop an analytical/numerical hybrid approach for the solution update. This customized Bayesian solver infers the unknown tensor factors and tensor ranks of realistic neural networks in a single training run, enabling training and quantifying the uncertainty of extremely large-scale deep learning models that are beyond the capability of existing Bayesian solvers.
- **Extensive numerical validations.** We test our algorithms on four benchmarks with model parameters ranging from 4×10^5 to 4.2×10^9. Our method can reduce the training variables by several orders of magnitude with little or even no loss of accuracy. For instance, our method achieves $26,000\times$ parameter reduction when training a large-scale DLRM as shown in Figure 1(b). We also compare our methods with existing tensorized neural network methods [3, 9, 26, 41] including post-training compression and fixed-rank tensorized training, which clearly demonstrates the advantage of our rank-adaptive training method in terms of variable reduction and model accuracy.

To the best of our knowledge, this work is the first end-to-end Bayesian method that automatically determines the tensor rank in large-scale neural network training (with billions of model parameters) and supports multiple low-rank tensor formats simultaneously. This work will enable energy-efficient and low-cost training of realistic neural networks in resource-constrained scenarios such as internet of things, robotic systems, and mobile phones. Our rank-adaptive tensorized training method has reduced the memory and energy cost by two orders of magnitude when training a two-layer neural network on a preliminary FPGA prototype [53]. The

Bayesian solution will enable uncertainty quantification of the prediction results, which is important in safety-critical applications such as autonomous driving and medical imaging.

1.2. Related work.

There is a massive body of work studying the pruning [34, 40], quantization [15, 58], knowledge distillation [21], and low-rank compression [27, 33, 43, 52] of deep neural networks. This work is most related to the following previous results.

Rank determination for linear tensor problems. Many heuristic methods have been developed to estimate the tensor ranks in tensor factorization and completion. Optimization-based approaches employ a heuristic tensor nuclear norm as the surrogate of tensor rank [8, 11], but they require expensive regularization on the unfolded tensor. A nice alternative solution is to use Bayesian inference to automatically estimate tensor ranks from observed data [12, 55, 59]. Current Bayesian tensor methods solve tensor factorization, completion, and regression problems on small-scale data where the observed data is a linear function of the hidden tensor. These problems allow closed-form parameter updates in mean-field Bayesian inference [12, 55, 59]. Sampling-based Bayesian methods (i.e., Markov chain Monte Carlo) require storing thousands of copies of the model, which is not feasible for large neural networks. Because the mean-field variational approach for linear tensor problems [55, 56] does not work for tensorized neural networks, this paper develops a scalable solver based on stochastic variaitonal inference [22].

Tensorized neural networks. Most work uses tensor decomposition to compress pretrained neural networks. Examples include employing CP and Tucker factorizations to compress convolutional layers [27, 33]. In these examples the convolutional filters are already in a tensor form. It is a common practice to reshape the weights in a fully connected layer into a high-order tensor which enables tensor factorization can achieve much higher a compression ratio than matrix factorization on convolution layers [33]. As shown in [27, 57], a neural network compressed by low-rank tensor decomposition can consumes less memory, latency, and energy on resource-constrained platforms such as mobile phones. Some recent approaches train low-rank tensorized neural networks [3, 23, 41] by assuming a low-rank tensorization with a fixed maximum rank. While it is possible to tune the tensor ranks in post-training tensor compression [27, 33] based on approximation errors, one has to use manual tuning or combinatorial search to determine tensor ranks in existing tensorized training methods [3, 23, 41]. This has been a major challenge that prevents one-shot training of realistic neural networks on edge devices. Also related to our work are [30] and [28]. The work in [30] uses ℓ_1 regularization to determine CP tensor ranks in a computer vision application but requires multiple hyperparameter tuning runs, which are undesirable in the compressed training setting. The work in [28] uses dropout to randomly drop entire tensor ranks as a form of regularization during training. The dropout rate in [28], or rank reduction ratio, is the key hyperparameter that our work determines automatically.

2. Preliminaries.

2.1. Tensors and tensor decomposition.

This paper uses lowercase letters (e.g., a) to denote scalars, bold lowercase letters (e.g., $\mathbf{a}$) to represent vectors, bold uppercase letters (e.g., $\mathbf{A}$) to represent matrices, and bold calligraphic letters (e.g., $\boldsymbol{\mathcal{A}}$) to denote tensors. A tensor is a generalization of a matrix or a multiway data array. An order-d tensor is a d-way

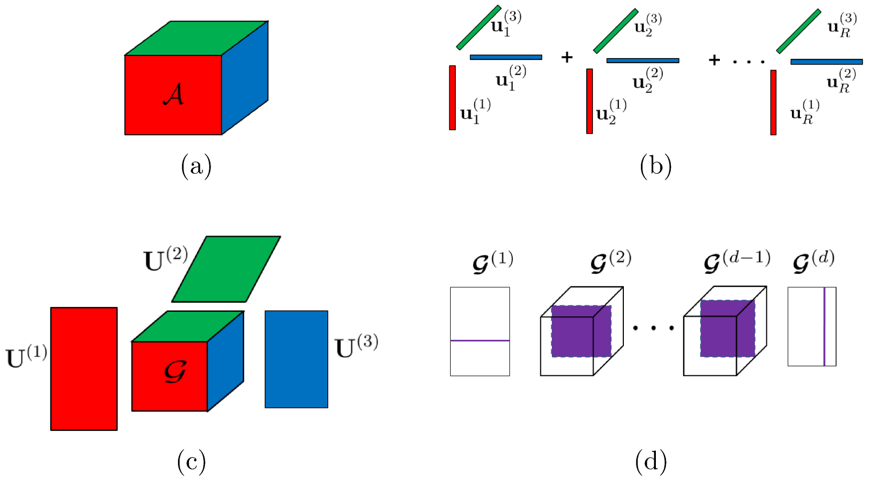


Figure 2. (a) *An order-3 tensor.* (b) *and* (c)*: Representations in CP and Tucker formats, respectively, where low-rank factors are color-coded to indicate the corresponding modes.* (d) *TT representation of an order-d tensor, where the purple lines and squares indicate $\mathcal{G}^{(n)}(:, i_n, :)$, which is the i_nth slice of the TT core $\mathcal{G}^{(n)}$ obtained by fixing its second index.*

data array $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_d}$, where I_n is the size of mode n. The $(i_1, i_2, \ldots, i_d)$th element of $\mathcal{A}$ is denoted as $a_{i_1 i_2 \cdots i_d}$. An order-3 tensor is shown in Figure 2(a).

Definition 2.1. *The mode-n product of a tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_n \times \cdots \times I_d}$ with a matrix $\mathbf{U} \in \mathbb{R}^{J \times I_n}$ is*

$$\mathcal{B} = \mathcal{A} \times_n \mathbf{U} \Longleftrightarrow b_{i_1 \ldots i_{n-1} j i_{n+1} \ldots i_d} = \sum_{i_n=1}^{I_n} a_{i_1 \ldots i_d} u_{j i_n}. \tag{2.1}$$

The result is still a d-dimensional tensor $\mathcal{B}$, but the mode-n size becomes J. In the special case $J = 1$, the nth mode diminishes and $\mathcal{B}$ becomes an order-$d-1$ tensor.

A tensor has a massive number of entries if d is large. This causes a high cost in both computing and storage. Fortunately, many practical tensors have a low-rank structure, and this property can be exploited to reduce the cost dramatically.

Definition 2.2. *A d-way tensor $\mathcal{A} \in \mathbb{R}^{I_1 \times \cdots \times I_d}$ is rank-1 if it can be written as a single outer product of d vectors*

$$\mathcal{A} = \mathbf{u}^{(1)} \circ \cdots \circ \mathbf{u}^{(d)} \textit{ with } \mathbf{u}^{(n)} \in \mathbb{R}^{I_n} \textit{ for } n = 1, \ldots, d.$$

Each element of $\mathcal{A}$ is $a_{i_1 i_2 \cdots i_d} = \prod_{n=1}^{d} u_{i_n}^{(n)}$, where $u_{i_n}^{(n)}$ is the i_nth element of the vector $\mathbf{u}^{(n)}$.

A rank-1 tensor can be stored with only d vectors. Most tensors are not rank-1, but many can be well-approximated via tensor decomposition [29] if their ranks are low. We will use the following four tensor decomposition formats to reduce the parameters of neural networks.

Definition 2.3. *The CP factorization [4, 17] expresses tensor $\mathcal{A}$ as the sum of multiple rank-1 tensors:*

$$\boldsymbol{\mathcal{A}} = \sum_{j=1}^{R} \mathbf{u}_j^{(1)} \circ \mathbf{u}_j^{(2)} \cdots \circ \mathbf{u}_j^{(d)}. \tag{2.2}$$

Here $\circ$ *denotes an outer product operator. The minimal integer* R *that ensures the equality is called the* CP rank *of* $\boldsymbol{\mathcal{A}}$*. To simplify notation we collect the rank-*1 *terms of the* n*th mode into a factor matrix* $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R}$ *with* $\mathbf{U}^{(n)}(:, j) = \mathbf{u}_j^{(n)}$*. A rank-*$R$ *CP factorization can be described with* d *factor matrices* $\{\mathbf{U}^{(n)}\}_{n=1}^{d}$ *using* $R\sum_n I_n$ *parameters.*

Definition 2.4. *The Tucker factorization* [49] *expresses a* d*-way tensor* $\boldsymbol{\mathcal{A}}$ *as a series of mode-*n *products:*

$$\boldsymbol{\mathcal{A}} = \boldsymbol{\mathcal{G}} \times_1 \mathbf{U}^{(1)} \times_2 \cdots \times_d \mathbf{U}^{(d)}. \tag{2.3}$$

Here $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_d}$ *is a small core tensor, and* $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ *is a factor matrix for the* n*th mode. The* Tucker rank *is the tuple* $(R_1, \ldots, R_d)$*. A Tucker factorization with ranks* $R_n = R$ *requires* $R^d + R\sum_n I_n$ *parameters.*

Definition 2.5. *The tensor-train (TT) factorization* [42] *expresses a* d*-way tensor* $\boldsymbol{\mathcal{A}}$ *as a collection of matrix products:*

$$a_{i_1 i_2 \ldots i_d} = \boldsymbol{\mathcal{G}}^{(1)}(:, i_1, :)\boldsymbol{\mathcal{G}}^{(2)}(:, i_2, :) \ldots \boldsymbol{\mathcal{G}}^{(d)}(:, i_d, :). \tag{2.4}$$

Each TT-core $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ *is an order-*3 *tensor. The tuple* $(R_0, R_1, \ldots, R_d)$ *is the* TT rank *and* $R_0 = R_d = 1$.

The TT format uses $\sum_n R_{n-1} I_n R_n$ parameters in total and leads to more expressive interactions than the CP format.

Let $\mathbf{A} \in \mathbb{R}^{I \times J}$ be a matrix. We assume that I and J can be factored as follows:

$$I = \prod_{n=1}^{d} I_n,\, J = \prod_{n=1}^{d} J_n. \tag{2.5}$$

We can reshape $\mathbf{A}$ into a tensor $\boldsymbol{\mathcal{A}}$ with dimensions $I_1 \times \cdots \times I_d \times J_1 \times \cdots \times J_d$, such that the (i, j)th element of $\mathbf{A}$ uniquely corresonds to the $(i_1, i_2, \ldots, i_d, j_1, j_2, \ldots, j_d)$th element of $\boldsymbol{\mathcal{A}}$. The TT decomposition can extended to compress the resulting order-$2d$ tensor as follows.

Definition 2.6. *The tensor-train matrix (TTM) factorization expresses an order-*$2d$ *tensor* $\boldsymbol{\mathcal{A}}$ *as* d *matrix products:*

$$a_{i_1 \ldots i_d j_1 \ldots j_d} = \boldsymbol{\mathcal{G}}^{(1)}(:, i_1, j_1, :)\boldsymbol{\mathcal{G}}^{(2)}(:, i_2, j_2, :) \ldots \boldsymbol{\mathcal{G}}^{(d)}(:, i_d, j_d, :). \tag{2.6}$$

Each TT-core $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times J_n \times R_n}$ *is an order* 4 *tensor. The tuple* $(R_0, R_1, R_2, \ldots, R_d)$ *is the* TT rank*, and as before* $R_0 = R_d = 1$*. This TTM factorization requires* $\sum_n R_{n-1} I_n J_n R_n$ *parameters to represent* $\boldsymbol{\mathcal{A}}$*.*

We provide a visual representation of the CP, Tucker, and TT formats in Figure 2(b)–(d).

2.2. Tensorized neural networks.

A deep neural network can be written as

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) = \mathbf{g}_L\left(\mathbf{g}_{L-1}\left(\cdots \mathbf{g}_1(\mathbf{x})\right)\right), \tag{2.7}$$

where $\mathbf{x}$ is an input data sample and $\mathbf{y}$ is an output label. Here $\mathbf{g}_k(\mathbf{z}) = \sigma(\mathbf{W}_k\mathbf{z} + \mathbf{b}_k)$ represents layer k, where σ is a nonlinear activation function, $\mathbf{W}_k$ and $\mathbf{b}_k$ are the weights and bias, respectively. Considering parameter dependence, we can rewrite (2.7) as

$$\mathbf{y} = \mathbf{h}(\mathbf{x} \mid \{\mathbf{W}_k, \mathbf{b}_k\}_{k=1}^{L}). \tag{2.8}$$

In a convolutional layer $\mathbf{W}_k$ should be replaced with tensor $\boldsymbol{\mathcal{W}}_k$. In modern neural networks, $\{\mathbf{W}_k\}_{k=1}^{L}$ contain millions to billions of parameters, which cause huge challenges in training and inference on various hardware platforms. A promising solution is to generate a compact neural network via low-rank tensor compression [9, 33, 41] as follows:

- **Folding to high-order tensors.** A weight matrix $\mathbf{W} \in \mathbb{R}^{I\times J}$ can be folded into an order-d tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1\times\cdots\times I_d}$, where $IJ = \prod_n I_n$. We can also fold $\mathbf{W}$ to an order-$2d$ tensor $\boldsymbol{\mathcal{A}} \in \mathbb{R}^{I_1\times\cdots\times I_d\times J_1\times\cdots\times J_d}$ such that $w_{ij} = a_{i_1\cdots i_d j_1\cdots j_d}$. While a convolution filter is already a tensor, we can reshape it to a higher-order tensor with reduced mode sizes.
- **Low-rank tensor compression.** After folding $\mathbf{W}$ into a higher-order tensor $\boldsymbol{\mathcal{A}}$, one can employ low-rank tensor compression to reduce the number of parameters. Either the CP, Tucker, TT, or TTM factorization can be applied [9, 27, 33].

Assume that $\boldsymbol{\Phi}_k$ includes all low-rank tensor factors required to represent $\mathbf{W}_k$. Considering the dependence of $\mathbf{W}_k$ on $\boldsymbol{\Phi}_k$, we can now write (2.8) as

$$\mathbf{y} = \mathbf{h}\left(\mathbf{x} \mid \{\mathbf{W}_k\left(\boldsymbol{\Phi}_k\right), \mathbf{b}_k\}_{k=1}^{L}\right) = \mathbf{f}\left(\mathbf{x} \mid \boldsymbol{\Psi}\right) \text{ with } \boldsymbol{\Psi} = \{\boldsymbol{\Phi}_k, \mathbf{b}_k\}_{k=1}^{L}. \tag{2.9}$$

Here $\boldsymbol{\Psi}$ include all tensor factors and bias vectors in a tensorized neural network. The number of variables in $\boldsymbol{\Psi}$ is often orders-of-magnitude smaller than that in the original model (2.8).

Please note the following:

- The tensor factors in $\boldsymbol{\Phi}_k$ depend on the tensor format we choose. In CP format, $\boldsymbol{\Phi}_k$ includes d matrix factors; in Tucker format, $\boldsymbol{\Phi}_k$ includes d factor matrices and a small order-d core tensor as shown in (2.3); when the TT or TTM format is used, $\boldsymbol{\Phi}_k$ includes d order-3 or order-4 TT cores shown in (2.4) and (2.6), respectively.
- The number of variables in each $\boldsymbol{\Phi}_k$ depends on the tensor ranks used in the compression. A higher tensor rank leads to higher expressive power but a lower compression ratio. In existing approaches, it is hard to select a proper tensor rank a priori.

Two main approaches exist to produce low-rank tensorized neural networks. The first approach trains an uncompressed neural network $\mathbf{h}$ and then performs tensor factorization on each of the weights $\{\mathbf{W}_k\}_{k=1}^{L}$. This train-then-compress approach suffers from two drawbacks:

- **High training costs.** The uncompressed training consumes a huge amount of memory, run-time, and energy on a hardware platform.
- **Lower accuracy.** The subsequent tensor compression causes accuracy loss, which becomes significant when the compression ratio is high.

The second approach is fixed-rank tensorized training. In this approach the user prespecifies the tensor rank and trains low-rank tensor factors of weight parameters. This approach avoids the compute and memory requirements of uncompressed training but requires that the user manually select a good rank a priori. This approach usually requires multiple training runs to select the rank. In addition a user-specified rank may achieve suboptimal compression.

3. Bayesian low-rank tensorized model.

In this work, we plan to develop a tensorized training method that can automatically determine the tensor ranks in the training process. This method requires only one training run and avoids the high cost of uncompressed training. Bayesian methods have been employed for tensor completion and factorization [55, 56, 59], where the observed data is a linear function of tensor elements. However, existing Bayesian tensor solvers do not work for tensorized neural networks due to the nonlinear forward model and large number of unknown variables.

3.1. High-level Bayesian formulation.

We first describe a general-purpose Bayesian model for training low-rank tensorized neural networks. For notational convenience we assume that our neural network $\mathbf{f}$ has one nonlinear layer and that its weight matrix $\mathbf{W}$ is folded to a single tensor $\boldsymbol{\mathcal{A}}$. Extending our method to general multilayer cases with multiple tensors is straightforward, and we will report results on general multilayer models in section 5.

Given a training dataset $\mathcal{D}$, our goal is to determine the unknown low-rank factors $\mathbf{\Phi}$ for $\boldsymbol{\mathcal{A}}$, the associated tensor ranks, and the bias vector $\mathbf{b}$. We introduce hyperparameters $\mathbf{\Lambda}$ to control the tensor ranks and model complexity. Our posterior distribution is

$$p(\mathbf{\Psi}, \mathbf{\Lambda}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{\Psi})p(\mathbf{\Psi}, \mathbf{\Lambda})}{p(\mathcal{D})} \text{ with } \mathbf{\Psi} = \{\mathbf{\Phi}, \mathbf{b}\}. \tag{3.1}$$

Here $p(\mathcal{D}|\mathbf{\Psi})$ is the model likelihood, $p(\mathbf{\Psi}, \mathbf{\Lambda})$ is the joint prior, and $p(\mathcal{D})$ is the model evidence

$$p(\mathcal{D}) = \int_{\mathbf{\Psi},\mathbf{\Lambda}} p(\mathcal{D}|\mathbf{\Psi})p(\mathbf{\Psi}, \mathbf{\Lambda})d\mathbf{\Psi}d\mathbf{\Lambda}. \tag{3.2}$$

The likelihood and joint prior are specified below:

- **Likelihood function.** $p(\mathcal{D}|\mathbf{\Psi})$ and data $\mathcal{D}$ are determined by a forward propagation model. Let $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ be a training sample where $\mathbf{x}$ is the neural network input and $\mathbf{y}$ is the associated true label. The multinomial likelihood function for a neural network classifier with C potential classes is

$$p(\mathcal{D}|\mathbf{\Psi}) \propto \prod_{(\mathbf{x},\mathbf{y})\in\mathcal{D}} \prod_{c=1}^{C} \mathbf{f}(\mathbf{x}|\mathbf{\Psi})_c^{y_c}, \tag{3.3}$$

  where y_c is the correct class label. Here $\mathbf{f}$ is the forward propagation model in (2.9) which is conditioned on the given low-rank tensor factors and bias vectors. We omit the multinomial distribution constant of proportionality for simplicity.
- **Joint prior** We place an independent prior over the low-rank tensor factors and the bias term. We choose a weak normal prior for the bias term:

$$p(\mathbf{\Psi}, \mathbf{\Lambda}) = p(\mathbf{b})p(\mathbf{\Phi}, \mathbf{\Lambda}), \quad p(\mathbf{b}) \propto \prod_i \frac{1}{\sigma_0^2} \exp\left(-\frac{b_i^2}{2\sigma_0^2}\right). \tag{3.4}$$

Here $p(\mathbf{\Phi}, \mathbf{\Lambda})$ is the joint prior for tensor factors $\mathbf{\Phi}$ and hyperparameters $\mathbf{\Lambda}$. The design of $p(\mathbf{\Phi}, \mathbf{\Lambda})$ depends on the tensor format we choose, which will be explained in sections 3.2 and 3.3.

3.2. Tensor factor priors. Proper priors should be chosen in order to automatically shrink tensor ranks in the training process. Here we will specify the joint prior $p(\mathbf{\Phi}, \mathbf{\Lambda})$ for the four tensor formats described in section 2.1: CP, Tucker, TT, and TTM.

Firstly we specify the general form of $p(\mathbf{\Phi}, \mathbf{\Lambda})$. For the CP format, we initialize each factor $\mathbf{U}^{(n)}$ as a matrix with R columns. Assume that R is larger than the actual rank r and all factors shrink to r columns in the training process. All CP factors have the same maximum rank (column number), so we use a single vector $\mathbf{\Lambda} = \boldsymbol{\lambda} \in \mathbb{R}^R$ to control the rank. The tensor rank in Tucker, TT, or TTM format is a vector, and the rank associated with each mode can be different. Therefore, we require a collection of vectors $\mathbf{\Lambda} = \{\boldsymbol{\lambda}^{(n)}\}_{n=1}^d$ to control the ranks of each mode individually. Here $\boldsymbol{\lambda}^{(n)} \in \mathbb{R}^{R_n}$, and the "maximum rank" R_n exceeds the "actual rank" r_n of mode n. As a result, we introduce the general form

$$p(\mathbf{\Phi}, \mathbf{\Lambda}) = \begin{cases} p(\mathbf{\Phi}|\boldsymbol{\lambda})p(\boldsymbol{\lambda}) \text{ for CP format,} \\ p(\mathbf{\Phi}|\{\boldsymbol{\lambda}^{(n)}\}) \prod\limits_{n=1}^{d} p(\boldsymbol{\lambda}^{(n)}) \text{ for Tucker, TT, and TTM formats,} \end{cases} \tag{3.5}$$

where the prior distribution(s) on $\boldsymbol{\lambda}$ or $\{\boldsymbol{\lambda}^{(n)}\}_{n=1}^d$ enforce(s) rank reduction.

Next we specify the tensor factor priors $p(\mathbf{\Phi}|\boldsymbol{\lambda})$ or $p(\mathbf{\Phi}|\{\boldsymbol{\lambda}^{(n)}\})$ for each tensor format, and we defer the prior on $\boldsymbol{\lambda}$ and $\{\boldsymbol{\lambda}^{(n)}\}_{n=1}^d$ to section 3.3.

- **CP format.** The CP tensor factors are d matrices $\mathbf{\Phi} = \{\mathbf{U}^{(n)}\}_{n=1}^d$. We assign a Gaussian prior with controllable variance to each element of each factor matrix $\mathbf{U}^{(n)}$:

$$p(\mathbf{\Phi}, \mathbf{\Lambda}) = p(\boldsymbol{\lambda}) \prod_n p\left(\mathbf{U}^{(n)}|\boldsymbol{\lambda}\right), \quad p\left(\mathbf{U}^{(n)} \mid \boldsymbol{\lambda}\right) = \prod_{i,j} \mathcal{N}\left(u_{ij}^{(n)} \mid 0, \lambda_j\right). \tag{3.6}$$

Here $u_{ij}^{(n)}$ is the (i, j)th element of $\mathbf{U}^{(n)}$. Each entry of $\boldsymbol{\lambda}$ controls one column of each factor matrix. If a single entry λ_j approaches zero, then the prior mean and prior variance of $u_{ij}^{(n)}$ are both close to zero for all row indices $i \in [1, I_n]$ and mode indices $n \in [1, d]$. This encourages the whole jth column of $\mathbf{U}^{(n)}$ to shrink to zero, leading to a rank reduction. The vector $\boldsymbol{\lambda}$ is shared across all modes; therefore, it will shrink the same column of all CP factor matrices simultaneously, as shown in Figure 3(a).

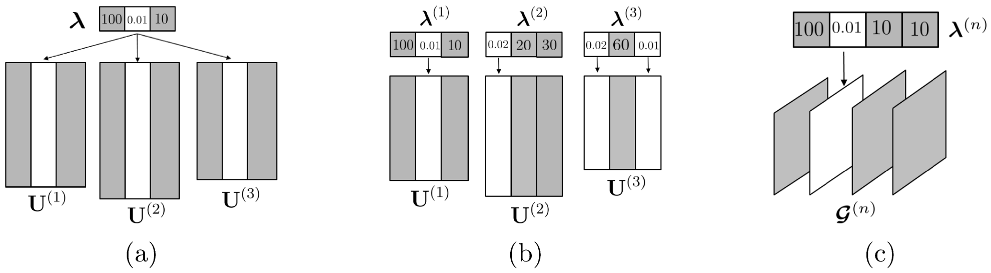


Figure 3. (a) *For the CP prior, if one element of* $\boldsymbol{\lambda}$ *is small, one column is removed from every factor matrix.* (b) *For the Tucker prior, if one element of* $\boldsymbol{\lambda}^{(n)}$ *is small, then one column of* $\mathbf{U}^{(n)}$ *shrinks to zero.* (c) *For the TT prior, if one element of* $\boldsymbol{\lambda}^{(n)}$ *is small, then one slice of* $\boldsymbol{\mathcal{G}}^{(n)}$ *shrinks to zero. The columns/slices to be removed are marked in white.*

- **Tucker format.** A Tucker factorization includes a core tensor and d factor matrices; therefore, $\mathbf{\Phi} = \{\boldsymbol{\mathcal{G}}, \{\mathbf{U}^{(n)}\}_{n=1}^{d}\}$. We also assign each factor matrix $\mathbf{U}^{(n)}$ with a variance-tunable Gaussian distribution. A Tucker model has d separate rank parameters $(r_1, \dots, r_d)$ to determine, one per factor matrix as shown in Figure 3(b). Furthermore, the factor matrices and core tensor are handled separately. Therefore, we propose the following prior distributions:

$$p(\mathbf{\Phi}, \mathbf{\Lambda}) = p(\boldsymbol{\mathcal{G}}) \prod_{n} p\left(\mathbf{U}^{(n)}|\boldsymbol{\lambda}^{(n)}\right) p\left(\boldsymbol{\lambda}^{(n)}\right), \quad p\left(\mathbf{U}^{(n)} \mid \boldsymbol{\lambda}^{(n)}\right) = \prod_{i,j} \mathcal{N}\left(u_{ij}^{(n)} \mid 0, \lambda_j^{(n)}\right).$$

We use d independent rank controlling vectors $\{\boldsymbol{\lambda}^{(n)}\}_{n=1}^{d}$ to control the prior variances of different factor matrices separately. The jth element of $\boldsymbol{\lambda}^{(n)}$ controls the jth column of factor matrix $\mathbf{U}^{(n)}$. Therefore $\boldsymbol{\lambda}^{(n)}$ controls r_n, the nth entry of the Tucker rank. We place a weak normal prior over the entries of the core tensor $\boldsymbol{\mathcal{G}}$:

$$p\left(\boldsymbol{\mathcal{G}}\right) = \prod_{i_1,\dots,i_d} \mathcal{N}\left(g_{i_1 \dots i_d} \mid 0, \sigma_0\right). \tag{3.7}$$

We make this choice to simplify parameter inference compared to the alternative of placing low-rank priors on both of the core tensor and the factor matrices.

- **TT format.** A TT factorization has d order-3 TT cores; therefore, $\mathbf{\Phi} = \{\boldsymbol{\mathcal{G}}^{(n)}\}_{n=1}^{d}$. The TT format requires a more complicated prior because each TT core $\boldsymbol{\mathcal{G}}^{(n)} \in \mathbb{R}^{r_{n-1} \times I_n \times r_n}$ depends on two rank parameters r_{n-1} and r_n. In order to automatically determine the TT rank, we choose $R_n > r_n$ and initialize the nth TT core with size $R_{n-1} \times I_n \times R_n$. The prior density of all TT cores is given as

$$\begin{aligned} p(\mathbf{\Phi}, \mathbf{\Lambda}) &= p\left(\boldsymbol{\mathcal{G}}^{(d)}|\boldsymbol{\lambda}^{(d-1)}\right) \prod_{1 \le n \le d-1} p\left(\boldsymbol{\mathcal{G}}^{(n)}|\boldsymbol{\lambda}^{(n)}\right) p\left(\boldsymbol{\lambda}^{(n)}\right), \\ p\left(\boldsymbol{\mathcal{G}}^{(n)} \mid \boldsymbol{\lambda}^{(n)}\right) &= \prod_{i,j,k} \mathcal{N}\left(g_{ijk}^{(n)} \mid 0, \lambda_k^{(n)}\right) \quad \text{for } n \in [1, d-1], \\ p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) &= \prod_{i,j,k} \mathcal{N}\left(g_{ijk}^{(d)} \mid 0, \lambda_i^{(d-1)}\right). \end{aligned} \tag{3.8}$$

We introduce a vector $\boldsymbol{\lambda}^{(n)} \in \mathbb{R}^{R_n}$ to control the actual rank r_n for mode 1 to $d-1$. As shown in Figure 3(c), the kth element of $\boldsymbol{\lambda}^{(n)}$ (i.e., $\lambda_k^{(n)}$) controls the prior variance of a slice $\boldsymbol{\mathcal{G}}^{(n)}(:,:,k)$. If $\lambda_k^{(n)}$ is small, the whole slice $\boldsymbol{\mathcal{G}}^{(n)}(:,:,k)$ is close to zero, leading to a rank reduction in the nth mode. Parameter $\boldsymbol{\lambda}^{(d-1)}$ controls two separate cores. This prevents any rank parameters from overlapping, and it simplifies posterior inference.

- **TTM format.** Similar to the TT format, a TTM decomposition also has d core tensors; therefore, $\mathbf{\Phi} = \{\boldsymbol{\mathcal{G}}^{(n)}\}_{n=1}^{d}$. The only difference is that each $\boldsymbol{\mathcal{G}}^{(n)}$ is an order-4 tensor, which is initalized with a size $R_{n-1} \times I_n \times J_n \times R_n$ in our Bayesian model. The prior for the TTM low-rank factors is

$$
\begin{aligned}
p(\mathbf{\Phi}, \mathbf{\Lambda}) &= p\left(\boldsymbol{\mathcal{G}}^{(d)}|\boldsymbol{\lambda}^{(d-1)}\right) \prod_{1\le n\le d-1} p\left(\boldsymbol{\mathcal{G}}^{(n)}|\boldsymbol{\lambda}^{(n)}\right) p\left(\boldsymbol{\lambda}^{(n)}\right), \\
p\left(\boldsymbol{\mathcal{G}}^{(n)} \mid \boldsymbol{\lambda}^{(n)}\right) &= \prod_{i,j,k,l} \mathcal{N}\left(g^{(n)}_{ijkl} \mid 0, \lambda^{(n)}_l\right) \text{ for } n \in [1, d-1], \\
p\left(\boldsymbol{\mathcal{G}}^{(d)} \mid \boldsymbol{\lambda}^{(d-1)}\right) &= \prod_{i,j,k,l} \mathcal{N}\left(g^{(d)}_{ijkl} \mid 0, \lambda^{(d-1)}_i\right).
\end{aligned} \tag{3.9}
$$

This prior is very similar to that of the TT format. We use a vector parameter $\boldsymbol{\lambda}^{(n)}$ to control the actual rank r_n of the nth mode for $n \in [1, d-1]$, and $\boldsymbol{\lambda}^{(d-1)}$ is shared among $\boldsymbol{\mathcal{G}}^{(d)}$ and $\boldsymbol{\mathcal{G}}^{(d-1)}$.

3.3. Rank-shrinking hyperparameter priors.

To complete the setup of the full Bayesian model (3.1), we still need to specify the prior of rank-control hyperparameters $\mathbf{\Lambda} = \boldsymbol{\lambda}$ (for CP) or $\mathbf{\Lambda} = \{\boldsymbol{\lambda}^{(n)}\}_{n=1}^{d}$ (for Tucker, TT, and TTM). Since small elements in $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^{(n)}$ lead to rank reductions in the tensor models, we choose two hyperprior densities that place high probability near zero. We focus our notation in this subsection on the CP model for simplicity.

We consider two choices of prior on the hyperparameter $\boldsymbol{\lambda}$: the half-Cauchy (HC) with scale parameter η and the improper log-uniform (LU) on $(0, \infty)$:

$$
p(\boldsymbol{\lambda}) = \prod_{i=1}^{R} p(\lambda_i) \quad \text{with } p(\lambda_i) = \begin{cases} \mathrm{HC}(\sqrt{\lambda_i}|0, \eta) \text{ or} \\ \mathrm{LU}(\sqrt{\lambda_i}). \end{cases} \tag{3.10}
$$

The improper LU distribution has a fatter tail than the HC distribution and is parameter-free. We illustrate both densities in Figure 4(a). The HC scaling parameter $\eta > 0$ can be adjusted to tune the tradeoff between accuracy and rank-sparsity. Decreasing the magnitude of η increases rank-sparsity. Both the HC density function,

$$
\mathrm{HC}(x|0, \eta) \propto \left(1 + \frac{x^2}{\eta^2}\right)^{-1}, \tag{3.11}
$$

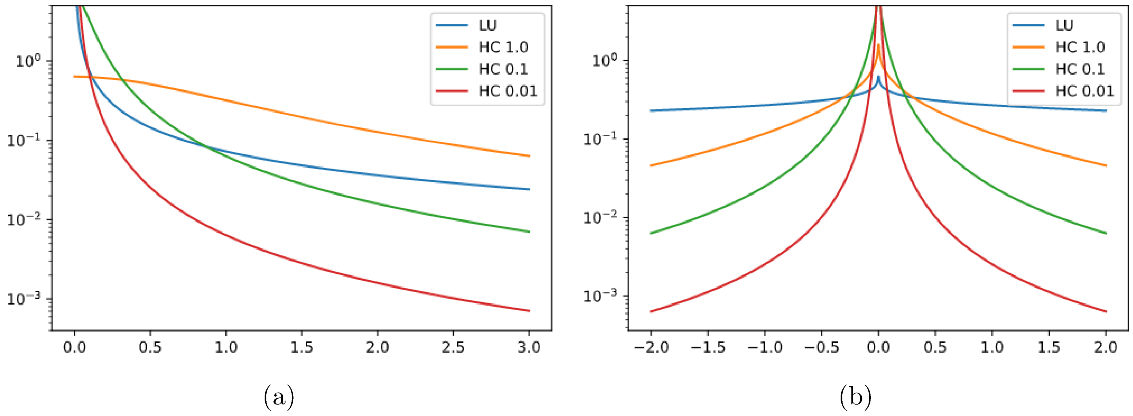


Figure 4. (a) *Comparison of the probability density functions of the LU and HC hyperprior on λ_j. Several values of the HC scale parameter η are given.* (b) *Comparison of the probability density functions of the corresponding marginal prior on the low-rank tensor factor entry $u^{(n)}_{ij}$.*

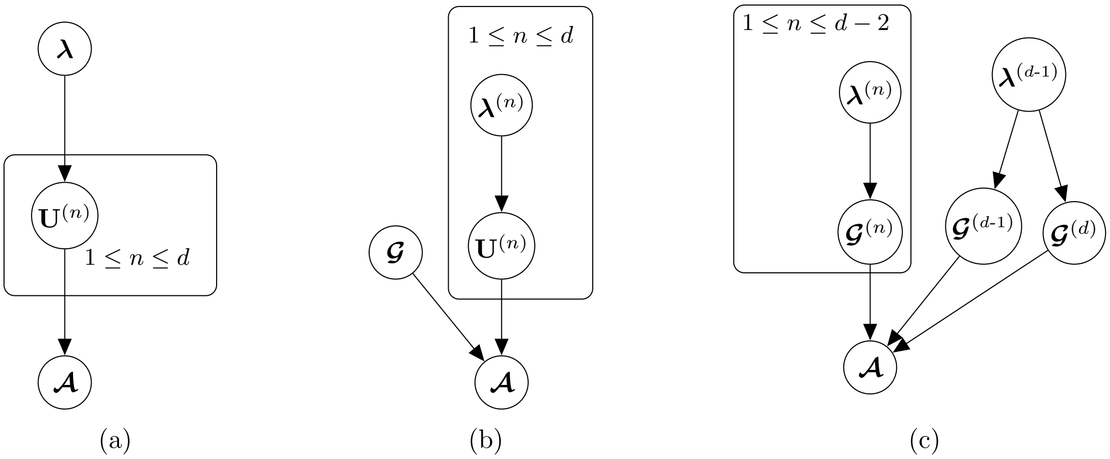

$\boldsymbol{\lambda}$
$\mathbf{U}^{(n)}$
$1 \le n \le d$
$\boldsymbol{\mathcal{A}}$
(a)
$1 \le n \le d$
$\boldsymbol{\lambda}^{(n)}$
$\mathbf{U}^{(n)}$
$\boldsymbol{\mathcal{G}}$
$\boldsymbol{\mathcal{A}}$
(b)
$1 \le n \le d-2$
$\boldsymbol{\lambda}^{(n)}$
$\boldsymbol{\mathcal{G}}^{(n)}$
$\boldsymbol{\lambda}^{(d\text{-}1)}$
$\boldsymbol{\mathcal{G}}^{(d\text{-}1)}$
$\boldsymbol{\mathcal{G}}^{(d)}$
$\boldsymbol{\mathcal{A}}$
(c)

Figure 5. (a) *CP graphical model;* (b) *Tucker graphical model;* (c) *TT/TTM graphical model.*

and the LU density function,

$$\mathrm{LU}(x) \propto x^{-1}, \tag{3.12}$$

place high probability in regions around zero. The parameter $\boldsymbol{\lambda}$ controls the prior variance of the tensor factors in $\boldsymbol{\Phi}$, all of which have prior mean zero. Therefore the prior density encodes a prior belief that the tensor rank is low, and it encourages structured rank shrinkage. We provide the Bayesian graphical models for each low-rank tensor format in Figure 5.

In Figure 4 we demonstrate how our prior induces rank-sparsity in a CP model. Figure 4(a) plots the prior density on the rank parameter λ_j. Figure 4(b) shows the corresponding marginal prior on $u_{ij}^{(n)}$. The flat tail and sharp peak of the marginal prior induced by the LU rank hyperprior leads to strong shrinkage of small values of $u_{ij}^{(n)}$ towards 0 but permits medium values to escape the "gravitational pull" around 0 [5]. In comparison, the marginal horseshoe prior induced by the HC hyperprior exerts a weaker shrinkage effect at small values of $u_{ij}^{(n)}$ but a stronger shrinkage effect on larger values.

4. Scalable parameter inference.

Now we discuss how to estimate the resulting posterior density (3.1). We develop an efficient tensorized Bayesian inference approach by improving stochastic variational inference (SVI) [22]. We consider SVI [22] due to its superior computational and memory efficiency over gradient-based Markov Chain Monte Carlo [39] and Stein variational gradient descent [35]. However, directly applying SVI to our tensorized training can cause numerical failures. Therefore, we will develop a customized SVI solver with an analytical/numerical hybrid parameter update that is suitable for our Bayesian tensorized neural networks.

4.1. Review of SVI.

Let $\boldsymbol{\theta}$ be the parameters to infer, and let $q(\boldsymbol{\theta})$ be the approximating distribution to the target posterior distribution

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}).$$

SVI [22] solves an optimization problem where the loss function is the Kullback–Leibler (KL) divergence and the goal is to find the best approximating density $q^\star$ among a parameterized class of densities $\mathcal{P}$:

$$q^\star(\boldsymbol{\theta}) = \mathop{\arg\min}_{q(\boldsymbol{\theta})\in\mathcal{P}} \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})\right), \quad \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})\right) = \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log \frac{q(\boldsymbol{\theta})}{p(\boldsymbol{\theta}|\mathcal{D})}\right]. \tag{4.1}$$

The KL divergence can be rewritten as

$$\begin{aligned}\mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta}|\mathcal{D})\right) &= \mathbb{E}_{q(\boldsymbol{\theta})}\left[\log q(\boldsymbol{\theta}) - \log p(\mathcal{D}|\boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\right] + \text{const.}\\ &= -\mathbb{E}_{q(\boldsymbol{\theta})}\left[\log p(\mathcal{D}|\boldsymbol{\theta})\right] + \mathrm{KL}\left(q(\boldsymbol{\theta})||p(\boldsymbol{\theta})\right) + \text{const.}\end{aligned} \tag{4.2}$$

This is a combination of the log-likelihood (model fit) and the divergence from the approximate posterior to the prior (low-rank). To approximate the log-likelihood one samples from the variational distribution q. The KL divergence is either approximated via sampling or evaluated in a closed form. The form in (4.2) requires the evaluation of the full-data model likelihood. If the data is large the full-data likelihood $p(\mathcal{D}|\boldsymbol{\theta})$ is intractable, so we approximate the likelihood by subsampling a minibatch $\mathcal{M} \subset \mathcal{D}$.

4.2. Challenges in training Bayesian tensorized neural networks.

Now we explain the challenges of directly applying SVI to train our Bayesian tensorized neural network model. As an example, we focus our notation on the CP format one-layer model with parameters

$$\boldsymbol{\theta} = \{\boldsymbol{\Phi}, \boldsymbol{\Lambda}\} = \left\{\left\{\mathbf{U}^{(n)}\right\}_{n=1}^{d}, \boldsymbol{\lambda}\right\}. \tag{4.3}$$

The extension to other tensor formats and to multiple layers is trivial. For notational convenience we omit the description of the bias term $\mathbf{b}$ since it is assigned a normal variational posterior and follows the standard update rules specified in [2].

In variational inference, it is a common practice to simplify a posterior density in order to reduce the computational cost. In our problem setting, we firstly use the mean-field approximation [25] to achieve a tractable optimization:

$$q\left(\left\{\mathbf{U}^{(n)}\right\}, \boldsymbol{\lambda}\right) = q_{\mathbf{U}}\left(\left\{\mathbf{U}^{(n)}\right\}\right) q_{\boldsymbol{\lambda}}\left(\boldsymbol{\lambda}\right). \tag{4.4}$$

We further model the posterior of the tensor factors with a normal distribution

$$q_{\mathbf{U}}\left(\left\{\mathbf{U}^{(n)}\right\}\right) = \prod_{n=1}^{d} q_{\mathbf{U}^{(n)}}\left(\mathbf{U}^{(n)}\right), \quad q_{\mathbf{U}^{(n)}}\left(\mathbf{U}^{(n)}\right) = \prod_{i,j} \mathcal{N}\left(u_{ij}^{(n)} | \overline{u_{ij}^{(n)}}, {\Sigma_{ij}^{(n)}}^2\right), \tag{4.5}$$

where $\overline{u_{ij}^{(n)}}$ and $\Sigma_{ij}^{(n)}$ are the (i,j)th elements of the unknown posterior mean $\overline{\mathbf{U}^{(n)}}$ and posterior standard deviation $\boldsymbol{\Sigma}^{(n)}$ to be inferred, respectively.

Now we discuss the challenges in learning the variational posterior distribution. We modify (4.2) to obtain our objective function:

$$\mathcal{L}(q) = -\mathbb{E}_{q(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda})} \log\, p\left(\mathcal{D}|\left\{\mathbf{U}^{(n)}\right\}\right) + \mathrm{KL}\left(q\left(\left\{\mathbf{U}^{(n)}\right\}, \boldsymbol{\lambda}\right)||p\left(\left\{\mathbf{U}^{(n)}\right\}, \boldsymbol{\lambda}\right)\right). \tag{4.6}$$

Due to the nonlinear tensorized forward model, we need to employ gradient-based iterations in SVI to update the tensor factor parameters. The expected log-likelihood in (4.6) must be approximated by sampling the variational distribution q. The first standard approach is to select a variational distribution $q\left(\{\mathbf{U}^{(n)}\}, \boldsymbol{\lambda}\right)$ for which the KL divergence in (4.6) can be obtained in a closed form. The second standard approach is to approximate the KL divergence term by sampling from the variational posterior. In practice, two challenges prevent us from applying these standard SVI approaches:

- **Challenge 1: Closed-form objectives require multiple training runs.** Variational distributions q that permit a closed-form approximation of the KL divergence require additional hyperparameters. Existing distributions that enable a closed-form KL divergence require a hierarchical Bayesian parameterization of the rank parameter $\boldsymbol{\lambda}$ [10, 50], requiring up to five additional hyperparameters for the new random variables [50]. Additional hyperparameters would require additional tuning runs and remove the benefits of one-shot tensorized training. Therefore, we avoid this option.
- **Challenge 2: Sampling-based approximation increases gradient variance.** Sampling-based approximation of the KL divergence leads to gradient instability during rank shrinkage. The gradient variance with respect to the low-rank tensor factor parameters is proportional to the variance of $1/\boldsymbol{\lambda}$, and it may explode during rank-shrinkage as $\boldsymbol{\lambda}$ approaches 0, so sampling $\boldsymbol{\lambda}$ is not feasible.

We provide more details about the second challenge. We consider the gradient of the objective function in (4.6) with respect to the parameters $\overline{u_{ij}^{(n)}}$ and $\Sigma_{ij}^{(n)}$. First we observe that

$$\text{KL}\left(\mathcal{N}\left(u_{ij}^{(n)}|\overline{u_{ij}^{(n)}}, {\Sigma_{ij}^{(n)}}^2\right)||\mathcal{N}\left(u_{ij}^{(n)}|0,\lambda_j\right)\right) \propto \frac{\overline{u_{ij}^{(n)}}^2 + {\Sigma_{ij}^{(n)}}^2}{\lambda_j}. \tag{4.7}$$

Let ϕ represent either parameter of $\{\overline{u_{ij}^{(n)}}, \Sigma_{ij}^{(n)}\}$. Then sampling $\boldsymbol{\lambda}$ yields a gradient variance

$$\mathbb{V}\left[\nabla_\phi \text{KL}\left(\mathcal{N}\left(u_{ij}^{(n)}|\overline{u_{ij}^{(n)}}, {\Sigma_{ij}^{(n)}}^2\right)||\mathcal{N}\left(u_{ij}^{(n)}|0,\lambda_j\right)\right)\right] \propto \mathbb{V}\left[\frac{1}{\lambda_j}\right]. \tag{4.8}$$

The goal of our low-rank prior is to shrink many λ_j's to 0 in the training process. If the distribution of λ_j is nondegenerate, even small uncertainties in the value of λ_j will lead to large variance in (4.8) as the posterior probability of λ_j concentrates around 0. As a result, a rank shrinkage can cause high-variance gradients which in turn may increase the magnitude of factor matrix parameters, as shown in Figure 6.

4.3. Simplified posterior for rank-controlling hyperparameters.

To avoid gradient variance explosion, we propose a deterministic approximation to the hyperparameter $\boldsymbol{\lambda}$:

$$q_{\boldsymbol{\lambda}}(\boldsymbol{\lambda}) = \delta_{\overline{\boldsymbol{\lambda}}}(\boldsymbol{\lambda}), \tag{4.9}$$

where δ is a delta function and $\overline{\boldsymbol{\lambda}}$ is the posterior mean of $\boldsymbol{\lambda}$. This delta approximation was used for empirical partially Bayes estimation in [37]. This approximation admits *closed-form updates* to the following subproblem when the factor matrices are fixed:

$$\underset{\overline{\lambda}_k}{\arg\min}\, \text{KL}\left(q\left(\left\{\mathbf{U}^{(n)}\right\}, \boldsymbol{\lambda}\right)||p\left(\left\{\mathbf{U}^{(n)}\right\}, \boldsymbol{\lambda}\right)\right). \tag{4.10}$$

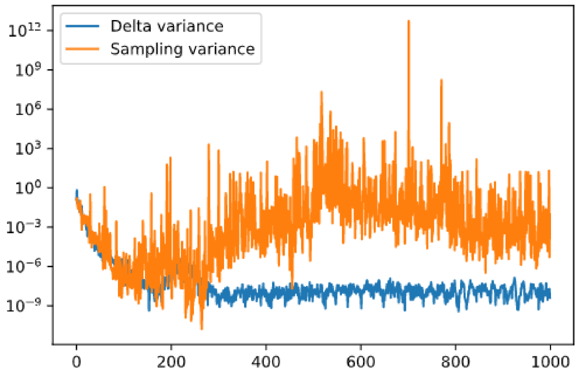


Figure 6. *The gradient variance of a single low-rank tensor factor parameter. Sampling the rank parameter* $\boldsymbol{\lambda}$ *leads to high-variance gradients, while our proposed delta approximation of hyperparameters reduces the gradient variance significantly (see section 4.3 and section 4.4).*

We provide the closed-form analytical updates for $\overline{\lambda}_k$ under each choice of prior in CP format and give the details in Appendix A. The results associated with other tensor formats can be obtained similarly. For the LU prior

$$\overline{\lambda}_k^{\star} \leftarrow \frac{M}{D+1}. \tag{4.11}$$

Here we have used the notations

$$D = \sum_n I_n, \quad M = \sum_{1\le n\le d} \sum_{1\le i\le I_n} \overline{u_{ik}^{(n)}}^2 + \Sigma_{ik}^{(n)2}. \tag{4.12}$$

The number of entries controlled by λ_j is D, and M is their combined magnitude and variance. In the case of the HC prior with scale parameter η, the update is

$$\overline{\lambda}_k^{\star} \leftarrow \frac{M - \eta^2 D + \sqrt{M^2 + (2D+8)\eta^2 M + \eta^4 D^2}}{2D+2}. \tag{4.13}$$

For the HC hyperprior, decreasing the magnitude of the scale parameter η decreases the magnitude of the update of $\overline{\lambda}_k^{\star}$, thereby increasing rank-sparsity.

4.4. Analytical/numerical hybrid parameter update in SVI.

With the proposed delta posterior approximation for $\boldsymbol{\lambda}$, now we can train our tensorized neural network training with an analytical/numerical hybrid parameter update rule in SVI. Specifically, in every iteration of SVI, we use a gradient-based half step to update the tensor factors in $\boldsymbol{\Phi}$ and closed-form half step to the hyperparameters $\overline{\boldsymbol{\lambda}}$. We apply the reparametrization trick

$$u_{ij}^{(n)} = \overline{u_{ij}^{(n)}} + z\Sigma_{ij}^{(n)}, \qquad z \sim \mathcal{N}(0,1) \tag{4.14}$$

to sample from the tensor factor distributions.

- **Half Step 1: Gradient update for tensor factors.** We sample the low-rank tensor factors $\boldsymbol{\Phi}$ and update all parameters of the tensor factor variational distributions using gradient descent on the loss $\mathcal{L}(q)$ of (4.6) with a learning rate α:

$$\boldsymbol{\Phi} \leftarrow \boldsymbol{\Phi} + \alpha \nabla_{\boldsymbol{\Phi}} \mathcal{L}(q)\,. \tag{4.15}$$

In the the CP model, the gradients for the posterior variance and mean of the factor matrices are given by

$$\begin{aligned} \nabla_{\Sigma_{ij}^{(n)}} \mathcal{L}(q) &= -z \nabla_{u_{ij}} \log\, p\left(\mathcal{D} \middle| \left\{\mathbf{U}^{(n)}\right\}\right) - \frac{1}{\Sigma_{ij}^{(n)}} + \frac{\Sigma_{ij}^{(n)}}{\overline{\lambda}_j}, \\ \nabla_{\overline{u_{ij}^{(n)}}} \mathcal{L}(q) &= -\nabla_{u_{ij}} \log\, p\left(\mathcal{D} \middle| \left\{\mathbf{U}^{(n)}\right\}\right) + \frac{\overline{u_{ij}^{(n)}}}{\overline{\lambda}_j}. \end{aligned} \tag{4.16}$$

Note that z is the random variable sampled during the forward pass due to the reparameterization in (4.14) and the gradients with respect to the log-likelihood are computed using standard automatic differentiation. We describe the gradients for the other three tensor formats in Appendix B.

- **Half Step 2: Incremental closed-form update for $\overline{\boldsymbol{\lambda}}$.** We analytically update the rank-controlling parameters $\boldsymbol{\lambda}$ based on the results in (4.11) and (4.13). We found empirically that incremental updates, rather than direct assignment of the results from (4.11) or (4.13), led to better performance. Therefore we adopt an incremental update strategy with learning rate γ for the rank parameter updates:

$$\overline{\lambda}_k \leftarrow \gamma \overline{\lambda}_k^{\star} + (1-\gamma)\overline{\lambda}_k. \tag{4.17}$$

As shown in Figure 6, this proposed hybrid parameter update can greatly reduce the gradient variance of tensor factors.

4.5. Algorithm flow and implementation issues.

The full description of our end-to-end tensorized training with rank determination is shown in Algorithm 4.1. We iteratively repeat the hybrid parameter updates for a predetermined number of epochs m. In the following, we discuss some important implementation issues.

Warmup schedule. A general challenge in Bayesian tensor computation is that poor initializations can lead to excessive rank shrinkage and trivial rank-zero solutions. In linear tensor problems such as tensor completion the SVD is used to generate high-quality initializations [55, 56]. For nonlinear tensorized neural networks we randomly initialize the factor matrices so the predictive accuracy is low and the KL divergence to the prior may dominate the local loss landscape around the initialization point. To avoid trivial rank-zero local optima early in the training process, we incrementally reweight the KL divergence from the variational approximation to the prior during the training process. Let e_w be the number of warmup training epochs and e be the current epoch. We reweight the KL divergence from the variational approximation to the prior by a factor β defined by

$$\beta = \min\left(1, \frac{e}{e_w}\right) \tag{4.18}$$

Algorithm 4.1. SVI-Based Tensorized Training with Rank Determination

```
Input: Factor learning rate $\alpha$, expectation maximization stepsize $\gamma$, rank cutoff $\epsilon$, warmup epochs $e_w$, total epochs $m$, tuning epochs $t$
for Epoch $e$ in $[1,\dots,m]$ do
  Assign $\beta$ according to (4.18).
  for each batch $\mathcal{B} \subset \mathcal{D}$ do
    Update the low-rank factor distribution variational parameters as in Half Step 1, (4.15).
    Update the rank-control hyperparameters as in Half Step 2, (4.17).
  end for
end for
Prune tensor ranks as described in (4.20).
```

and update the loss from (4.6) accordingly:

$$\mathcal{L}(q) = -\log\mathbb{E}_{q(\{\mathbf{U}^{(n)}\},\boldsymbol{\lambda})} p\left(\mathcal{D}|\left\{\mathbf{U}^{(n)}\right\}\right) + \beta\mathrm{KL}\left(q\left(\left\{\mathbf{U}^{(n)}\right\},\boldsymbol{\lambda}\right)||p\left(\left\{\mathbf{U}^{(n)}\right\},\boldsymbol{\lambda}\right)\right). \tag{4.19}$$

Gradually increasing the weight of the KL divergence to the prior avoids early local optima in which all ranks shrink to zero. We have found empirically that $e_w = m/2$ is a good choice for the number of warmup steps.

Rank pruning. After we run our Bayesian solver we truncate the ranks with variance $\overline{\lambda}_k$ below a prespecified threshold ϵ. For example, for the CP format if $\overline{\lambda}_k < \epsilon$ we assign

$$\overline{u_{ik}^{(n)}} \leftarrow 0 \text{ and } \Sigma_{ik}^{(n)} \leftarrow 0 \quad \text{for } 1 \leq n \leq d, 1 \leq i \leq I_n. \tag{4.20}$$

The associated kth column of $\mathbf{U}^{(n)}$ is removed, leading to a rank shrinkage and automatic model parameter reduction.

5. Experiments.

We demonstrate the applications of our rank-adaptive tensorized end-to-end training method on several neural network models. Our method trains a Bayesian neural network; therefore, we report the predictive accuracy of the posterior mean. In order to compare the performance, we implement the following methods in our experiments:

- **Baseline.** A standard training method, where model parameters are uncompressed.
- **TC-MR [27, 33].** Train and then compress with maximum ranks. We train a uncompressed neural network with the "baseline" method, followed by a tensor decomposition and fine-tuning. For the DLRM we fine-tune for one epoch. In all other experiments we fine-tune for 20 epochs. This approach requires that the user select the compression rank. Here we use the maximum rank used in our Bayesian model. This approach has been studied for computer vision tasks using the CP decomposition in [33] and the Tucker decomposition in [14, 27]. We compare against the algorithms of [27, 33] but on different architectures.
- **TC-OR.** Train and then compress with oracle rank (r in CP or $\mathbf{r} = [r_1, r_2, \dots, r_d]$ for other formats). This method follows the same procedures of TC-MR [27, 33], except that it uses the "oracle rank" discovered by our proposed rank determination method.

In practice this “TC-OR” method would require a combinatorial rank search over a high-dimensional discrete space to discover the same rank as our method.

- **FR.** Fixed-rank tensorized training. We implement tensorized training [3, 9, 23, 41] with a tensor rank fixed a priori. Determining the tensor ranks is challenging in this approach. In our experiments we reuse the well-tuned parameters from previous literature. The convolutional neural network experiment and architecture in the supplemental material are taken from [9]. The natural language processing (NLP) and DLRM experiment architectures are taken from [23].
- **ARD-LU.** The first version of our proposed tensorized training method with automatic rank determination. We use the LU prior in (3.12) for the rank-control hyperparameters. All tensor factors are initialized with a maximum rank (R for CP and $\mathbf{R} = [R_, \dots, R_d]$ for other formats), and the actual ranks (r for CP and $\mathbf{r} = [r_1, \dots r_d]$ for other formats) are automatically determined by our training process. To compare our method with FR, we set the maximum rank to the rank used in FR.
- **ARD-HC.** The second version of our proposed training method using the HC prior (3.11) for the rank-control hyperparameters.

As shown in Table 1 our proposed methods enjoy all of the listed advantages compared with other methods. The proposed automatic tensor rank determination avoids the expensive multiple training runs in FR, and it also results in the (almost) smallest models for inference. We consider four low-rank tensor formats for each tensorized method. Therefore, our experiments involve the implementation of 21 specific methods in total (20 tensorized implementations plus one baseline method). For all experiments we list the full tensor dimension and rank settings in the supplement. For all experiments we set the rank parameter learning rate $\gamma = 0.9$.

Remark 5.1. In our Bayesian training, every tensorized model parameter is equipped with two training variables (i.e., posterior mean and variance). Therefore the number of training variables is 2× that of the tensorized model parameter numbers. This parameter overhead in Bayesian training brings in the capability of uncertainty quantification in output prediction, which is important for safety-critical applications. Our Bayesian model also allows a pointwise maximum a posteriori training. In such training, the only additional parameters required are the rank-control parameters, so the number of training variables is only slightly larger than the number of training variables in fixed-rank tensorized training.

5.1. Synthetic example for rank determination.

First we test the ability of our proposed method to infer the tensor rank of model parameters in a neural network. For each tensor

Table 1

Summary of different training methods.

Method	Memory cost of training	# training runs	Model size for inference
Baseline	high	1	huge
FR [41]	low	many	small
TC-MR [33]	high	1	small
TC-OR [33]	high	1	small
ARD-LU (Proposed)	low	1	small
ARD-HC (Proposed)	low	1	small

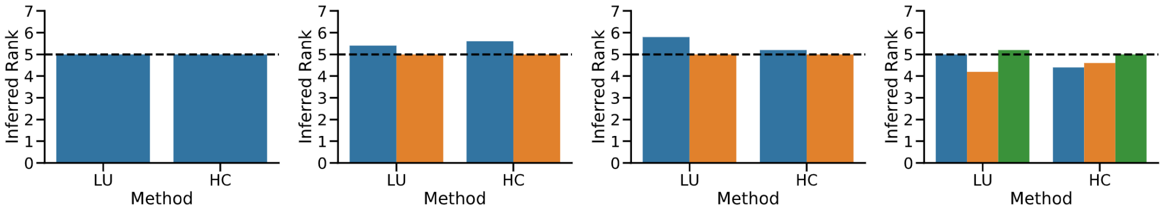
format we construct a synthetic version of the Modified National Institute of Standards and Technology (MNIST) dataset using a one-layer tensorized neural network (equivalent to tensorized logistic regression). The tensorized layer is fully connected, and the fixed tensor rank is five for each tensor format: 5 for CP, [5, 5, 5] for Tucker and [1, 5, 5, 1] for TT/TTM (Table 2). We use the rank-5 model to generate synthetic labels for the MNIST images. Then we train a set of low-rank tensorized models with a maximum rank of 10 on the synthetic dataset. For the CP, TT, and Tucker formats we reshape the weight matrix $\mathbf{W} \in \mathbb{R}^{784\times 10}$ into a tensor of shape size $[28, 28, 10]$ (i.e., an order-3 tensor of size $28 \times 28 \times 10$). For the TTM format we use the dimensions $[4, 7, 4], [7, 2, 5]$.

We plot the mean inferred ranks for our LU and HC priors in Figure 7. The actual CP rank is exactly recovered in our model. The inferred ranks of Tucker, TT, and TTM are close to but not equal to the exact values, because tensor ranks are not unique, which is a fundamental difference between matrices and tensors.

5.2. MNIST.

Next we test a neural network with two fully connected layers on the MNIST dataset with images of size 28×28. The first fully connected layer is size 784×512 and has a rectified linear unit activation function. The second fully connected layer is size 512×10 with a softmax activation function. Exact tensor dimensions are given in Table 2 of Appendix D. In all cases our automatic rank determination can achieve the highest compression ratio in training. Our proposed automatic rank determination both improves accuracy and reduces parameter number in all tensor formats except the TT format which has slight accuracy loss but the highest compression ratio. We hypothesize that the automatic rank reduction can reduce overfitting on the simple MNIST task. The TTM format is best-suited to fully connected layers, achieving the second-highest compression ratios and the second-best accuracy. In Figure 8 we plot the rank determination output of a single training run using our LU prior. We note that our algorithm discovers the actual ranks that are nearly impossible to determine via hand-tuning or combinatorial search (for example, [1,20,3,2,1] in the TTM model from a maximum rank of [1,20,20,20,1], which may require up to 16,000 searches).

With the obtained Bayesian solution, we can quantify the uncertainty of our model as a by-product. Popular metrics for uncertainty measures include negative log-likelihood, expected calibration error, which measures model over-/underconfidence, and out-of-distribution input detection [32]. In Figure 9, we show the classification uncertainty of an image that is hard to recognize in practice. With the CP tensorized model trained from ARD-LU, we plot the



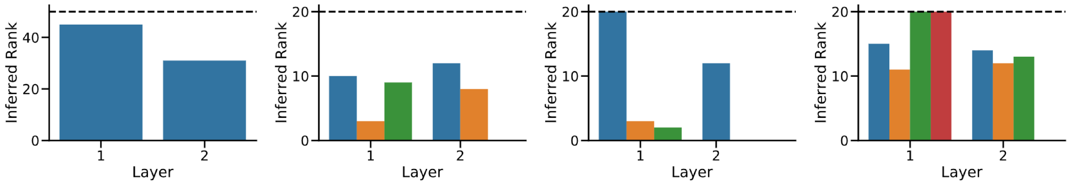
(a) CP format (b) TT format (c) TTM format (d) Tucker format

Figure 7. *The inferred ranks for a synthetic example. The true rank (dashed lines) is* 5*, and maximum rank is set to* 10*. The inferred ranks of different modes are given by colored bars.*

Table 2
Training results of the MNIST example.

Tensor format	Model	Training parameter #	Final parameter #	Accuracy
	Baseline	407,050	407,050	98.09
CP	FR	8,622 (47.2×)	8,622 (47.2×)	97.52
	TC-MR [33]	407,050 (1×)	8,622 (47.2×)	97.32
	TC-OR [33]	407,050 (1×)	7,175 (56.7×)	97.36
	ARD-LU (Proposed)	17,344 (23.5×)	7,175 (56.7×)	98.06
	ARD-HC (Proposed)	17,344 (23.5×)	7,134 (57.1×)	97.98
Tucker	FR [3]	171,762 (2.4×)	171,762 (2.4×)	97.93
	TC-MR [27]	407,050 (1×)	171,762 (2.4×)	98.00
	TC-OR [27]	407,050 (1×)	100,758 (4.0×)	97.91
	ARD-LU (Proposed)	343,644 (1.18×)	100,758 (4.0×)	98.30
	ARD-HC (Proposed)	343,644 (1.18×)	91,332 (4.5×)	98.30
TT	FR [41]	26,562 (13.9×)	26,562 (15.3×)	97.78
	TC-MR	407,050 (1×)	26,562 (15.3×)	97.43
	TC-OR	407,050 (1×)	4,224 (96.4×)	96.91
	ARD-LU (Proposed)	53,224 (7.65×)	4,224 (96.4×)	96.28
	ARD-HC (Proposed)	53,224 (7.65×)	4,276 (95.2×)	97.04
TTM	FR [41]	29,242 (13.9×)	29,242 (13.9×)	98.06
	TC-MR	407,050 (1×)	29,242 (13.9×)	97.47
	TC-OR	407,050 (1×)	6,144 (66.3×)	96.61
	ARD-LU (Proposed)	58,564 (6.95×)	6,144 (66.3×)	98.24
	ARD-HC (Proposed)	58,564 (6.95×)	5,200 (78.3×)	98.23

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance, so the training parameter number is 2× of that in FR. The results of FR rely on manual rank tuning in contrast to our automatic rank determination procedure.



(a) CP format (b) TT format (c) TTM format (d) Tucker format

Figure 8. *Inferred ranks for one run of the MNIST experiment using an LU prior. The maximum rank is given by a dashed black line. The inferred ranks are given by colored bars.*

mean and variance of the predicted softmax outputs in Figure 9(b). This plot clearly shows that this image looks like “2,” “3,” or “7” with the highest probability of being classified as “7.” Figure 9(c) further plots the marginal predictive density of the two most likely labels “2” and “7.”

5.3. Embedding table for NLP. We continue to validate our algorithm with a sentiment classification task from [26]. Like many NLP models, the first layer is a large embedding table. Embedding tables are a promising target for tensor compression because their required input dimension equals the number of unique tokens in the input dataset (i.e., number of vocabulary words, number of users). Tensor decomposition can enforce weight sharing and dramatically

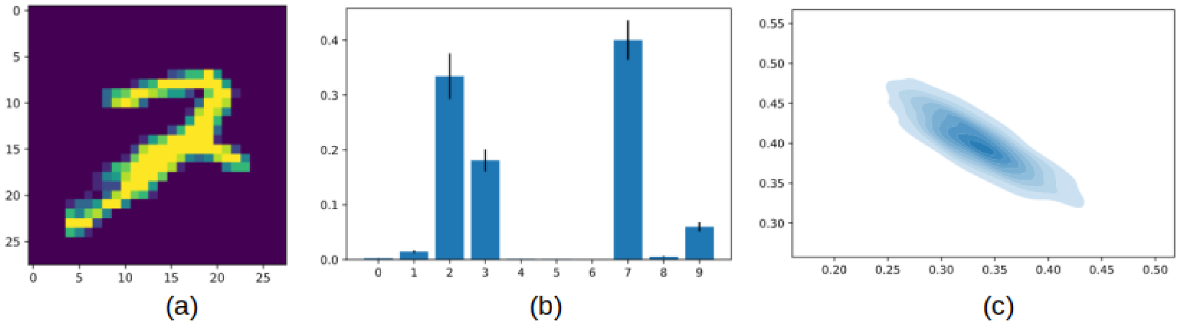


Figure 9. (a) *A challenging MNIST image with true label "2."* (b) *Mean and standard deviation of the CP ARD-LU model softmax outputs.* (c) *Marginal predictive density of the two most likely labels "2" (x-axis) and "7" (y-axis).*

reduce the parameter count of these models. Recent work in tensorized neural networks has applied the TTM format to compress large embedding tables with a high ratio [26]. We replicate a sentiment classification model on the IMDB dataset from this work. The neural network model consists of an embedding table with dimension $25,000 \times 256$, two bidirectional long short-term memory layers with hidden unit size 128, and a fully connected layer with 256 hidden units. Following the setting in [26] we do not tensorize these layers. Dropout masks are applied to the output of each layer except the last. Exact tensor dimensions are given in Table 3 of Appendix D.

We test all methods on the sentiment classification problem. The tensor dimensions and maximum ranks used to compress the embedding table are given in the supplementary material (supplement.pdf [local/web 329KB]). The outcomes of our experiments are reported in Table 3. Compared with all other tensor approaches, our methods (ARD-LU and ARD-HC) have achieved the best compression ratio for all tensor formats at little to no accuracy cost. The TTM format outperforms all other models (including the baseline uncompressed model) in terms of accuracy, though we note that the CP model performs well despite its extremely low parameter number.

5.4. DLRM system. We continue to use our proposed Bayesian tensorized method to train the benchmark DLRM [38]. In DLRM, embedding tables are used to process categorical features, while continuous features are processed with a bottom multilayer perceptron. Then, second-order interactions of different features are computed explicitly. The results are processed with a top multilayer perceptron and fed into a sigmoid function in order to give a probability of a click. The whole model has over 4 billion training variables.

We tensorize the five largest embedding tables to reduce the training variables. Exact tensor dimensions are given in Table 4 of Appendix D. Our experiment results are reported in Table 4. Our proposed automatic rank reduction enables parameter reduction at little to no accuracy cost over fixed-rank tensorized training. Our approach outperforms the train-then-compress approach which requires expensive full-model training. Compared with baseline full-size training, our method achieves to up to $27,664\times$ (in TT format) parameter reduction during training with little accuracy loss. Our one-shot training also greatly increases the compression ratio over fixed-rank training at little to no accuracy cost, enabling up to $7\times$ higher compression ratios in the TTM model.

Table 3
Training results on the NLP embedding table.

Tensor type	Model	Training parameter #	Final model parameter #	Accuracy
	Baseline	6,400,000	6,400,000	88.34
CP	FR	8,276 (774×)	8,276 (774×)	87.44
	TC-MR	6,400,000 (1×)	8,276 (774×)	74.46
	TC-OR	6,400,000 (1×)	6,138 (1024×)	73.21
	ARD-LU (Proposed)	16,602 (385×)	6,138 (1024×)	87.61
	ARD-HC (Proposed)	16,602 (385×)	6,476 (998×)	87.54
Tucker	FR	78,540 (81×)	78,540 (81×)	87.80
	TC-MR	6,400,000 (1×)	78,540 (81×)	75.12
	TC-OR	6,400,000 (1×)	61,920 (103×)	71.97
	ARD-LU (Proposed)	157,105 (40×)	61,920 (103×)	87.79
	ARD-HC (Proposed)	157,105 (40×)	58,120 (110×)	88.01
TT	FR [23]	28,260 (226×)	28,260 (226×)	85.6
	TC-MR	6,400,000 (1×)	28,260 (226×)	82.34
	TC-OR	6,400,000 (1×)	22,982 (278×)	71.81
	ARD-LU (Proposed)	56,640 (113×)	22,982 (278×)	85.33
	ARD-HC (Proposed)	56,640 (113×)	19,363 (331×)	85.82
TTM	FR [23]	22,312 (287×)	22,312 (287×)	88.59
	TC-MR	6,400,000 (1×)	22,312 (287×)	83.79
	TC-OR	6,400,000 (1×)	15,932 (402×)	84.83
	ARD-LU (Proposed)	44,724 (143×)	15,932 (402×)	88.93
	ARD-HC (Proposed)	44,724 (143×)	14,275 (448×)	88.78

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance of each tensorized model parameter. The results of FR rely on manual rank tuning in contrast to our automatic rank determination procedure.

The train-then-compress approach can be expensive for this large-scale problem. Because the trained embedding tables are extremely large, compressing them in Tucker or CP format is computationally expensive and time-consuming. This challenge can be avoided in our end-to-end-training approaches because we do not need to explicitly form the embedding tables.

5.5. Impact: On-device training and FPGA acceleration. We have demonstrated that our method can successfully train large end-to-end tensor compressed neural networks and increase the compression ratio during training. End-to-end compressed training has a major impact on embedded device training by reducing off-chip memory reads which are an energy and latency bottleneck [47]. In [53] FPGA acceleration of our method demonstrates 123× gains in energy efficiency and 59× speedup over nontensorized training on embedded device CPU. These latency and efficiency gains show how our method enables practical on-device training of compact neural networks from scratch.

6. Conclusion and future work. This work has proposed a variational Bayesian method for one-shot end-to-end training of tensorized neural networks. Our work has addressed the fundamental challenge of automatic rank determination, which is important for training compact neural network models on resource-constrained hardware platforms. The customized SVI method developed in this paper enables us to train tensorized neural networks with billions

Table 4

Training results on the DLRM embedding tables.

Tensor type	Model	Training parameter #	Final model parameter #	Accuracy
	Baseline	4,248,739,968	4,248,739,968	78.75
CP	FR	1,141,597 (3,721×)	1,141,597 (3,721×)	78.60
	TC-MR	4,248,739,968 (1×)	1,141,597 (3,721×)	75.41
	TC-OR	4,248,739,968 (1×)	563,839 (7,535×)	74.92
	ARD-LU (Proposed)	2,284,844 (1860×)	563,839 (7,535×)	78.61
	ARD-HC (Proposed)	2,284,844 (1860×)	570,685 (7,444×)	78.57
Tucker	FR	1,131,212 (3,755×)	1,131,212 (3,755×)	78.60
	TC-MR	4,248,739,968 (1×)	1,131,212 (3,755×)	78.67
	TC-OR	4,248,739,968 (1×)	436,579 (9,731×)	78.50
	ARD-LU (Proposed)	2,262,852 (1,877×)	436,579 (9,731×)	78.64
	ARD-HC (Proposed)	2,262,852 (1,877×)	402,023 (10,568×)	78.62
TT	FR [23]	1,135,752 (3,740×)	1,135,752 (3,740×)	78.68
	TC-MR	4,248,739,968 (1×)	1,135,752 (3,740×)	78.39
	TC-OR	4,248,739,968 (1×)	153,582 (27,664×)	78.45
	ARD-LU (Proposed)	2,271,864 (1870×)	153,582 (27,664×)	78.67
	ARD-HC (Proposed)	2,271,864 (1870×)	159,529 (26,633×)	78.63
TTM	FR [23]	1,130,048 (3759×)	1,130,048 (3759×)	78.73
	TC-MR	4,248,739,968 (1×)	1,130,048 (3759×)	78.43
	TC-OR	4,248,739,968 (1×)	199,504 (21, 296×)	78.62
	ARD-LU (Proposed)	2,260,256 (1879×)	199,504 (21,296×)	78.72
	ARD-HC (Proposed)	2,260,256 (1879×)	163,976 (25,910×)	78.73

Note: the training parameters in ARD-LU and ARD-HC include posterior mean and variance of every tensorized model parameters, so the number of training variables is 2× of that in FR. The results of FR rely on manual rank tuning in contrast to our automatic rank determination procedure.

of uncompressed model parameters. Our experiments have demonstrated that the proposed end-to-end tensorized training can reduce the training variables by several orders of magnitude. Our proposed method has outperformed all existing tensor compression methods on the tested benchmarks in terms of both compression ratios and predictive accuracy.

This work will enable ultra–memory- and energy-efficient training of artificial intelligence models on resource-constraint computing platforms, as demonstrated by our preliminary on-FPGA tensorized training in [53]. We will further investigate the theoretical and algorithm/hardware co-design issues in this direction, especially for training large-size neural networks on resource-constraint computing platforms.

REFERENCES

[1] J. M. Alvarez and M. Salzmann, *Compression-aware training of deep networks*, in Proceedings of the Conference on Neural Information Processing Systems, 2017, pp. 856–867.

[2] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, *Weight uncertainty in neural network*, in Proceedings of the International Conference on Machine Learning, 2015, pp. 1613–1622.

[3] G. G. Calvi, A. Moniri, M. Mahfouz, Q. Zhao, and D. P. Mandic, *Compression and Interpretability of Deep Neural Networks via Tucker Tensor Layer*, preprint, arXiv:1903.06133 [cs.LG], 2019.

[4] J. D. Carroll and J.-J. Chang, *Analysis of individual differences in multidimensional scaling via an N-way generalization of "Eckart-Young" decomposition*, Psychometrika, 35 (1970), pp. 283–319.

[5] C. M. Carvalho, N. G. Polson, and J. G. Scott, *The horseshoe estimator for sparse signals*, Biometrika, 97 (2010), pp. 465–480.

[6] C. Cui, C. Hawkins, and Z. Zhang, *Tensor methods for generating compact uncertainty quantification and deep learning models*, in Proceedings of the International Conference on Computer-Aided Design, 2019, pp. 1–6.

[7] C. Deng, F. Sun, X. Qian, J. Lin, Z. Wang, and B. Yuan, *TIE: Energy-efficient tensor train-based inference engine for deep neural network*, in Proceedings of the ACM/IEEE International Symposium on Computer Architecture , 2019, pp. 264–278.

[8] S. Gandy, B. Recht, and I. Yamada, *Tensor completion and low-n-rank tensor recovery via convex optimization*, Inverse Problems, 27 (2011), 025010.

[9] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, *Ultimate Tensorization: Compressing Convolutional and FC Layers Alike*, preprint, arXiv:1611.03214 [cs.LG], 2016.

[10] S. Ghosh, J. Yao, and F. Doshi-Velez, *Model selection in Bayesian neural networks via horseshoe priors*, J. Mach. Learn. Res., 20 (2019), pp. 1–46.

[11] D. Goldfarb and Z. Qin, *Robust low-rank tensor recovery: Models and algorithms*, SIAM J. Matrix Anal. Appl., 35 (2014), pp. 225–253.

[12] R. Guhaniyogi, S. Qamar, and D. B. Dunson, *Bayesian tensor regression*, J. Mach. Learn. Res., 18 (2017), pp. 2733–2763.

[13] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, *Deep learning with limited numerical precision*, in Proceedings of the International Conference on Machine Learning, 2015, pp. 1737–1746.

[14] J. Gusak, M. Kholiavchenko, E. Ponomarev, L. Markeeva, I. Oseledets, and A. Cichocki, *MUSCO: Multi-Stage Compression of Neural Networks*, preprint, arXiv:1903.09973 [cs.LG], 2019.

[15] S. Han, H. Mao, and W. J. Dally, *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*, preprint, arXiv:1510.00149 [cs.CV], 2015.

[16] S. J. Hanson and L. Y. Pratt, *Comparing biases for minimal network construction with backpropagation*, in Proceedings of the Conference on Neural Information Processing Systems, 1989, pp. 177–185.

[17] R. A. Harshman and M. E. Lundy, *PARAFAC: Parallel factor analysis*, Comput. Statist. Data Anal., 18 (1994), pp. 39–72.

[18] C. Hawkins and Z. Zhang, *Variational Bayesian inference for robust streaming tensor factorization and completion*, in Proceedings of the 2018 IEEE International Conference on Data Mining, IEEE, 2018, pp. 1446–1451.

[19] C. Hawkins and Z. Zhang, *Bayesian tensorized neural networks with automatic rank selection*, Neurocomputing, 453 (2021), pp. 172–180.

[20] Z. He, S. Gao, L. Xiao, D. Liu, H. He, and D. Barber, *Wider and deeper, cheaper and faster: Tensorized LSTMs for sequence learning*, in Proceedings of the Conference on Neural Information Processing Systems, 30 (2017), pp. 1–11.

[21] G. Hinton, O. Vinyals, and J. Dean, *Distilling the Knowledge in a Neural Network*, preprint, arXiv:1503.02531 [stat.ML], 2015.

[22] M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley, *Stochastic variational inference*, J. Mach. Learn. Res., 14 (2013), pp. 1303–1347.

[23] O. Hrinchuk, V. Khrulkov, L. Mirvakhabova, E. Orlova, and I. Oseledets, *Tensorized embedding layers*, in Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, 2020, pp. 4847–4860.

[24] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, *Quantized neural networks: Training neural networks with low precision weights and activations*, J. Mach. Learn. Res., 18 (2017), pp. 6869–6898.

[25] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, *An introduction to variational methods for graphical models*, Mach. learn., 37 (1999), pp. 183–233.

[26] V. Khrulkov, O. Hrinchuk, L. Mirvakhabova, and I. Oseledets, *Tensorized Embedding Layers for Efficient Model Compression*, preprint, arXiv:1901.10787 [cs.CL], 2019.

[27] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, *Compression of Deep Convolutional Neural Networks for Fast and Low Power Mobile Applications*, preprint, arXiv:1511.06530 [cs.CV], 2015.

[28] A. Kolbeinsson, J. Kossaifi, Y. Panagakis, A. Bulat, A. Anandkumar, I. Tzoulaki, and P. M. Matthews, *Tensor dropout for robust learning*, IEEE J. Sel. Top. Signal Process., 15 (2021), pp. 630–640.

[29] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.

[30] J. Kossaifi, A. Toisoul, A. Bulat, Y. Panagakis, T. M. Hospedales, and M. Pantic, *Factorized higher-order CNNs with an application to spatio-temporal emotion estimation*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 6060–6069.

[31] U. Köster, T. Webb, X. Wang, M. Nassar, A. K. Bansal, W. Constable, O. Elibol, S. Gray, S. Hall, L. Hornof, et al., *Flexpoint: An adaptive numerical format for efficient training of deep neural networks*, in Proceedings of the Conference on Neural Information Processing Systems, 2017, pp. 1742–1752.

[32] B. Lakshminarayanan, A. Pritzel, and C. Blundell, *Simple and scalable predictive uncertainty estimation using deep ensembles*, in Proceedings of the Conference on Neural Information Processing Systems, 2017, pp. 6402–6413.

[33] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, *Speeding-Up Convolutional Neural Networks Using Fine-Tuned CP-Decomposition*, preprint, arXiv:1412.6553 [cs.CV], 2014.

[34] Y. LeCun, J. S. Denker, and S. A. Solla, *Optimal brain damage*, in Proceedings of the Conference on Neural Information Processing Systems, 1990, pp. 598–605.

[35] Q. Liu and D. Wang, *Stein variational gradient descent: A general purpose Bayesian inference algorithm*, in Proceedings of the Conference on Neural Information Processing Systems, 2016, pp. 2378–2386.

[36] X. Ma, P. Zhang, S. Zhang, N. Duan, Y. Hou, M. Zhou, and D. Song, *A tensorized transformer for language modeling*, in Proceedings of the Conference on Neural Information Processing Systems, 2019, pp. 2232–2242.

[37] S. Nakajima and M. Sugiyama, *Analysis of empirical MAP and empirical partially Bayes: Can they be alternatives to variational Bayes?*, in Proceedings of the International Conference on Artificial Intelligence and Statistics, 2014, pp. 20–28.

[38] M. Naumov, D. Mudigere, H.-J. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C.-J. Wu, A. G. Azzolini, et al., *Deep Learning Recommendation Model for Personalization and Recommendation Systems*, preprint, arXiv:1906.00091 [cs.IR], 2019.

[39] R. M. Neal, *Bayesian Learning for Neural Networks*, Springer Science & Business Media, New York, 2012.

[40] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, *Structured Bayesian pruning via log-normal multiplicative noise*, in Proceedings of the Conference on Neural Information Processing Systems, 2017, pp. 6775–6784.

[41] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, *Tensorizing neural networks*, in Proceedings of the Conference on Neural Information Processing Systems, 2015, pp. 442–450.

[42] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput., 33 (2011), pp. 2295–2317.

[43] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arisoy, and B. Ramabhadran, *Low-rank matrix factorization for deep neural network training with high-dimensional output targets*, in proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, 2013, pp. 6655–6659.

[44] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, preprint, arXiv:1409.1556 [cs.CV], 2014.

[45] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in NLP*, in Proceedings of the Annual Meeting of the Association for Computational Linguistics, 2019, pp. 3645–3650.

[46] X. Sun, N. Wang, C.-Y. Chen, J. Ni, A. Agrawal, X. Cui, S. Venkataramani, K. El Maghraoui, V. V. Srinivasan, and K. Gopalakrishnan, *Ultra-low precision 4-bit training of deep neural networks*, in Proceedings of the Conference on Neural Information Processing Systems, 33 (2020).

[47] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, *Hardware for machine learning: Challenges and opportunities*, in Proceedings of the IEEE Custom Integrated Circuits Conference, 2017, pp. 1–8.

[48] S. Teerapittayanon, B. McDanel, and H.-T. Kung, *Distributed deep neural networks over the cloud, the edge and end devices*, in Proceedings of the International Conference on Distributed Computing Systems, 2017, pp. 328–339.

[49] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika, 31 (1966), pp. 279–311.

[50] M. P. Wand, J. T. Ormerod, S. A. Padoan, and R. Frühwirth, *Mean field variational Bayes for elaborate distributions*, Bayesian Anal., 6 (2011), pp. 847–900.

[51] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, *Learning structured sparsity in deep neural networks*, in Proceedings of the Conference on Neural Information Processing Systems, 29 (2016), pp. 2074–2082.

[52] J. Xue, J. Li, and Y. Gong, *Restructuring of deep neural network acoustic models with singular value decomposition*, in Proceedings of Interspeech, 2013, pp. 2365–2369.

[53] K. Zhang, C. Hawkins, X. Zhang, C. Hao, and Z. Zhang, *On-FPGA training with ultra memory reduction: A low-precision tensor method*, in Proceedings of the ICLR Workshop of Hardware Aware Efficient Training, (2021).

[54] K. Zhang, X. Zhang, and Z. Zhang, *Tucker tensor decomposition on FPGA*, in Proceedings of the International Conference on Computer-Aided Design, 2019, pp. 1–8.

[55] Q. Zhao, L. Zhang, and A. Cichocki, *Bayesian CP factorization of incomplete tensors with automatic rank determination*, IEEE Trans. Pattern Anal. Mach. Intell., 37 (2015), pp. 1751–1763.

[56] Q. Zhao, G. Zhou, L. Zhang, A. Cichocki, and S.-I. Amari, *Bayesian robust tensor factorization for incomplete multiway data*, IEEE Trans. Neural Netw. Learn. Syst., 27 (2016), pp. 736–748.

[57] P. Zhen, B. Liu, Y. Cheng, H.-B. Chen, and H. Yu, *Fast video facial expression recognition by deeply tensor-compressed ISTM neural network on mobile device*, in Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, 2019, pp. 298–300.

[58] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, *Incremental Network Quantization: Towards Lossless CNNs with Low-Precision Weights*, preprint, arXiv:1702.03044 [cs.CV], 2017.

[59] H. Zhou, L. Li, and H. Zhu, *Tensor regression with applications in neuroimaging data analysis*, J. Amer. Statist. Assoc., 108 (2013), pp. 540–552.