ORIGINAL RESEARCH
published: 07 January 2022
doi: 10.3389/frai.2021.668353

General-Purpose Bayesian Tensor Learning With Automatic Rank Determination and Uncertainty Quantification

Kaiqi Zhang [1†], Cole Hawkins [2†] and Zheng Zhang [1]*

[1] *Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, United States,* [2] *Department of Mathematics, University of California, Santa Barbara, Santa Barbara, CA, United States*

A major challenge in many machine learning tasks is that the model expressive power depends on model size. Low-rank tensor methods are an efficient tool for handling the curse of dimensionality in many large-scale machine learning models. The major challenges in training a tensor learning model include how to process the high-volume data, how to determine the tensor rank automatically, and how to estimate the uncertainty of the results. While existing tensor learning focuses on a specific task, this paper proposes a generic Bayesian framework that can be employed to solve a broad class of tensor learning problems such as tensor completion, tensor regression, and tensorized neural networks. We develop a low-rank tensor prior for automatic rank determination in nonlinear problems. Our method is implemented with both stochastic gradient Hamiltonian Monte Carlo (SGHMC) and Stein Variational Gradient Descent (SVGD). We compare the automatic rank determination and uncertainty quantification of these two solvers. We demonstrate that our proposed method can determine the tensor rank automatically and can quantify the uncertainty of the obtained results. We validate our framework on tensor completion tasks and tensorized neural network training tasks.

Keywords: deep learning, tensor decomposition, tensor learning, Bayesian inference, uncertainty quantification

OPEN ACCESS

Edited by:
Jean Kossaifi,
Nvidia, United States

Reviewed by:
Julia Gusak,
Skolkovo Institute of Science and Technology, Russia
Jlahao Su,
University of Maryland, United States

****Correspondence:***
Zheng Zhang
zhengzhang@ece.ucsb.edu

†These authors have contributed equally to this work

Specialty section:
This article was submitted to Machine Learning and Artificial Intelligence, a section of the journal Frontiers in Artificial Intelligence

Received: *16 February 2021*
Accepted: *13 December 2021*
Published: *07 January 2022*

Citation:
Zhang K, Hawkins C and Zhang Z (2022) General-Purpose Bayesian Tensor Learning With Automatic Rank Determination and Uncertainty Quantification. Front. Artif. Intell. 4:668353. doi: 10.3389/frai.2021.668353

1. INTRODUCTION

Tensors (Kolda and Bader, 2009) are a generalization of matrices to describe and process multidimensional data arrays. Due to its ability to represent a huge amount of data by low-rank factorization, tensor computation has been applied in data recovery and compression (Acar et al., 2011; Jain and Oh, 2014; Austin et al., 2016), machine learning (Cichocki, 2014; Novikov et al., 2015; Sidiropoulos et al., 2017), uncertainty quantification (Zhang et al., 2014, 2016), and so forth. However, most of the existing tensor algorithms rely on numerical optimization, and estimating the tensor rank exactly is NP-Hard in some tensor formats (Hillar and Lim, 2013).

To overcome the rank determination challenge, Bayesian methods have been employed successfully in tensor completion tasks (Chu and Ghahramani, 2009; Xiong et al., 2010; Rai et al., 2014; Zhao et al., 2015a,c; Hawkins and Zhang, 2018; Gilbert and Wells, 2019). The key idea is to represent the tensor factors as some hidden statistical variables and to automatically determine the tensor ranks based on the training data and a proper rank-shrinking prior density. Chu and Ghahramani (2009) proposed a maximum a posteriori (MAP) MAP estimation, but a point prediction cannot estimate the uncertainty. In order to estimate model uncertainties, Gibbs sampling and mean-field approximate Bayesian methods have been employed in Xiong et al. (2010), Rai et al. (2014), and Zhao et al. (2015a,b), respectively. The former assumes that a conditional posterior density function can be obtained analytically and be sampled from easily. The latter assumes that the hidden parameters are mutually independent to each other. These methods work well in some simple tensor learning tasks such as tensor completion and factorization, but the may become over-simplified when solving more complicated problems such as tensorized neural networks.

Paper Contributions. This paper presents a Bayesian framework that is applicable to a broad class of tensor learning problems, e.g., tensor factorization/completion, tensor regression, and tensorized neural networks. Given a tensor learning model with a specified prior density, likelihood function and low-rank tensor approximation format, our framework estimates the posterior distribution of all the factors and automatically determine the tensor ranks via more flexible Bayesian inference methods such as Hamiltonian Monte Carlo (HMC) (Duane et al., 1987) and Stein Variational Gradient Descent (SVGD) (Liu and Wang, 2016). Compared with the mean-field Bayesian tensor completion (Zhao et al., 2015a,b), our tensor learning approach is more flexible because it does not require the strong assumption of independent hidden parameters. Further, our approach can be applied to highly non-linear tensor problems, i.e., tensorized neural networks. Due to the huge amount of training data in many tensor learning problems, estimating the full gradient can be computationally expensive. Therefore, we replace the full gradient in a tensor learning problem with the stochastic gradient (Chen et al., 2014) while achieving a similar level of accuracy. Because of that, our method has larger scalability than traditional Bayesian methods. Our HMC-based sampling approach to tensor learning returns a set of random samples following the posterior distribution, therefore, we can provide uncertainty estimations for both the model parameters and the predictive results. In certain situations HMC may require that the user store too many model copies for inference. For these situations we also develop an SVGD-based framework which applies deterministic updates to produce a small-sized set of particles to approximate the posterior distribution and approximate model uncertainty. We compare both approaches in our experiments. We note that sampling-based approaches have been employed in Bayesian neural networks (Neal, 1992; Liu and Wang, 2016) and data compression (Schmidt and Mohamed, 2009; Şimşekli et al., 2015). However, a thorough investigation of the generalized Bayesian tensor learning problem has not been reported. Our method have several advantages compared with existing methods: (1) **generic**–our method framework can deal with a broad class of tensor learning problems, including the tensor decomposition, tensor completion, tensor regression, tensorized neural networks, etc. (2) **scalable**–the stochastic-gradient implementation enables us to handle tensor learning problems with massive data. (3) **uncertainty-aware**–the resulting posterior samples provide uncertainty estimations for both the model parameters and predictive results.

This manuscript is an extended version of our recent work (Hawkins and Zhang, 2021), which reported SVGD training for Bayesian tensorized neural networks. Our manuscript extends Hawkins and Zhang (2021) in the following ways

1. In our previous work, we tested only one Bayesian sampler (SVGD). In this work, we compare two Bayesian samplers which have different memory/compute trade offs during both the training and inference stages of model deployment.
2. In our previous work, we considered only one tensor format (tensor-train) and one rank determination task (tensorized neural networks). In order to test the generality of our method we test our methods on both the tensor-train and Tucker formats and in two rank determination settings: tensor completion and tensorized neural networks.
3. In our previous work, the rank-threshold operation required a user-defined cutoff. We introduce a rank-thresholding cutoff that requires no user intervention.

2. GENERALIZED BAYESIAN TENSOR LEARNING

This section will present the model and numerical solver of our generic Bayesian tensor learning framework. We will demonstrate specific applications of our framework in later sections.

2.1. A Generalized Bayesian Model

We consider a generalized tensor learning problem: given a set of observed data $\mathcal{D} = \{D_1, D_2, \dots D_N\}$, we want to estimate the posterior density $p(\boldsymbol{\mathcal{X}}|\mathcal{D})$ of an unknown tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots I_d}$. Because $\boldsymbol{\mathcal{X}}$ has a huge number of unknown variables, directly solving the Bayesian tensor learning problem is computationally expensive.

Our framework allows various kinds of low-rank tensor representations (Kolda and Bader, 2009), such as CANDECOMP/PARAFAC (CP) (Harshman et al., 1970), tensor-train (TT) (Oseledets, 2011), and Tucker (Tucker, 1966). A low-rank tensor representation can significantly reduce the number of unknown variables. For instance, low-rank CP and tensor-train representations may reduce the number of unknowns from an exponential function of d to a linear one. In a general setting, we denote all model parameters (including the tensor factors and some additional hyper-parameters such as noise level or rank controlling parameters) as Θ, and the unknown tensor can be written as $\boldsymbol{\mathcal{X}}(\Theta)$. Then our goal is to estimate the posterior density

$$P(\Theta|\mathcal{D}) \propto \prod_{n=1}^{N} P(D_n|\Theta)P(\Theta). \tag{1}$$

Here $P(\mathcal{D}|\Theta) = \prod_{n=1}^{N} P(D_n|\Theta)$ is a likelihood function, $P(\Theta)$ is a prior probability density. A key advantage of this Bayesian parameterized description is as follows: by properly choosing a

Algorithm 1: SGHMC with thermostats

```
Input : The dataset $\mathcal{D}$, the potential $U(\Theta)$, the mass $\mathbf{M}$,
        Leapfrog steps per sample $m$, the maximal number
        of samples $T$
Initialize $\Theta$ by minimizing $U(\Theta)$ using SGD, Adam, etc.
for $t = 1, 2, \ldots T$ do
    Sample the momentum $\mathbf{p} \sim \mathcal{N}(0, \mathbf{M})$.
    Draw a mini-batch $\mathcal{B} \subset \mathcal{D}$ to formulate the unbiased
    potential function $\tilde{U}(\Theta)$ by equation (4).
    for $i = 1$ to $m$ do
        Update $\Theta, \mathbf{p}, c$ using (6)
    end
    $\Theta^{(t)} \leftarrow \Theta$
end
Output: The sample set of $\{\Theta^{(t)}\}_{t=1}^{T}$.
```

Algorithm 2: SVGD

```
Input : The dataset $\mathcal{D}$, the potential $U(\Theta)$, the maximal
        number of samples $n$, number of SVGD iterations $I$
Initialize $\Theta$ by minimizing $U(\Theta)$ using SGD, Adam, etc.
for $i = 1$ to $n$ do
    Update $\Theta$ by taking one stochastic gradient step.
    Initialize $\Theta^i \leftarrow \Theta$.
end
for $t = 1, 2, \ldots I$ do
    Draw a mini-batch $\mathcal{B} \subset \mathcal{D}$ to formulate the unbiased
    potential function $\tilde{U}(\Theta)$ by equation (4).
    for $i = 1$ to $n$ do
        Update $\Theta^i$ using (7)
    end
end
Output: The sample set of $\{\Theta^i\}_{i=1}^{n}$.
```

prior density $P(\Theta)$, one can control the structure of Θ and thus automatically enforce a low-rank representation for $\boldsymbol{\mathcal{X}}(\Theta)$ based on the observed data $\boldsymbol{\mathcal{D}}$. Doing so overcomes the difficulty of rank determination in optimization-based tensor learning.

The formulation (1) is very generic. In practice, one only needs to specify the following information in order to use our Bayesian tensor learning framework:

- The learning task, such as tensor completion, multi-task tensor learning, tensorized neural networks for classification or regression, etc;
- A low-rank parameterization format, such as CP, Tucker, tensor-train factorization, etc;
- A prior density $P(\Theta)$ for tensor factors and hyper-parameters.

The first two decide the likelihood function $P(\mathcal{D}|\Theta)$, and we will make it clear in section 3. The third decides how compact the resulting model would be: a stronger low-rank prior could result in a model with much fewer model parameters.

2.2. Stochastic Gradient HMC (SGHMC) Solver

Now we need to estimate the hidden tensor factors and hyper-parameters by computing the posterior density in (1). Existing methods (Zhao et al., 2015a; Hawkins and Zhang, 2018) do not apply to generalized tensor learning problems because they rely on Bayesian models that make strong assumptions about the posterior density and require linear models. The first Bayesian solver we employ is the Hamiltonian Monte Carlo (HMC) (Duane et al., 1987) to make our framework applicable to a broad class of tensor learning problems. HMC is an extension to Markov chain Monte Carlo (MCMC) Andrieu et al. (2003), and it uses the gradient information to increase efficiency.

The HMC method avoids the random walks in a standard MCMC framework by simulating the following dynamic system:

$$\frac{d\Theta}{dt} = \mathbf{M}^{-1}\mathbf{p}, \qquad \frac{d\mathbf{p}}{dt} = -\nabla U(\Theta). \tag{2}$$

Here $\mathbf{p}$ is the auxiliary momentum variable with the same dimension as Θ, $\mathbf{M}$ is a mass matrix. Here $U(\Theta)$ is the potential energy which is equal to the negative log posterior:

$$U(\Theta) = -\log P(\Theta|\mathcal{D}) = -\sum_{n=1}^{N} \log P(D_n|\Theta) - \log P(\Theta). \tag{3}$$

The HMC method starts from a random initial guess of Θ to simulate a sample of Θ, and its steady-state distribution converges to our desired posterior density $P(\Theta|\mathcal{D})$.

Standard HMC becomes inefficient when we solve a tensor learning problem with massive training samples, because computing the gradient requires estimating $\nabla \log P(D_n|\Theta)$ for every index n over the whole data set. This often happens in completing a huge-size tensor data set or training a tensorized neural network. To reduce the cost, we use the stochastic unbiased estimator of $U(\Theta)$:

$$\tilde{U}(\Theta) = -\frac{N}{|\mathcal{B}|} \sum_{D_i \in \mathcal{B}} \log P(D_i|\Theta) - \log P(\Theta) + \text{const.} \tag{4}$$

Here $\mathcal{B} \subset \mathcal{D}$ denotes a mini-batch with $|\mathcal{B}| \ll N$. Then one can update the parameters *via* $\frac{d\Theta}{dt} = \mathbf{M}^{-1}\mathbf{p}$ and $\frac{d\mathbf{p}}{dt} = -\nabla\tilde{U}(\Theta)$. To compensate the noise introduced by the stochastic gradient, we adopt the thermostats method (Ding et al., 2014) for our tensor learning framework. Specifically, a friction term c is introduced, i.e.,

$$\begin{aligned} \frac{d\Theta}{dt} &= \mathbf{M}^{-1}\mathbf{p}, \qquad \frac{d\mathbf{p}}{dt} = -\nabla\tilde{U}(\Theta) - c\mathbf{p}, \frac{dc}{dt} \\ &= \frac{1}{|\Theta|}\mathbf{tr}(\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p}) - 1. \end{aligned} \tag{5}$$

The friction term changes accordingly to keep the average kinetic energy $\frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p}$ constant, thus keeping the distribution of samples invariant. Our framework employs a slightly modified leapfrog approach Iserles (1986) to solve the Hamiltonian system because it has a smaller integration error compared with other methods Duane et al. (1987):

$$\begin{aligned}\mathbf{p}_{t+\epsilon/2} &\leftarrow \mathbf{p}_t - \tfrac{1}{2}\epsilon(\nabla\tilde{U}(\Theta_t) + c_t\mathbf{p}_t), \quad \Theta_{t+\epsilon} \leftarrow \Theta_t + \epsilon\mathbf{p}_{t+\epsilon/2},\\ \mathbf{p}_{t+\epsilon} &\leftarrow \mathbf{p}_{t+\epsilon/2} - \tfrac{1}{2}\epsilon(\nabla\tilde{U}(\Theta_{t+\epsilon}) + c_t\mathbf{p}_{t+\epsilon/2}),\\ c_{t+\epsilon} &\leftarrow c_t + \epsilon(\tfrac{1}{|\Theta|}\mathbf{tr}(\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p}) - 1),\end{aligned} \tag{6}$$

where ϵ is the stepsize, and t is the iteration index.

2.3. Stein Variational Gradient Descent (SVGD) Solver

The HMC solver described in the previous section can accurately represent arbitrary distributions given a sufficient sampling budget. The disadvantage of the HMC approach is that it may require a large number of model copies for accurate uncertainty quantification (Neal, 1992). Therefore, we also consider the Stein Variational Gradient Descent (SVGD) (Liu and Wang, 2016) to approximate the posterior density $p(\mathbf{\Theta}|\mathcal{D})$. SVGD uses a small number of particles to approximate a target distribution. The advantage of this approach compared to HMC is a lower memory cost due to a lower particle number. The disadvantage compared to HMC is a potential reduction in the accuracy of the final posterior representation.

SVGD aims to find a set of particles $\{\mathbf{\Theta}^i\}_{i=1}^n$ such that $q(\mathbf{\Theta}) = \frac{1}{n}\sum_{i=1}^n k(\mathbf{\Theta}, \mathbf{\Theta}^i)$ approximates the true posterior $p(\mathbf{\Theta}|\mathcal{D})$. Here $k(\cdot,\cdot)$ is a positive definite kernel, and we use the radial basis function kernel in this work. The particles can be found by minimizing the KL divergence between $q(\mathbf{\Theta})$ and $p\left(\mathbf{\Theta}|\mathcal{D}\right)$. The optimal update $\phi(\cdot)$ is derived in (Liu and Wang, 2016) and takes the form

$$\begin{aligned}\mathbf{\Theta}^k &\leftarrow \mathbf{\Theta}^k + \epsilon\phi(\mathbf{\Theta}^k), \quad \phi(\mathbf{\Theta}^k)\\ &= \frac{1}{n}\sum_{i=1}^n \left[k(\mathbf{\Theta}^i, \mathbf{\Theta}^k)\nabla_{\mathbf{\Theta}^i} U(\mathbf{\Theta}^i) + \nabla_{\mathbf{\Theta}^i} k(\mathbf{\Theta}^i, \mathbf{\Theta}^k)\right]\end{aligned} \tag{7}$$

where ϵ is the step size. The gradient of the potential function $\nabla_{\mathbf{\Theta}^i} U(\mathbf{\Theta}^i)$ can be approximated with a stochastic gradient $\nabla_{\mathbf{\Theta}^i} \hat{U}(\mathbf{\Theta}^i)$ when the data size $|\mathcal{D}|$ is large. The SVGD training update is, therefore, a deterministic function of the existing particle locations and minibatch gradients. The particle locations are incorporated through the kernel function $k(\cdot,\cdot)$ and the minibatch gradients of each individual particles are $\nabla_{\mathbf{\Theta}^i} U(\mathbf{\Theta}^{\boldsymbol{i}})$. To sample from the SVGD distribution during training or inference one computes a deterministic forward pass for each of the n particles $\{\mathbf{\Theta}\}_{i=1}^n$. At each step of the SVGD Bayesian learning process we require a deterministic forward/backward pass for each particle to compute the gradient in Equation (7). In practice we use ten particles to compute our posterior approximation. Unlike the HMC update, in which we re-sample the momentum parameter, no additional noise is introduced during the training process.

We make two observations about the computational requirements of SVGD. Due to the update in Equation (7) this method requires the computation of n gradients of the potential energy function U at each step. Second, each computation requires that we acesss n particles. Therefore, the per-step memory and compute costs of the SVGD solver are $n\times$ as large as the per-step costs of the HMC solver. The advantage of the SVGD model is the compact final representation for uncertainty quantification (Liu and Wang, 2016).

3. SPECIFYING POTENTIAL ENERGY FUNCTIONS

One can run our Bayesian tensor learning framework as a black-box after specifying the energy function based on three components: a learning task, a low-rank tensor representation format, and a prior density. In this section, we will first give the details of two cases: Bayesian tensor completion with a Gaussian likelihood and low-rank CP representation, and Bayesian tensorized neural network classification with a multinomial likelihood and a low-rank tensor-train representation. Then we will also briefly show the Bayesian models for some other cases.

3.1. Bayesian CP Tensor Completion

Given $\mathcal{D} = \{\Omega, \boldsymbol{\mathcal{Y}}_\Omega\}$ where $\boldsymbol{\mathcal{Y}}_\Omega$ denotes some partially observed noisy tensor elements and Ω specifies the sample indices, we aim to find a low-rank tensor $\boldsymbol{\mathcal{X}}$ such that $\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} + \boldsymbol{\mathcal{E}}$, where $\boldsymbol{\mathcal{E}}$ denotes a Gaussian noise tensor with zero mean and variance σ. We use the CP factorization (Harshman et al., 1970; Bro, 1997) to paramterize $\boldsymbol{\mathcal{X}}$:

$$\boldsymbol{\mathcal{X}} = \sum_{r=1}^R \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(d)} = [\![\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}]\!], \tag{8}$$

where $\circ$ denotes the outer product of vectors and $[\![\cdot]\!]$ denotes a Kruskal operator. In the Bayesian tensor learning, we need to estimate CP factors $\{\mathbf{A}^{(k)}\}_{k=1}^d$ and CP rank R. Following the Bayesian CP tensor completion (Zhao et al., 2015c), we set the hidden parameters as $\Theta = \{\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(d)}, \Lambda, \tau\}$, where hyper-parameters $\Lambda = \text{diag}(\lambda_1, \ldots, \lambda_R)$ and $\tau = 1/\sigma$ control the tensor ranks and noise level, respectively. The Gaussian noise assumption leads to the Gaussian likelihood

$$P(\boldsymbol{\mathcal{Y}}_\Omega|\Theta) = \mathcal{N}(\boldsymbol{\mathcal{Y}}_\Omega|[\![\mathbf{A}^{(1)}, \ldots \mathbf{A}^{(d)}]\!], \tau^{-1}) \tag{9}$$

where Ω denotes the observed tensor entries. The negative log likelihood associated with each observation is

$$-\log P(\boldsymbol{\mathcal{Y}}_i|\Theta) = \frac{1}{2}\left(\left(\boldsymbol{\mathcal{Y}}_i - f(i;\Theta)\right)^2\tau - \log\tau\right) + \text{const}, \tag{10}$$

where f denotes the forward evaluation from CP factors to the i-th element of tensor $\boldsymbol{\mathcal{X}}$.

Next, our goal is to develop a rank-shrinkage prior. The prior density will enforce structured sparsity on the CP factor matrices, leading to rank shrinkage. We define this rank-shrinkage prior density as

$$\begin{aligned} P(\Theta) &= \prod_{k=1}^{d}\prod_{i_k=1}^{I_k}\mathcal{N}(\mathbf{A}^{(k)}(i_k,:)|0,\Lambda^{-1}) \\ &\prod_{r=1}^{R}\text{Gamma}(\lambda_r|a,b)\text{Gamma}(\tau|c,d), \end{aligned} \tag{11}$$

We remark that the Gaussian prior on the factor matrices enforces that all factor matrix entries in the same column (same rank) share the same variance. Therefore, as the hyperparameter $\lambda_j \to \infty$ all entries in the columns $\{\mathbf{A}^{(k)}(:,j)\}$ shrink to 0.

This rank-shrinkage prior leads to the following negative log-priors:

$$\begin{aligned} -\log P(\mathbf{A}^{(k)}|\Lambda) &= \frac{1}{2}\sum_{r=1}^{R}\left(|\mathbf{A}^{(k)}(:,r)|^2\lambda_r - \sum_{k=1}^{d} I_k\log\lambda_r\right), \\ -\log P(\Lambda) &= \sum_{r=1}^{R} -(a-1)\log\lambda_r + b\lambda_r. \end{aligned} \tag{12}$$

The noise level τ may vary among different datasets, and λ_r can be very large for diminishing ranks. Therefore, we slightly modify the model to ensure the numerical stability in HMC-based Bayesian tensor learning. Instead of sampling τ and λ_r directly, we sample $\hat{\tau} = \log(\tau)$ and $\hat{\lambda}_r = \lambda_r^{-1}$ for better numerical stability, because these values can vary dramatically among different datasets. Therefore, we use the following prior distributions:

$$P(\hat{\tau}) = \frac{\exp(c\hat{\tau})\exp(-e^{\hat{\tau}})/d)}{d^c\mathcal{T}(c)},\ P(\hat{\lambda}) = \frac{b^a}{\mathcal{T}(a)}(1/\hat{\lambda})^{a+1}\exp(-b/\hat{\lambda}). \tag{13}$$

Combining equations (12) and (13), we have the modified negative log-prior

$$\begin{aligned} -\log P(\Theta) = \sum_{r=1}^{R}\Big(\frac{1}{2}\sum_{k=1}^{d}\Big(&|\mathbf{A}^{(k)}(:,r)|^2/\hat{\lambda}_r + I_k\log\hat{\lambda}_r\Big) \\ &+ (\alpha+1)\log\hat{\lambda}_r + \frac{\beta}{\hat{\lambda}_r}\Big) \\ &- c\hat{\tau} + \exp\hat{\tau}/d. \end{aligned} \tag{14}$$

Based on the above prior density and our given Gaussian noise model, the potential function (neglecting the constants) for the Bayesian CP tensor completion is

$$\begin{aligned} U(\Theta) &= -\log P(\Theta|\mathcal{Y}_\Omega) = -\log P(\Theta) - \sum_{i\in\Omega}\log P(\mathcal{Y}_i|\Theta) \\ &= \frac{1}{2}\left(\left\|\left(\mathcal{Y} - [\![\mathbf{A}^{(1)},\ldots,\mathbf{A}^{(d)}]\!]\right)_\Omega\right\|_F^2 \exp\hat{\tau} - |\Omega|\hat{\tau}\right) \\ &+ \sum_{r=1}^{R}\Big(\frac{1}{2}\sum_{k=1}^{d}\Big(|\mathbf{A}^{(k)}(:,r)|^2/\hat{\lambda}_r + I_k\log\hat{\lambda}_r\Big) \\ &+ (a+1)\log\hat{\lambda}_r + \frac{b}{\hat{\lambda}_r}\Big) - c\hat{\tau} + \exp\hat{\tau}/d. \end{aligned} \tag{15}$$

We note that in order to use other low-rank tensor formats, all that is necessary is a change in the prior density. We provide the specific prior densities for the tensor-train and Tucker formats in later sections.

3.2. Bayesian Tensorized Neural Networks

We further show how to apply our Bayesian tensor learning to train a tensorized deep neural network in the tensor-train (TT) format. Given the training data $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^{N}$, we want to find a low-rank tensor $\boldsymbol{\mathcal{W}}$ in the TT format to describe the weight matrices or convolution filters such that $\mathbf{y} = g(\mathbf{x}, \boldsymbol{\mathcal{W}})$, where g denotes the forward propagation model of a neural network.

For a weight matrix $\mathbf{W}$ of size $M \times J$, one can decompose $M = \prod_{k=1}^{K} m_k$ and $J = \prod_{k=1}^{K} j_k$, then reformulate $\mathbf{W}$ as a $2K$-dimension tensor $\boldsymbol{\mathcal{W}}$ with size $m_1 \times j_1 \times \cdots \times m_K \times j_K$. Afterwards, $\boldsymbol{\mathcal{W}}$ is approximated by a low-rank tensor-train decomposition

$$\begin{aligned} \boldsymbol{\mathcal{W}} &= [\![\boldsymbol{\mathcal{G}}^{(1)},\ldots,\boldsymbol{\mathcal{G}}^{(K)}]\!]_{TT} \Longleftrightarrow \boldsymbol{\mathcal{W}}(i_1,\cdots,i_K,l_1,\cdots,l_K) \\ &= \boldsymbol{\mathcal{G}}^{(1)}(:,i_1,l_1,:)\cdots\boldsymbol{\mathcal{G}}^{(K)}(:,i_K,l_K,:) \end{aligned} \tag{16}$$

where $\boldsymbol{\mathcal{G}}^{(k)} \in \mathbb{R}^{R_{k-1}\times m_k\times j_k\times R_k}$ is called the TT core, R_k is the TT rank, $R_0 = R_K = 1$, and $[\![\cdot]\!]_{TT}$ denotes the tensor-train product (Oseledets, 2011). The convolutional layers can be decomposed in a similar way. The convolution kernel $\boldsymbol{\mathcal{C}}$ is a 4-th dimension tensor in $M \times J \times H \times W$, where H and W denote the height and width of the convolution window. This tensor can be further viewed as a $(2K+2)$-dimensional tensor with size $m_1 \times j_1 \times \cdots \times m_K \times j_K \times H \times W$. In our experiments $H = W = 3$ remain unchanged, and we only compress along the remaining dimensions, i.e.,

$$\boldsymbol{\mathcal{C}} = [\![\boldsymbol{\mathcal{G}}^{(1)},\boldsymbol{\mathcal{G}}^{(2)},\ldots,\boldsymbol{\mathcal{G}}^{(2K)}]\!]_{TT}. \tag{17}$$

The shape of each factors $\boldsymbol{\mathcal{G}}^{(1)}, \boldsymbol{\mathcal{G}}^{(2)}, \ldots \boldsymbol{\mathcal{G}}^{(2K-1)}, \boldsymbol{\mathcal{G}}^{(2K)}$ are $m_1 \times R_1, R_1 \times j_1 \times R_2, \ldots, R_{2K-2} \times m_K \times R_{2K-1}, R_{2K-1} \times j_K \times H \times W$, respectively. The parameters in both fully connected layers and convolutional layers can be represented as

$$\Theta = \{\boldsymbol{\mathcal{G}}^{(1)}, \Lambda^{(1)}, \boldsymbol{\mathcal{G}}^{(2)}, \Lambda^{(2)}, \ldots\}, \tag{18}$$

where hyper-parameters $\Lambda^{(k)} = \text{diag}(\lambda_1^{(k)}, \ldots \lambda_{R_k}^{(k)})$ are used to control the rank R_k.

Here a Gaussian prior is placed over each tensor factor and a Gamma prior is placed over $\Lambda^{(k)}$,

$$\begin{aligned} P(\boldsymbol{\mathcal{G}}^{(k)}|\Lambda^{(k-1)},\Lambda^{(k)}) &= \prod_{i,j}\mathcal{N}\big(\boldsymbol{\mathcal{G}}^{(k)}(i,:,j)\big|0,(c_k\lambda_i^{(k-1)}\lambda_j^{(k)})^{-1}\big), \\ P(\Lambda^{(k)}) &= \prod_{r=1}^{R_k}\text{Gamma}(\lambda_r^{(k)}|\alpha,\beta), \\ P(\Theta) &= \prod_{k=1}^{d} P(\boldsymbol{\mathcal{G}}^{(k)}|\Lambda^{(k-1)},\Lambda^{(k)})\prod_{k=1}^{d-1}P(\Lambda^{(k)}). \end{aligned} \tag{19}$$

where α and β are constants. Once the estimated parameter $\lambda_r^{(k)}$ is larger than a threshold, we delete one horizontal slice of

$\mathcal{G}^{(k)}$ and one frontal slice of $\mathcal{G}^{(k+1)}$. The distribution of $\Lambda^{(k)}$ is the same as Equation (30). The prior of unknown parameters $\Theta = \{\mathcal{G}^{(k)}, \Lambda^{(k)}\}$ is $P(\Theta) = \prod_{k=1}^{d} P(U^{(k)}|\Lambda^{(k)})P(\Lambda^{(k)})$. The idea of the tensor-train low-rank prior is structurally similar to the idea of the low-rank CP prior in Equation (12). However, instead of shrinking columns of every factor matrices, each element $\lambda_j^{(k)}$ controls the prior variance of two factor tensor slices, each of which can shrink to 0. The negative log prior is

$$\begin{aligned} -\log P(\Theta) =& \frac{1}{2}\sum_{k=1}^{d}\sum_{i=1}^{I_k}\left\langle \mathcal{G}^{(k)}(:,i,:) * \mathcal{G}^{(k)}(:,i,:), (\Lambda^{(k-1)} \otimes \Lambda^{(k)})^{-1}\right\rangle \\ &-\frac{1}{2}\sum_{k=1}^{d}\left(I_k R_k \sum_{r=1}^{R_{k-1}} \lambda_r^{(k-1)} + I_k R_{k-1}\sum_{r=1}^{R_k}\lambda_r^{(k)}\right) \\ &-\sum_{k=1}^{d-1}\sum_{r=1}^{R_k}\left((a-1)\log\lambda_r^{(k)} - b\lambda_r^{(k)}\right). \end{aligned} \tag{20}$$

Here $*$ denotes the element-wise product of two tensors or matrices.

For all hyperparameters $\lambda_r^{(k)}$, we sample $\hat{\lambda}_r^{(k)} = \log\lambda_r^{(k)}$ and use the log Gamma distribution as a prior. The potential function can be computed as

$$U(\Theta) = -\log P(\Theta|\mathcal{D}) = \sum_{n=1}^{N} \text{loss}(\mathbf{y}_n, g(\mathbf{x}_n, \Theta)) - \log P(\Theta), \tag{21}$$

where loss$(\cdot)$ is the negative log likelihood and $g(\cdot)$ denotes the neural network. The loss function can be the cross entropy loss for classification problems and the mean square error loss for regression problems. After getting the potential function, we can apply the SGHMC or SVGD framework to draw samples for the parameters Θ.

In this framework, the tensor ranks can be adjusted automatically to reduce the neural network model size in training. We propose to set a threshold and reduce the rank when

$$\hat{\lambda}_r^{(k)} \geq \log(\frac{1}{2}S_k R_{k-1} + \frac{1}{2}S_{k+1}R_{k+1} + \alpha) + \log\beta - \epsilon, \tag{22}$$

where $S_k = M_k J_k$ for the fully connected layers and $S_{2k-1} = M_k, S_{2k} = J_k$ for the convolutional layers. We select this threshold for λ by setting $\mathcal{G}^k(:,i,:) = 0$ and maximizing the negative log prior from (20) with respect to Λ. Therefore, the threshold is chosen as an MAP point of the marginal log-prior conditioned on the value of the tensor factors.

3.3. More General Models

Our Bayesian tensor learning framework can also be applied to other low-rank tensorized neural network formats such as the CP and Tucker formats, other tensorized neural network tasks such as regression instead of classification, and to other tensor completion and factorization approaches using the tensor-train or Tucker formats. For instance, we only need to change the likelihood function to a Gaussian distribution when solving a neural network regression task. We summarize these results in **Table 1**. In this subsection we provide the likelihoods for more general tensor models and a Bayesian rank-shrinkage prior for the low-rank Tucker format.

TABLE 1 | Equations to calculate the potential energy $U(\Theta)$ for different tensor learning tasks and with different low-rank tensor formats.

Learning tasks	Likelihood	CP	Tucker	Tensor-Train
Tensor completion	Gaussian	(12)+(14) +(10)	(31)+(14)+(10)	(20)+(14)+(10)
Neural network classification	Multinomial	(12)+(24)	(31)+(24)	(20)+(24)
Neural network regression	Gaussian	(12)+(26)	(31)+(26)	(20)+(26)

Classification Problems. In most classification problems, the neural network can give a likelihood $\hat{\mathbf{y}}_i = f(\mathbf{x}_i; \Theta)$ directly, where $f(\mathbf{x}_i; \Theta)$ is the propagation function of the network, $\hat{\mathbf{y}}_i$ is a vector and each element denotes the probability that x_i belongs to one class. It is usually the softmax of output of the last linear layer. Suppose $\mathbf{y}_i$ is a vector with size C, C is the total number of classes,

$$\mathbf{y}_{ic} = \begin{cases} 1, & \text{if } \mathbf{x}_i \text{ in class } c \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

The negative log likelihood is

$$-\log P(\mathbf{y}_i|\Theta) = \langle \mathbf{y}_i, -\log f(x_i; \Theta)\rangle. \tag{24}$$

Regression Problems. In a regression problem, it is usually assumed to have a Gaussian likelihood function

$$P(\mathbf{y}_i|\Theta) = \mathcal{N}(\mathbf{y}_i|f(\mathbf{x}_i; \Theta), \sigma^2). \tag{25}$$

This leads to the following negative log-likelihood:

$$-\log P(\mathbf{y}_i|\Theta) = \frac{1}{2}(\mathbf{y}_i - f(\mathbf{x}_i; \Theta))^2/\sigma^2, \tag{26}$$

where σ is a hyperparameter denoting the variance.

Tucker Tensor Prior. A popular alternative to the CP and tensor-train tensor formats is the low-rank Tucker format (Tucker, 1966). The Tucker decomposition projects the original tensor $\boldsymbol{\mathcal{X}}$ into a smaller kernel tensor $\boldsymbol{\mathcal{G}}$,

$$\begin{aligned} \boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{G}} \bigotimes_{k=1}^{d} \mathbf{U}^{(k)} &\Longleftrightarrow \boldsymbol{\mathcal{X}}(i_1,\cdots,i_d) \\ &= \sum_{r_1=1}^{R_1}\cdots\sum_{r_d=1}^{R_d} \boldsymbol{\mathcal{G}}(i_1,\cdots,i_d)\mathbf{U}^{(1)}(i_1,r_1)\cdots\mathbf{U}^{(d)}(i_d,r_d). \end{aligned} \tag{27}$$

Similar to Zhao et al. (2015b), the priors of $\mathbf{U}^{(k)}$ and $\boldsymbol{\mathcal{G}}$ are set as

$$P(\mathbf{U}^{(k)}|\Lambda^{(k)}) = \prod_{i_k=1}^{I_k} \mathcal{N}(\mathbf{U}^{(k)}(i_k,:)|0, (\Lambda^{(k)})^{-1}) \tag{28}$$

and

$$P(\mathcal{G}|\Lambda^{(1)},\ldots,\Lambda^{(d)}) = \prod_{r_1,\ldots,r_d} \mathcal{N}\left(\mathcal{G}(r_1,\ldots,r_d)\,\middle|\,0,\beta\prod_{k=1}^{d}\left(\lambda_{r_k}^{(k)}\right)^{-1}\right) \tag{29}$$

respectively. Here β is a constant scaling factor. For simplicity, we assume β is a constant instead of a random variable, which differs from Zhao et al. (2015b). The hyperparameter $\Lambda^{(k)}$ follows from the Gamma distribution

$$P(\Lambda^{(k)}) = \prod_{r=1}^{R_k} \mathrm{Gamma}(\lambda_r^{(k)}|a,b). \tag{30}$$

Here, $\Lambda^{(k)}$ is shared between $\mathbf{U}^{(k)}$ and $\mathcal{G}$. We observe that similar to other low-rank tensor formats this prior enforces structural rank-shrinkage. This approach differs from the CP format in that each low-rank hyperparamter $\lambda_{r_k}^{(k)}$ controls entries in both a factor matrix and the Tucker tensor core $\mathcal{G}$.

In summary, the prior of the unknown parameters $\Theta = \{\mathcal{G}, \mathbf{U}^{(k)}, \Lambda^{(k)}\}$ is

$$P(\Theta) = P(\mathcal{G}|\Lambda^{(1)}\ldots\Lambda^{(1)})\prod_{k=1}^{d} P(\mathbf{U}^{(k)}|\Lambda^{(k)})P(\Lambda^{(k)})$$

The negative log prior is

$$\begin{aligned}-\log P(\Theta) = {} & \frac{1}{2}\sum_{k=1}^{d}\mathbf{tr}(\mathbf{U}^{(k)}\Lambda^{(k)}(\mathbf{U}^{(k)})^T) + \frac{1}{2}\left\langle \beta\bigotimes_{k=1}^{d}\Lambda^{(k)}, \mathcal{G}*\mathcal{G}\right\rangle \\ & - \frac{1}{2}\sum_{k=1}^{d}\left((I_k + \prod_{t\neq k} R_t)\sum_{r=1}^{R_k}\log\lambda_r^{(k)}\right) \\ & - \sum_{k=1}^{d}\sum_{r=1}^{R_k}\left((a-1)\log\lambda_r^{(k)} - b\lambda_r^{(k)}\right)\end{aligned} \tag{31}$$

where $\mathcal{G}*\mathcal{G}$ is the element-wise product of two tensors.

4. NUMERICAL EXPERIMENTS

4.1. Tensor Completion

We first consider a synthetic example and an MRI data experiment to show the efficiency of our proposed methods in tensor data recovery, uncertainty estimation, and automatic rank determination in CP tensor format. In both our SGHMC and SVGD implementations, we generate the initial guess by two steps: we first use the ADAM method (Kingma and Ba, 2014) to reach the neighborhood of a local optimal, and then reduce its rank by truncating all $\hat{\lambda}_r$ below the threshold given in Equation (22). After that, with SGHMC algorithm, we discard the first 50 samples and use the next 450 samples for evaluation. With SVGD algorithm, we use 10 samples for evaluation.

4.1.1. Synthetic Dataset

We first randomly generated a $20\times20\times20$ tensor with the ground truth rank of 5. We consider two sets of experiments. **Case 1: tensor factors with uniform distributions.** Assume the factors follow an independent uniform distribution between 0 and 1. We randomly select 10% of the tensor data to be observed. **Case 2: tensor factors with Gaussian distribution**. Assume the factors follow a Gaussian distribution with zero mean and variance one. We randomly sample 20% entries of the whole tensor. We select a higher observation ratio in this task because it is more difficult than task with uniformly distributed tensor factors. This is because the true data and the noise have similar distributions. In both cases, we perturb all elements with identically independent distributed Gaussian noise $\mathcal{N}(0,\sigma^2)$. In Equation (11), we set $R = 15, a = c = 1, b = 1$ for the uniform factors and $b = 4$ for Gaussian factors, $d = 10^6$ when $\sigma = 0.001$ and $d = 10^4$ otherwise. For the HMC approach we divided the parameters in two groups, and set the mass of factors matrices as 1 and the mass of all other parameters to 100, as the values of the factor matrices vary more during sampling. We report the root mean square error

$$\mathrm{RMSE}: \|\overline{\mathcal{X}} - \mathcal{Y}\|_F/\sqrt{\prod_k I_k}$$

where $\mathcal{Y}$ is the ground truth and $\|\cdot\|_F$ is the Frobenius norm, and SD is the predicted noise standard deviation. The results are shown in **Table 2**.

For the tensor with Gaussian factors, our proposed methods perform almost as well as the mean field approximation (BFCP) (Zhao et al., 2015a) in terms of RMSE and SD. For the tensor with uniform-distributed factors, the BFCP method always underestimates the rank and results in high recovery error and SD. This is because the mean-field assumption on the posterior in the BFCP method places a strong Gaussian assumption on the approximating distribution. We also observe in **Table 2** that the SVGD approach can produce lower RMSE estimates than the proposed HMC approach, but may underestimate the uncertainty and predict an SD that is too low.

4.1.2. MRI Dataset

We continue to consider the PINCAT MRI dataset (Candes et al., 2013). This is a $128 \times 128 \times 50$ complex-value tensor and we only consider its amplitude. We re-scale the tensor such that the average amplitude $\|\mathcal{A}\|_F/\sqrt{I_1I_2I_3} = 1$. We randomly sample 80000 ($\approx$ 10%) elements. The parameters in (11) are set to be $a = 1, b = 0.2, c = 1, d = 10^4$, and $R = 80$. We compare our results with BFCP report them in **Table 2**. The results in **Table 2** demonstrate that our methods obtain a much lower RMSE and SD than BFCP, and that BFCP underestimates the rank.

4.2. Tensorized Neural Networks

In this section, we present numerical experiments of our Bayesian tensor learning framework for tensorized neural network tasks. We evaluate both the compression capabilities and

TABLE 2 | Numerical results of tensor completion for the synthetic experiment and MRI dataset.

		Proposed-HMC			Proposed-SVGD			BFCP (Zhao et al., 2015a)		
True factors	**Noise**	**Est. Rank**	**RMSE**	**SD**	**Est. Rank**	**RMSE**	**SD**	**Est. Rank**	**RMSE**	**SD**
	0.001	5	0.0013	0.0047	5	0.0019	0.0011	1	0.1476	0.1517
Uniform	0.003	5	0.0038	0.0040	5	0.0031	0.0016	1	0.1499	0.1607
Rank-5	0.01	5	0.0128	0.0118	5	0.0114	0.0098	1	0.1386	0.1365
	0.03	5	0.0403	0.0318	5	0.0512	0.0071	1	0.1468	0.1523
	0.001	5	0.0013	0.0025	5	0.0019	0.0007	6	0.0005	0.0011
Gaussian	0.003	5	0.0038	0.0031	5	0.0027	0.0019	6	0.0033	0.0033
Rank-5	0.01	5	0.0130	0.0102	5	0.0119	0.0069	5	0.0106	0.0110
	0.03	5	0.0418	0.0236	5	0.0336	0.0193	7	0.0338	0.0354
MRI dataset		65	0.0856	0.0670	65	0.0727	0.0319	17	0.1495	0.1456

TABLE 3 | Results of different networks on two datasets.

		#Parameters	MAP		Proposed	
Dataset	**Network**	**(compression ratio)**	**LL**	**Accuracy**	**LL**	**Accuracy**
Fashion-MNIST	NN	$3.97 \times 10^5(1\times)$	–0.7118	88.91%	–0.6730	89.41%
	TT-NN	$2.63 \times 10^4(15.1\times)$	–0.6687	87.07%	–0.6337	87.78%
	HMC-BF-TT-NN	$4.02 \times 10^3(98.8\times)$	–0.3317	88.24%	–0.3254	88.64%
	SVGD-BF-TT-NN	$2.8 \times 10^4(14.1\times)$	–0.3317	88.24%	–0.3261	88.57%
	Tucker-NN	$2.57 \times 10^5(1.54\times)$	–1.1673	87.20%	–1.0984	87.53%
	HMC-BF-Tucker-NN	$3.10 \times 10^4(12.8\times)$	–1.2948	87.18%	–0.4405	88.18%
	SVGD-BF-Tucker-NN	$3.10 \times 10^4(12.8\times)$	–1.2948	87.18%	–0.4705	87.86%
CIFAR-10	CNN	$9.91 \times 10^6(1\times)$	-0.5337	91.54%	–0.5370	91.53%
	TT-CNN	$6.93 \times 10^5(14.3\times)$	–0.6077	89.00%	–0.5329	90.13%
	HMC-BF-TT-CNN	$7.83 \times 10^4(127\times)$	–0.3936	86.68%	–0.3623	88.01%
	SVGD-BF-TT-NN	$7.83 \times 10^4(127\times)$	–0.3936	86.68%	–0.3419	88.41%

LL, predictive log likelihood (the larger the better); TT, tensor train decomposition; Tucker, Tucker decomposition; BF, Bayesian low rank prior.

accuracy of our proposed method on two common computer vision datasets.

4.2.1. Datasets

We first consider the Fashion-MNIST dataset (Xiao et al., 2017) by a two layer neural network. The first layer (FC1) is a 784×500 fully connected layer with a ReLU activation and the second layer (FC2) is a 500×10 fully connected layer with the softmax activation. We convert FC1 as a 8-th order tensor and FC2 as a 4-th order tensor for the tensor-train decomposition. For the Tucker decomposition, we convert FC1 as a 4-th order tensor and FC2 into a 3-th order tensor.

Next we consider the CIFAR-10 dataset. We build a 6-layer convolutional neural network (CNN) containing 4 convolution layers and 2 fully connected layers. Each convolution layer has a kernel size of 3×3 and padding of 1. The number of channels in each convolution layer is 128, 256, 256, 256, respectively. The size of the first fully connected layer (FC1) is 512. A batch normalization layer and a ReLU activation layer is placed after each convolution and fully-connected layer. A maxpooling layer with kernel size of 2×2 is placed after the second and the fourth convolution layer.

4.2.2. Numerical Results

We use the ADAM method to minimize the negative posterior to get an initial point, then shrink the rank according to equation (22). In Fashion-MNIST dataset, the initialization takes 50 epochs, and in Cifar-10 dataset, it takes 150 epochs. We searched the hyperparameters in the grid defined by β among $[0.1, 0.2, 0.5, 1, 2, 5, 10]$ and α among $[1, 2, 5]$ and picked the one with the best trade-off between accuracy and compression ratio. Afterwards, we generate $T = 450$ SGHMC samples aftering discarding the first 50 samples, or $n = 10$ SVGD samples.We evaluate the accuracy of this model using two criterion: the predictive log likelihood (LL) and the prediction accuracy. The results for different benchmarks using different tensor formats are shown in **Table 3**. We compare the proposed Bayesian learning with the optimization method that maximize a posterior (MAP) directly. It is shown that our tensor learning framework outperforms MAP in almost every case in terms of both the accuracy and the log likelihood (LL). The improvement in log likelihood indicates that our model can predict the uncertainty better than the MAP method. Besides, our method achieves a compression ratio of up to 98.8× in Fashion-MNIST and 127× in CIFAR-10 in terms of the number of model parameters

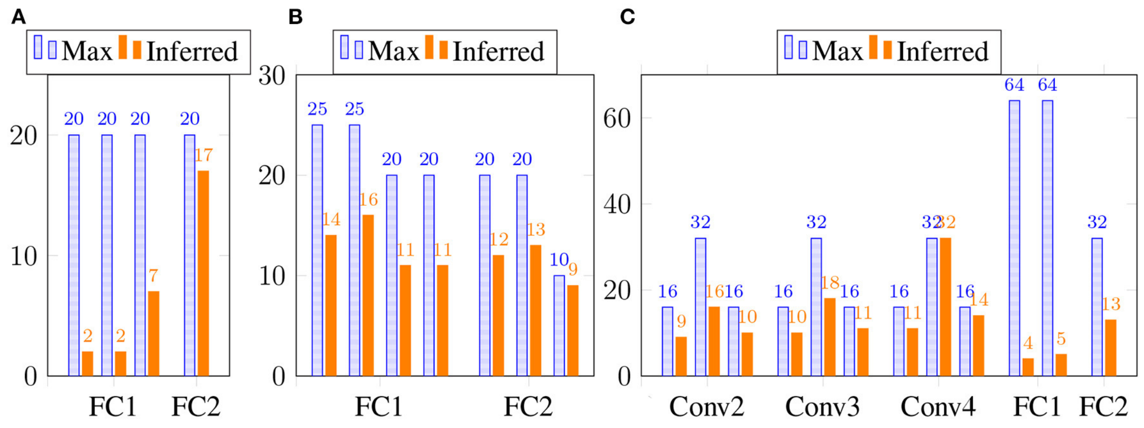

FIGURE 1 | The inferred TT rank at different layers. **(A)** 2 TT-FC layers for Fashion-MNIST. **(B)** 2 Tucker-FC layers for Fashion-MNIST. **(C)** 4 TT-Conv and 2 TT-FC layers for CIFAR-10.

compared with the baseline network. One SGD initialization of our method on the CIFAR-10 problem takes approximately 3 h to run on an NVIDIA Titan V GPU with 12GB of memory. Either sampling process (SVGD or HMC) takes less than 10 min.

We also show the estimated tensor-train ranks of the estimated weight matrices and convolution filters in **Figure 1**. Clearly, our Bayesian tensor learning framework can perform model compression in the training process with automatic rank determination.

5. RELATED WORK

There are many recent works on training a tensor learning model. There are a series of works targeting on tensor completion/factorization problems. Chu and Ghahramani (2009) proposed tensor completion algorithm using a maximum a posteriori (MAP) MAP estimation. Various attempts to evalute the uncertainly of completion using Bayesian method has been made in Xiong et al. (2010), Rai et al. (2014), Zhao et al. (2015a,c). On the other hand, some other works targets only on training a tensorized neural networks Jaderberg et al. (2014) demonstrated the first attempt to compress a CNN using tensor decomposition method. Tai et al. (2015) further improved that by training such a neural network from scratch and evaluate it on a large network. Kossaifi et al. (2019) proposed to compress a CNN by parametrizing the entire structure with a single, high-order tensor. In Kolbeinsson et al. (2021), tensor dropout technique was proposed. This technique can be applied to tensor factorizations and improve the robustness and generalization abilities while provide more computationally and memory efficient models. Bulat et al. (2021) further empirically demonstrated that tensor dropout method can improve the robustness to adversarial attacks. Hayashi et al. (2019) proposed a graphical notation to represent all kinds of decomposition used in tensorized CNN and experimentally compare the tradeoff between accuracy and efficiency. Our work proposes a generic, scalable and uncertainty-aware algorithm to solve a broad class of tensor learning problems, including but not limited to tensor factorization/completion, regression, and tensorized neural networks. Furthermore, these works either focused on post-training compression or fixed-rank training, while our work is the first to our knowledge that perform on-shot rank-adaptive training.

6. CONCLUSION

We have presented a generic Bayesian framework that is applicable to various tensor learning task described with different low-rank tensor representations. This framework is implemented with Hamiltonian Monte Carlo and Stein variational gradient descent. Among the wide range of applications in tensor learning tasks, we have specifically tested our methods by tensor completion with CP format and tensorized Bayesian neural networks with both tensor train and Tucker formats. In tensor completion, our method has shown better accuracy and capability of rank determination than the state-of-the-art mean-field approximation. In the Bayesian neural network, our method has demonstrated a significant compression ratio in the end-to-end training of tensorized neural networks, as well as better accuracy than the maximum-a-posterior training.

DATA AVAILABILITY STATEMENT

The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

AUTHOR CONTRIBUTIONS

KZ and CH developed the low-rank Bayesian model under the guidance of ZZ. KZ coded the proposed HMC approach and CH coded the proposed SVGD approach. All authors contributed to the article and approved the submitted version.

FUNDING

This work was supported by NSF CCF-1817037 and DOE ASCR grant no. DE-SC0021323.

REFERENCES

Acar, E., Dunlavy, D. M., Kolda, T. G., and Mørup, M. (2011). Scalable tensor factorizations for incomplete data. *Chemometrics Intell. Lab. Syst.* 106, 41–56. doi: 10.1016/j.chemolab.2010.08.004

Andrieu, C., De Freitas, N., Doucet, A., and Jordan, M. I. (2003). An introduction to mcmc for machine learning. *Mach. Learn.* 50, 5–43. doi: 10.1023/A:1020281327116

Austin, W., Ballard, G., and Kolda, T. G. (2016). "Parallel tensor compression for large-scale scientific data," in *Intl. Parallel and Distributed Processing Symp.*, (Chicago, IL: ACM), 912–922.

Bro, R. (1997). Parafac. tutorial and applications. *Chemometrics Intell. Lab. Syst.* 38, 149–171. doi: 10.1016/S0169-7439(97)00032-4

Bulat, A., Kossaifi, J., Bhattacharya, S., Panagakis, Y., Hospedales, T., Tzimiropoulos, G., et al. (2021). Defensive tensorization. *arXiv preprint* arXiv:2110.13859.

Candes, E. J., Sing-Long, C. A., and Trzasko, J. D. (2013). Unbiased risk estimates for singular value thresholding and spectral estimators. *IEEE Trans. Signal Process.* 61, 4643–4657. doi: 10.1109/TSP.2013.2270464

Chen, T., Fox, E., and Guestrin, C. (2014). "Stochastic gradient hamiltonian monte carlo," in *International Conference on Machine Learning*, (Beijing: ICML), 1683–1691.

Chu, W. and Ghahramani, Z. (2009). "Probabilistic models for incomplete multi-dimensional arrays," in *Artificial Intelligence and Statistics*, (Florida, FL: Clearwater Beach), 89–96.

Cichocki, A. (2014). Era of big data processing: a new approach via tensor networks and tensor decompositions. *arXiv preprint* arXiv:1403.2048.

Ding, N., Fang, Y., Babbush, R., Chen, C., Skeel, R. D., and Neven, H. (2014). "Bayesian sampling using stochastic gradient thermostats," in *Advances in Neural Information Processing Systems*, (Vancouver, VN: MIT Press), 3203–3211.

Duane, S., Kennedy, A. D., Pendleton, B. J., and Roweth, D. (1987). Hybrid monte carlo. *Phys. Lett. B* 195, 216–222. doi: 10.1016/0370-2693(87)91197-X

Gilbert, D. E., and Wells, M. T. (2019). Tuning free rank-sparse bayesian matrix and tensor completion with global-local priors. *arXiv preprint* arXiv:1905.11496.

Harshman, R. A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis. *UCLA Working Papers in Phonetics* 16, 1–84. Available Online at: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.144.5652&rep=rep1&type=pdf.

Hawkins, C., and Zhang, Z. (2018). Robust factorization and completion of streaming tensor data via variational bayesian inference. *arXiv preprint* arXiv:1809.01265.

Hawkins, C., and Zhang, Z. (2021). Bayesian tensorized neural networks with automatic rank selection. *Neurocomputing* 453, 172–180. doi: 10.1016/j.neucom.2021.04.117

Hayashi, K., Yamaguchi, T., Sugawara, Y., and Maeda, S.-I. (2019). "Exploring unexplored tensor network decompositions for convolutional neural networks," in *Advances in Neural Information Processing Systems*, Vol. 32 (Cambridge, MA:), 5552–5562.

Hillar, C. J., and Lim, L.-H. (2013). Most tensor problems are np-hard. *J. ACM* 60, 45. doi: 10.1145/2512329

Iserles, A. (1986). Generalized leapfrog methods. *IMA J. Numer. Anal.* 6, 381–392. doi: 10.1093/imanum/6.4.381

Jaderberg, M., Vedaldi, A., and Zisserman, A. (2014). Speeding up convolutional neural networks with low rank expansions. *arXiv preprint* arXiv:1405.3866.

Jain, P., and Oh, S. (2014). "Provable tensor factorization with missing data," in *Advances in Neural Information Processing Systems*, (Montreal, MO: MIT Press), 1431–1439.

Kingma, D. P., and Ba, J. (2014). Adam: a method for stochastic optimization. *arXiv preprint* arXiv:1412.6980.

Kolbeinsson, A., Kossaifi, J., Panagakis, Y., Bulat, A., Anandkumar, A., Tzoulaki, I., et al. (2021). Tensor dropout for robust learning. *IEEE J. Sel. Topics Signal Process.* 15, 630–640. doi: 10.1109/JSTSP.2021.3064182

Kolda, T. G., and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Rev.* 51, 455–500. doi: 10.1137/07070111X

Kossaifi, J., Bulat, A., Tzimiropoulos, G., and Pantic, M. (2019). T-net: Parametrizing fully convolutional nets with a single high-order tensor. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* Long Beach, CA, USA.

Liu, Q., and Wang, D. (2016). "Stein variational gradient descent: a general purpose bayesian inference algorithm," in *Advances in Neural Information Processing Systems*, Vol. 29 (Cambridge, MA: MIT Press), 2378–2386.

Neal, R. M. (1992). *Bayesian Training of Backpropagation Networks by the Hybrid Monte Carlo Method*. Technical Report, Department of Computer Science, University of Toronto

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. (2015). "Tensorizing neural networks," in *Advances in Neural Information Processing Systems*, (Cambridge, MA: MIT Press), 442–450.

Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM J. Sci. Comput.* 33, 2295–2317. doi: 10.1137/090752286

Rai, P., Wang, Y., Guo, S., Chen, G., Dunson, D., and Carin, L. (2014). "Scalable bayesian low-rank decomposition of incomplete multiway tensors," in *International Conference on Machine Learning*, (Glasgow: ICML), 1800–1808.

Schmidt, M. N. and Mohamed, S. (2009). "Probabilistic non-negative tensor factorization using markov chain monte carlo," in *European Signal Processing Conf.*, (Glasgow: ICML), 1918–1922.

Sidiropoulos, N. D., De Lathauwer, L., Fu, X., Huang, K., Papalexakis, E. E., and Faloutsos, C. (2017). Tensor decomposition for signal processing and machine learning. *IEEE Trans. Signal Process.* 65, 3551–3582. doi: 10.1109/TSP.2017.2690524

Şimşekli, U., Koptagel, H., Güldaş, H., Cemgil, A. T., Öztoprak, F., and Birbil, Ş. İ. (2015). Parallel stochastic gradient markov chain monte carlo for matrix factorisation models. *arXiv preprint* arXiv:1506.01418.

Tai, C., Xiao, T., Zhang, Y., Wang, X., and Weinan E. (2015). Convolutional neural networks with low-rank regularization. *arXiv preprint* arXiv:1511.06067.

Tucker, L. R. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 279–311. doi: 10.1007/BF02289464

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint* arXiv:1708.07747.

Xiong, L., Chen, X., Huang, T.-K., Schneider, J., and Carbonell, J. G. (2010). "Temporal collaborative filtering with bayesian probabilistic tensor factorization," in *Proceedings of the 2010 SIAM International Conference on Data Mining*, (Beijing: SIAM), 211–222.

Zhang, Z., Weng, T.-W., and Daniel, L. (2016). Big-data tensor recovery for high-dimensional uncertainty quantification of process variations. *IEEE Trans. Compon. Packag. Manufact. Technol.* 7, 687–697. doi: 10.1109/TCPMT.2016.2628703

Zhang, Z., Yang, X., Oseledets, I. V., Karniadakis, G. E., and Daniel, L. (2014). Enabling high-dimensional hierarchical uncertainty quantification by anova and tensor-train decomposition. *IEEE Trans. Comput.-Aied Design Integr. Circuits Syst.* 34, 63–76. doi: 10.1109/TCAD.2014.2369505

Zhao, Q., Zhang, L., and Cichocki, A. (2015a). Bayesian cp factorization of incomplete tensors with automatic rank determination. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 1751–1763. doi: 10.1109/TPAMI.2015.2392756

Zhao, Q., Zhang, L., and Cichocki, A. (2015b). Bayesian sparse tucker models for dimension reduction and tensor completion. *arXiv preprint* arXiv:1505.02343.

Zhao, Q., Zhou, G., Zhang, L., Cichocki, A., and Amari, S.-I. (2015c). Bayesian robust tensor factorization for incomplete multiway data. *IEEE Trans. Neural Netw. Learn. Syst.* 27, 736–748. doi: 10.1109/TNNLS.2015.2423694

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.