Noise adaptive tensor train decomposition for low-rank embedding of noisy data

Noise Noise Adaptive Tensor Train Decomposition for Low-Rank Embedding of Noisy Data

Xinsheng Li[1], K. Selçuk Candan[1], and Maria Luisa Sapino[2]

[1] Arizona State University, Tempa AZ 85281, USA
{lxinshen,candan}@asu.edu
[2] University of Torino, I-10149 Torino, Italy
marialuisa.sapino@unito.it

Abstract. Tensor train decomposition, one of the widely used tensor decomposition techniques, is designed to avoid the curse of dimensionality, in the form of the exposition of intermediary results, which plagues other tensor decomposition techniques. However, many tensor decomposition schemes, including tensor train decomposition is sensitive to noise in the input data streams: Recent research has shown that it is possible to improve the resilience of the tensor decomposition process to noise and other forms of imperfections in the data by relying on probabilistic techniques. However, these techniques have a major deficiency: they treat the entire tensor uniformly, ignoring potential non-uniformities in the noise distribution. In this paper, we note that noise is rarely uniformly distributed in the data and propose a *Noise-Profile Adaptive Tensor Train Decomposition (*`NTTD`*)* method, which aims to tackle this challenge. In particular, `NTTD` leverages a model-based noise adaptive tensor train decomposition strategy: any *rough* priori knowledge about the noise profiles of the tensor enable us to develop a sample assignment strategy that best suits the noise distribution of the given tensor.

1 Introduction

Tensors and tensor decomposition (such as CP [14] and Tucker [32]) are increasingly being used for AI and machine learning tasks, from anomaly detection, correlation analysis [30], to pattern discovery [19, 20].

Tensor Train Decomposition A common problem faced by tensor decomposition techniques, such as Tucker, which generates dense core tensors, is that, even when the input is sparse, the intermediary and final steps in the decomposition may lead to very large datasets. Recent research has shown that several generalizations of higher order tensors' low-rank decompositions, such as hierarchical Tucker (HT) [18] and the Tensor Train (TT) [28] format, are effective solutions to this problem. Both HT and TT are designed to avoid the curse of dimensionality. Intuitively, the TT decomposition (which can be interpreted as a special case of HT, without a recursive formulation) avoids the creation of a high-modal dense core, by splitting the core into a sequence of low (3) modal

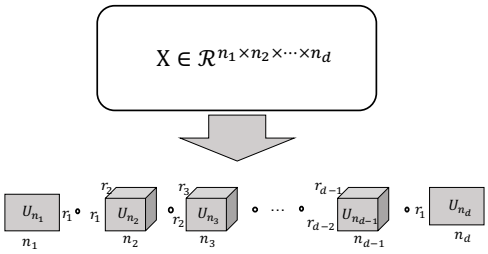


Fig. 1: Illustration of Tensor Train (TT) decomposition

cores (Figure 1). Since, computation and storage are exponential in the number of modes, TT is widely used for decomposition in various applications [31, 34].

Challenge: Noisy Data The problem the tensor train decomposition faces is that the overall decomposition process can be negatively affected by the noise and low quality in the data, which is especially a concern for web and web-based user data [6, 35]. Especially for sparse data, avoiding over-fitting to noisy data can be a significant challenge. Recent research has shown that it may be possible to avoid such over-fitting by relying on probabilistic techniques [33]. Unfortunately, existing probabilistic approaches have one major deficiency: they treat the entire tensor uniformly, ignoring possible non-uniformities in the distribution of noise in the given tensor. [27] has shown that if available, even *rough* a priori knowledge about the noise profiles of the tensor may enable CP-based decomposition strategies that are robust against noise, but these *uni-core* techniques are not applicable to the *multi-core* tensor train decomposition process, which results in a sequence of low-modal cores.

Contribution: Noise-Profile Adaptive Tensor Train Decomposition (`NTTD`) In this paper, we propose a *Noise-Profile Adaptive Tensor Train Decomposition (`NTTD`)* method, which leverages *rough* a prior information about noise in the data (which may be user provided or obtained through automated techniques) to improve decomposition accuracy. `NTTD` decomposes each mode matricization probabilistically through Bayesian factorization – the resulting factor matrix are then reconstructed to obtain the tensor approximations. Most importantly,

> `NTTD` *provides a resource allocation strategy, which accounts for the impacts of (a) the noise density of each mode and (b) inherent approximation error of the Tensor Train decomposition process, on the overall decomposition accuracy of the input tensor.*

In other words, *a priori* knowledge about noise distribution on the tensor and the inherently approximate nature of the tensor train decomposition process are both considered to obtain a decomposition strategy, which involves (a) the order of the modes and (b) the number of Gibbs samples allocated to each step of the decomposition process, that best suits the noise distribution of the given tensor.

Algorithm 1 PTTD

Input: d dimensional tensor $\boldsymbol{\mathcal{X}}$, Rank $R = \{r_1, ..., r_{d-1}\}$
Output: Decomposed factors $\boldsymbol{U}^{(1)}$, ..., $\boldsymbol{U}^{(d)}$ of
TT-approximation $\tilde{\boldsymbol{\mathcal{X}}}$

1. generate the appropriate sampling number for each mode ,$S = \{s_1, ..., s_{d_1}\}$ with **intelligent sampling assignment strategy**
2. Temporary Tensor: $M = \boldsymbol{\mathcal{X}}$
3. for k = 1 to d-1 do
   (a) $M := reshape(M, [r_{k-1}n_k, \frac{numel(M)}{r_{k-1}n_k}])$
   (b) apply the Probabilistic Matrix Factorization (PMF) on the matrix M with pre-given rank r_k and sampling number s_k to get the $\boldsymbol{U}^{(k)} and \boldsymbol{V}^{(k)}$
   (c) New core: $\boldsymbol{U}^{(k)} = reshape(U, [r_{k-1}, n_k, r_k])$
   (d) $M := SV^{(k)T}$
4. $U_d = M$
5. Return tensor $\tilde{\boldsymbol{\mathcal{X}}}$ in TT-format with scores $\boldsymbol{U}^{(1)}$, ..., $\boldsymbol{U}^{(d)}$

2 Background and Notations

The tensor model maps a schema with N attributes to an N-modal array and the decomposition process generalizes the matrix decomposition process to tensors. The two most popular tensor decomposition algorithms are the Tucker [32] and the CANDECOMP/PARAFAC (CP) [14] decompositions.

2.1 Tensor Train Decomposition

A major difficulty with the Tucker decomposition is that the dense core can be prohibitively large and expensive for high-modal tensors. While several tensor network approaches [28, 7, 8, 17] (where network have been proposed to avoid large, dense core tensors, the tensor train (TT) format [28, 31], which creates a linear tensor network (or a matrix product state, MPS [26]) avoids the deficiencies of many other complex decomposition structures:

Definition 1 (Tensor Train (TT) Format). *Let* $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times \cdots \times n_d}$ *be a tensor of order d. As we see in Figure 1 (in Introduction) shown, the* tensor train decomposition *decomposes* $\boldsymbol{\mathcal{X}}$ *into d matrices* $\mathbf{U}_{n_1}, \mathbf{U}_{n_2}, \ldots, \mathbf{U}_{n_d}$ *such that,*

$$\boldsymbol{\mathcal{X}} \approx \tilde{\boldsymbol{\mathcal{X}}} = \mathbf{U}_{n_1} \circ \mathbf{U}_{n_2} \circ \cdots \circ \mathbf{U}_{n_d}, \tag{1}$$

where $\mathbf{U}_{n_1} \in \mathbb{R}^{n_1 \times r_1}$, $\mathbf{U}_{n_i} \in \mathbb{R}^{r_{i-1} \times n_i \times r_i} (i = 2, \ldots, d-1)$, *and* $\mathbf{U}_{n_d} \in \mathbb{R}^{r_{d-1} \times n_d}$. $\Diamond$

Tensor train decomposition proceeds with matricizations along one mode at a time: at each step, a low-dimensional core corresponding to the current mode is obtained and the remainder of the data (which now has one less mode) is passed to the next step in the process [28].

3 Probabilistic TT Decomposition (PTTD)

Given the limitations of SVD on sparse and noisy tensors, the first step is to introduce a *probabilistic TT decomposition* scheme (`PTTD`), which extends TT tensor train decomposition framework with probabilistic matrix factorization [29].

`PTTD` replaces the SVD decomposition step in tensor train decomposition with probabilistic matrix factorization, in order to avoid over-fitting due to data sparsity and noise. More specifically,, the tensor $\boldsymbol{\mathcal{X}}$ is matricized as a matrix, M, and then we apply probabilistic matrix factorization on this matrix. Under a Bayesian formulation, the prior distributions over U and V are assumed to be Gaussian:

$$p(U|\mu_U, \Lambda_U) = \prod_{i=1}^{n_1} \mathcal{N}(U_i|\mu_U, \Lambda_U^{-1}) \tag{2}$$

A similar formulation holds for V. The resulting factor matrix, U, is assigned as the first TT factor matrix. The matrix V is reshaped into the matrix M_{next} to be factorized in the next step. This probabilistic factorization and reshape processes are repeated until the decomposition is completed. The pseudo code of the algorithm is presented in Algorithm 1.

Since exact evaluation is analytically intractable, we need to resort to approximate inference. While variational methods [16, 21] are possible, they can produce inaccurate results because they tend to involve overly simple approximations to the posterior. MCMC-based methods [36], however, where the factor matrix $\{U_i^{(k)}, V_j^{(k)}\}$ are sampled by running a Markov chain, have been shown to asymptotically approach the exact results.

Note that, in and of itself, `PTTD` does not leverage **a priori** knowledge about noise distribution and internal decomposition interaction, but it provides the framework in which noise-profile based adaptation can be implemented. More specifically, each row of factor matrices U and V follows a Gaussian distribution and this Gaussian is related to the uncertainty in the corresponding element and, thus, provides an opportunity to discover the distribution of data noise across the tensor, as we discuss in the next Section (Section 4).

4 Noise Adaptive Probabilistic Tensor Train Decomposition (`NTTD`)

One key advantage of the probabilistic decomposition framework presented above is that it can simultaneously uncover (Gaussian) noise while obtaining the decomposition[28]. Yet, it fails to account for the *potentially available (user-provided or automatically discovered) knowledge* about (a) the distribution of the *external* data noise across the tensor and (b) the noise generated *internally* due to the inherent imperfections in the decomposition process at the different steps of the tensor train network.

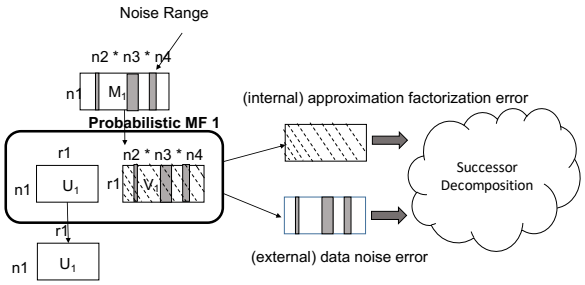


Fig. 2: Two types of errors propagate to downstream matricizations in tensor train decomposition: (*internal*) approximate factorization error and (*external*) data noise error

4.1 External and Internal Noise

External (Data) Noise. In this paper, we define, (external data) *noise density* as the ratio of the cells that are subject to noise. Without loss of generality, we assume noise exists only on cells that have values (i.e., the observed values can be faulty, but there are no spurious observations) and, thus, we formalize noise density as the ratio of the non-null cells that are subject to noise. Note that noise may impact the observed values in the tensor in different ways: in *value-independent noise* [27], the correct data may be overwritten by a completely random new value, whereas in *value-correlated noise*, existing values may be perturbed (often with a Gaussian noise, defined by a standard deviation, σ). We refer to the amount of perturbation as the *noise intensity*.

Internal (Decomposition) Noise. As we discussed in Section 2.1, in the tensor train format, the network structure acts as a "train" or "chain" of tensors: the core tensors only interact with their neighboring cores as illustrated in Figure 1. The corresponding tensor train decomposition relies on sequential projections (formulated as sequential matrix factorizations) and *the decomposition accuracy of the intermediate matrix, M_k, depends on the accuracy of the previous matrix M_{k-1}'s (approximate) decomposition; similarly, the factorization error of M_k propagates to the following sequence of (intermediate) matrices, $M_{k+1}, ..., M_N$, of the chain.* This implies that a predecessor matrix which is poorly decomposed due to data noise or approximation error may negatively impact decomposition accuracies also for the successor matrices.

4.2 Noise Adaptation through Sample Assignment

Consequently, the inaccuracies resulting from each intermediate decomposition along the chain (whether due to data noise or factorization approximation error, Figure 2) need to be carefully considered during planning and resource allocation. The proposed *noise-profile adaptive tensor decomposition* (`NTTD`) algorithm adapts to (user provided or automatically discovered) a priori knowledge about noise by selecting a resource assignment strategy that best suits to the internal

and external noise profiles. More specifically, NTTD assigns Gibbs samples to the decompositions of the various individual matricizations in a way that maximizes the overall decomposition accuracy of the whole tensor.

4.3 Gibbs Sampling and (Internal) Decomposition Error

As we discussed in Section 3, the probabilistic tensor train decomposition process consists of several sequential probabilistic matrix decompositions. Consequently, any inaccuracies generated in any of the upstream decompositions will *propagate* to the downstream matrix decompositions along the "train" structure. In this section, we ignore the external data noise and focus on the impact of this internal noise generated due to decomposition inaccuracies. More specifically, we aim to investigate how to allocate Gibbs samples in a way that is sensitive to (a) the internal noise generated by the individual matrix factorizations, (b) the downstream (internal) noise propagation, and (c) their impacts to the accuracy of the overall tensor train decomposition.

As discussed in Section 3, Gibbs sampling is used for tackling the challenge of evaluating the predictive distribution of the posterior by approximating the expectation by an average of samples drawn from the posterior distribution through a Markov Chain Monte Carlo (MCMC) technique. As we see in Figure 2, for each intermediate matrix decomposition in PTTD, two factor matrices are generated: The U factor matrix is used to construct the core tensor corresponding to the current mode, whereas the V factor matrix is re-shaped as an input matrix for the successor decomposition step. Therefore,

- the accuracy of the U_k matrix has *direct* impact on the accuracy of one of the cores, whereas
- the accuracy of the V_k matrix *indirectly* influences accuracies of all downstream cores,

This observation, along with the observation that more samples can help provide better accuracy (*to certain degree*) in matrix factorization, can be used to improve the overall decomposition accuracy, to help allocate Gibbs samples to the different steps in tensor decomposition. More specifically, we argue that the number of samples for an intermediate matrix, M_k, should be allocated proportional to the size of the factor matrix, $size(U_k) + size(V_k)$, which reflects the number of unknowns to be discovered during the factorization of matrix M_k. In other words, the *internal decomposition error* sensitive sampling number, $L_{i\_err}(M_k)$, for matrix M_k can be computed as

$$L_{i\_err}(M_k) = L_{min}(M_k) + \lceil \gamma_{i\_err} \times (size(U_k) + size(V_k)) \rceil,$$

where γ_{i_err} is a scaling parameter such that the sum of all the sample counts is equal to the total number, $L_{i\_err(total)}$, of samples allocated for dealing with *internal decomposition errors* for the whole tensor decomposition:

$$L_{i\_err(total)} = \sum_{k=1}^{d-1} L_{i\_err}(M_k) \tag{3}$$

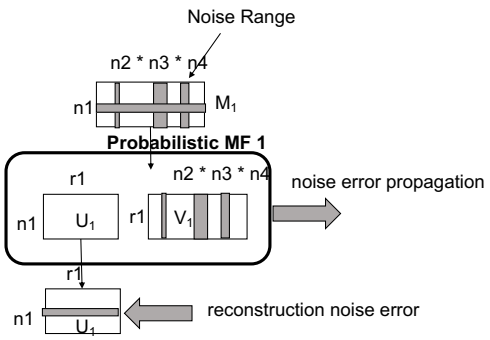


Fig. 3: Illustration of decomposition noise error propagation and reconstruction noise error for the first decomposition step

4.4 Gibbs Sampling and (External) Noise

Equation 7, above, helps allocate samples across intermediate decomposition phases. However, it ignores one crucial piece of information that may be available: *distribution of the noise across the input tensor.*

The basic probabilistic tensor train decomposition (Section 3) assumes the noise is uniformly distributed across the tensor. In the real world, however, noise is rarely *uniformly* distributed along the entire tensor. More often, we would expect that noise would be clustered across slices of the tensor (corresponding, for example, to unreliable information sources or difficult to obtain data). In many cases, even if we do not have precise knowledge about the cells that are subject to such noise or the amount of noise they contain, we may have a rough idea about the distribution of noise across the different modes [27]. As we experimentally show in Section 5, there is a direct relationship between the noise distribution across the tensor and the number of Gibbs samples it requires for accurate decomposition. Consequently, given a tensor with non-uniform noise distribution across different modes, uniform assignment of the number samples, $L_{n\_err}(M_k) = \frac{L_{n\_err(total)}}{d-1}$ (where $L_{n\_err(total)}$ is the total number of Gibbs samples for tackling the impact of noise) becomes ineffective. Therefore, in this section, we aim to answer the question

> *can we leverage rough information that may be available about noise distribution in improving the accuracy of the overall tensor train decomposition?*

Noise taints accuracy through two distinct mechanisms: (a) impact of noise during decomposition and, for applications (such as recommendation and prediction) that involve the recovery of missing entries in the tensor, (b) impact of noise during reconstruction . As we see in Figure 3, the noise in the input matrix partitions itself into the resulting factor matrices U and V. The factor matrix V_i is reshaped as input matrix for the following tensor train decomposition steps, therefore is involved in the propagation of the noise to downstream steps during

the tensor train decomposition process (Figure 4). The matrix, U_i, however, is separated into a factor matrix (for U_1) or more generally to a core tensor for factor $i > 1$, and thus impacts accuracy during reconstruction. We discuss these next.

Impact of Noise During Reconstruction. The noise reconstruction error taints the overall accuracy in the reconstruction process due to the matrix tensor multiplication operations involved in the recomposition of the (approximate) tensor. For example, if the ith object of U_k is polluted by the noise, after the reconstruction process, the complete slice $\tilde{\boldsymbol{\mathcal{X}}}_{*,\ldots,*,k(i),*,\ldots,*}$ will be tainted by the noise pollution from column $U_{k(i)}$ due to the matrix and tensor multiplication. Consequently, to account for the noise reconstruction error, the number of Gibbs samples should be proportional to the mode noise density, nd_k.

Impact of Noise During Decomposition. A naive approach to allocate the number of samples for a noisy matrix, M_k, is to allocate it proportional to its noise density, nd_k. However, since the probabilistic tensor train decomposition process follows a "train" structure, errors propagate downstream as shown in Figure 4. Consequently, allocating sampling number proportional to the noise density maybe not the best strategy.

As mentioned earlier, the Gibbs sampling algorithm cycles through the latent variables, sampling each one from its distribution conditional on the current values of all other variables. Due to the use of conjugate priors for the parameters and hyperparameters in the Bayesian PMF model, the conditional distributions derived from the posterior distribution are easy to sample from. In particular, the conditional distribution over the feature vector U_i, conditioned on the other features V_i, observed matrix cell value M_i, and the values of the hyperparameters are Gaussian:

$$\begin{aligned} p(U_i|M, V, \Theta_U, \alpha) &= \mathcal{N}(U_i|\mu_i, \Lambda_i^{-1}) \\ &\approx \prod_{i=1}^{n_1} \Big([\mathcal{N}(\widetilde{M}_{i,j}|U_i^T, V_i, \alpha^{(-1)})]^{n_{i,j}} \\ &\qquad \times p(U_i|\mu_U, \Lambda_U^{-1}) \Big). \end{aligned} \tag{4}$$

Note that the conditional distribution over the latent feature matrix U factorizes into the product of conditional distributions over the individual feature vector:

$$p(U|M, V, \Theta_U) = \prod_{i=1}^{n_1} p(U_i|M, V, \Theta_U) \tag{5}$$

We see that the conditional distributions over the V feature vectors and the V mode hyperparameters have exactly the same form.

Equations 4 and 5, along with figure 4, indicate how errors propagate downstream. In particular, in Figure 4, red columns of the first matricization, M_1, show the columns that are noise polluted. During the decomposition, the corresponding columns of resulting factor matrix V_1 (highlighted also in red) are also tainted with stronger noise than other columns of V_1. This tainting process

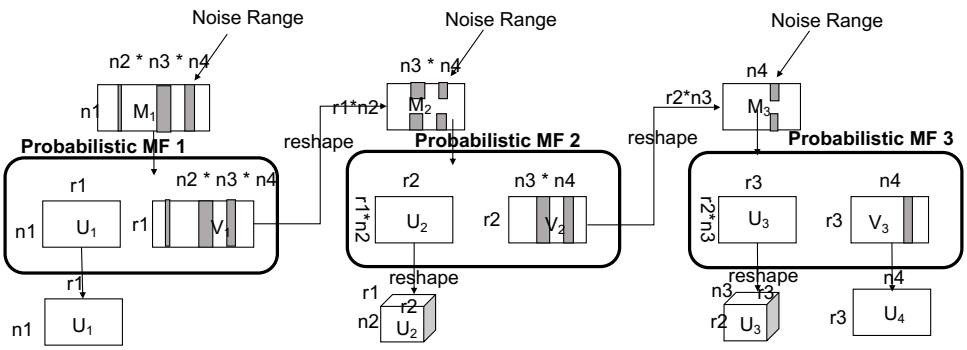


Fig. 4: Illustration of noise error propagation

flows downstream (*subject to matrix re-shape operations*) as shown in the figure 4. Consequently, for decomposition phase k, the number of samples should be proportional to

$$\sum_{j=k}^{d-1} \prod_{i=j+1}^{d} nd_i, \tag{6}$$

where, $\prod_{i=j+1}^{d} nd_i$ is the noise density of matricization of V_k on mode k . The $\sum_{j=k}^{d-1}$ operation, above, takes into account the accumulation process of the noise on all downstream decomposition steps.

Combining Decomposition and Reconstruction Impacts of Noise. Assuming that the decomposed tensor will be utilized for an application (such as recommendation) which necessitates reconstruction of the approximate tensor, we need to consider reconstruction and decomposition errors together when assigning the number of Gibbs samples. In other words, for an intermediate matrix, M_k, the number of samples must be allocated proportional to the sum of reconstruction and decomposition errors, i.e, $nd_k + \sum_{j=k}^{d-1} \prod_{i=j+1}^{d} nd_i$. This leads to the following formula for the number $L_{n\_err}(M_k)$ of samples:

$$L_{min}(M_k) + \lceil \gamma_{n\_err} \times (\sum_{j=k}^{d-1} \prod_{i=j+1}^{d} nd_i + nd_k) \rceil \times L_{n\_err(total)},$$

where γ_{n_err} is a scaling parameter such that the sum of all the sample counts is equal to the total number, $L_{n\_err(total)}$, of samples allocated for dealing with *noise errors*:

$$L_{n\_err(total)} = \sum_{k=1}^{d-1} L_{n\_err}(M_k). \tag{7}$$

4.5 Overall Sample Assignment

While considering the error propagation, both internal decomposition error (Section 4.3) and external noise error (Section 4.4) need to be accounted for. There-

Parameters	*Alternative values*
Dataset	Ciao; BxCrossing; MovieLens
Noise Density	**10**%; 20%; 30%;
Noise Intensity (σ)	**1**,3,5
Total Samples (L_{total})	**90**; 135; 180;
Min. Samples ($L_{min}(M_k)$)	$L_{total}/(3\times(d-1)) = L_{total}/9$

Table 1: Parameters – default values, used unless otherwise specified, are highlighted

fore, the combined sample assignment equation, for matricization, M_k, in the tensor train decomposition process, can be written as

$$\begin{aligned} L(M_k) = &\lceil \gamma_{n\_err} \times (\sum_{j=k}^{d-1} \prod_{i=j+1}^{d} nd_i + nd_k) \rceil \times L_{n\_err(total)} \\ &+ \lceil \gamma_{i\_err} \times (size(U_k) + size(V_k)) \rceil \times L_{i\_err(total)} \\ &+ L_{min}(M_k) \end{aligned} \tag{8}$$

where $L_{min}(M_k)$ is the minimum number of samples a (non-noisy) tensor of the given size would need for accurate decomposition and γ_{n_err} and γ_{i_err} are two scaling parameters, selected such that the total number of samples is equal to the number, L_{total}, of samples allocated for the whole tensor:

$$L_{total} = \sum_{k=1}^{d-1} L(M_k).$$

The parameters, γ_{n_err} and γ_{i_err}, also control the relative impacts of the internal and external noise. In the experiments, they are set such that the number of samples allocated to handle internal and external noise are the same.

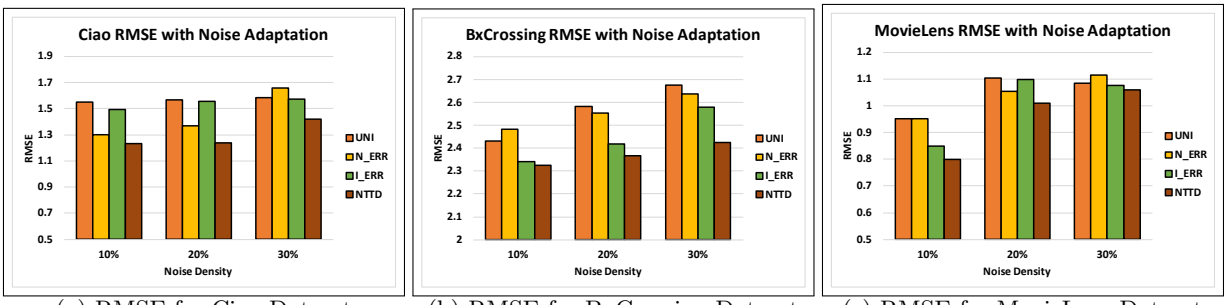
5 Experimental Evaluation

In this section, we report experiments that aim to assess the effectiveness of the proposed noise adaptive tensor train decomposition approach.

5.1 Experiment Setup

Key parameters and their values are reported in Table 1

Data Sets. In these experiments, we used three user-centered datasets: Ciao [39], MovieLens [15, 37] and BxCrossing [38]. Ciao dataset is represented in the form of $143 \times 200 \times 12 \times 4$ (density 5.68E-04) with the schema ¡user, product, category, helpfullness¿. BxCrossing dataset is represented in the form of $2599 \times 34 \times 16 \times 76$ (density 2.48E-0.5) with the schema ¡user, book, published year, user age¿. The MovieLens dataset is represented in the form of $247 \times 112 \times 48 \times 21$



(a) RMSE for Ciao Dataset (b) RMSE for BxCrossing Dataset (c) RMSE for MovieLens Dataset

Fig. 5: RMSE with Different Data Sets and Noise Densities ($L_{total} = 90$)

(density 8.86E-06) with the schema ¡user, movie, age, location¿. In Ciao and MovieLens data sets, the tensor cells contain rating values between 1 and 5 or (if the rating does not exist) a special "null" symbol. And for the BxCrossing dataset, the tensor cells contain rating values between 1 and 10.

Noise. To observe the different degrees of noise, we selected a random portion of the non-null cells and randomly perturbed the value. It is a worst-case scenario for `NTTD`, where the noise is distributed *uniformly on the tensor*; but the experiments show that even in this case, `NTTD` can take into account the noise density difference across the data modes, implied by the difference in corresponding data densities. Therefore, in the experiments, the noise density for different modes is approximated by the corresponding data density.

Alternative Strategies. We compare the proposed approach against other sampling strategies: uniform, internal-noise only, and external-noise only sample assignment:

- In *uniform* strategy (`UNI`), L_{total} is uniformly divided among the three matricizations in the tensor train decomposition and default `PTTD` is used for decomposition.
- In *internal-noise only* strategy (`I_ERR`), γ_{n_err} is set to zero in Equation 8, focusing the assignment to only internal decomposition error.
- In *external-noise only* strategy (`N_ERR`), γ_{i_err} is set to zero in Equation 8, focusing sample assignment to the impact of noise and its propagation.

Evaluation Criterion. We use the root mean squares error (RMSE) inaccuracy measure to assess the decomposition effectiveness. Each experiment was run 10 times with different random noise distributions and averages are reported.

Hardware and Software. We ran experiments on an eight-core CPU Nehalem Node with 16.00GB RAM. All codes were implemented in Matlab and run using Matlab R2016b. For tensor decomposition, we used MATLAB Tensor Toolbox Version 2.6

5.2 Discussion of the Results

Overview In Figure 5, we compare the performance of `NTTD` with noise-adaptive sample assignments against other strategies for different noise densities. As we

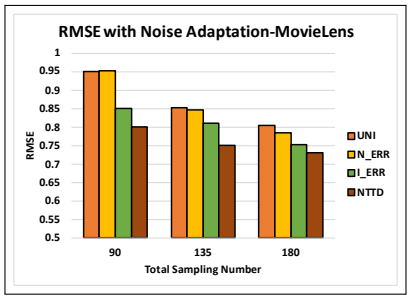


Fig. 6: RMSE with different num. of samples; i.e. L_{total} is 90, 135, or 180 (noise density 10%, noise intensity 1)

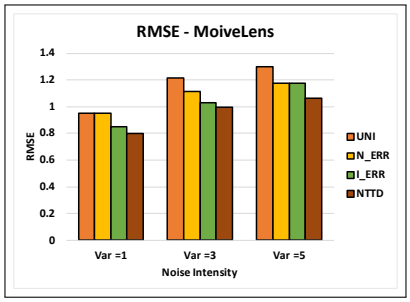


Fig. 7: RMSE with different noise intensities; i.e., σ is 1, 3, or 5 (noise density 10%)

see in this figure, the proposed `NTTD` strategy is able to allocate Gibbs samples effectively to significantly reduce RMSE relative to `PTTD` with uniform sample assignment. Moreover, we also see that internal- and external-only strategies that ignore part of the error can actually hurt the accuracy and perform worse than the uniform strategy. These show that the proposed noise-adaptive strategy is effective in leveraging rough knowledge about external noise distributions and internal decomposition errors to better allocate the Gibbs samples across the tensor.

Impact of the total number of samples. A key parameter of the `NTTD` algorithm is the number of total Gibbs samples. As we see in Figure 6, as we would expect, increasing the number of Gibbs samples helps reduce the overall decomposition error. Note that, among the four strategies, `NTTD` strategy is the one that provides most consistent and quickest drop in error. The figure shows the result for the MovieLens data; the results are similar also for the other data sets.

Impact of the noise intensity. In Figure 7, we consider the MovieLens data set with different noise intensities. As we expect, in this data set, increased noise corresponds to increased RMSE. However, `NTTD` provides the best results for all noise intensities considered. `NTTD` is also the best strategy for the other two data sets.

6 Conclusion

Recent research has shown that probabilistic techniques can ease the problem of overfitting caused by noise, especially on sparse data. However, existing techniques ignore potential non-uniformities in the noise distribution. In this paper, we proposed a novel noise-adaptive tensor train decomposition (`NTTD`) technique that leverages rough information about noise distribution to improve the tensor decomposition performance. `NTTD` decomposes each intermediate matrix probabilistically through Bayesian factorization. The noise profiles of tensor and their alignments are then leveraged to develop a strategy that considers the internal decomposition error as well as external to obtain a Gibb s sample assignment data noise best suits the noise profile of a given tensor.

References

1. E. Acar, *et al.* Multiway Analysis of Epilepsy Tensors. Bioinformatics, pages 10-18, 2007.
2. Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, N. Tang: Detecting Data Errors: Where are we and what needs to be done? PVLDB 9(12): 993-1004 (2016)
3. C. A. Andersson and R. Bro. The N-Way Toolbox for Matlab. Chem. and Intel. Lab. Systems, 52(1):1-4, 2000.
4. B. W. Bader, T. G. Kolda, *et al.* MATLAB Tensor Toolbox Version 2.5, Available online, January 2012. URL: http://www.sandia.gov/∼tgkolda/TensorToolbox.
5. B. W. Bader, R.A. Harshman, and T.G. Kolda. Temporal analysis of social networks using three-way dedicom. Sandia National Laboratories TR SAND2006-2161, 2006.
6. R. Balakrishnan, S. Kambhampati: SourceRank: relevance and trust assessment for deep web sources based on inter-source agreement. WWW 2010: 1055-1056
7. J. Ballani, L. Grasedyck, and M. Kluge, Black box approximation of tensors in hierarchical Tucker format, Linear Algebra Appl., 438 (2013), pp. 639-657.
8. J. Ballani and L. Grasedyck, A projection method to solve linear systems in tensor format, Numer. Linear Algebra Appl., 20 (2013), pp. 27-43
9. S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. WWW, 1998.
10. S. Chakrabarti, Dynamic personalized pagerank in entity-relation graphs. WWW 2007.
11. E. C. Chi and T. G. Kolda Making Tensor Factorizations Robust to Non-Gaussian Noise. tech. report, arXiv: 1010.3043v1, 2010.
12. W. Chu, Z. Ghahramani, Probabilistic Models for Incomplete Multi-dimensional Arrays. AISTATS 2009.
13. X. Chu, L. F. Ilyas, P. Papotti, Y. Ye: RuleMiner: Data quality rules discovery. ICDE 2014
14. R. A. Harshman, Foundations of the PARAFAC procedure: Model and conditions for an explanatory multi-mode factor analysis. UCLA Working Papers in Phonetics, 16:1-84, 1970.
15. F. M. Harper and J. A. Konstan. The MovieLens Datasets: History and Context. TiiS 5, 4, Article 19, 2015.

16. G. E. Hinton, C. Van. Keeping the neural networks simple by minimizing the description length of the weights. COLT (pp. 513). (1993)
17. S. Holtz, T. Rohwedder, and R. Schneider, The alternating linear scheme for tensor optimization in the tensor train format, SIAM J. Sci. Comput., 34 (2012), pp. A683-A713.
18. L. Grasedyck and W. Hackbusch, An introduction to hierarchical (H-) rank and TT-rank of tensors with examples, Comput. Methods Appl. Math, 3 (2011), pp. 291-304.
19. I. Jeon, E. Papalexakis, U. Kang, and C. Faloutsos. HaTen2: Billionscale tensor decompositions. ICDE 2015
20. B. Jeon, *et al.* SCouT: Scalable Coupled Matrix-Tensor Factorization - Algorithm and Discoveries. ICDE 2016.
21. M. I. Jordan, Z. Ghahramani, T. S. Jaakkola L. K. Saul. An introduction to variational methods for graphical models. Machine Learning, 37, 183.(1999)
22. U. Kang, *et al.* Gigatensor: scaling tensor analysis up by 100 times algorithms and discoveries. KDD 2012
23. P. M. Kroonenberg and J. De Leeuw, Principal component analysis of three-mode data by means of alternating least squares algorithms, Psychometrika, 45 (1980), pp. 6997.
24. H. Kim and H. Park, Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method, SIAM J. Matrix Anal. Appl., vol. 30, no. 2, pages 713-730, 2008
25. M. Kim and K.S. Candan. Efficient Static and Dynamic In-Database Tensor Decompositions on Chunk-Based Array Stores. CIKM, 2014.
26. A. Cichocki, N. Lee, I. Oseledets, A. H. Phan, Q. Zhao and D. Mandic 2016 Tensor networks for dimensionality reduction and large-scale optimization: part 1 low-rank tensor decompositions Found. Trends Mach. Learn. 9 249429
27. X. Li, K. S. Candan, M. L. Sapino, nTD: Noise Adaptive Tensor Decomposition. WWW 2017
28. I. V. Oseledets, Tensor-train decomposition, SIAM J. Sci. Comput., 33 (2011), pp. 2295-2317.
29. R. Salakhutdinov and A. Mnih, Probabilistic Matrix Factorization. NIPS 07, pages 1257-1264
30. J. Sun, S. Papadimitriou, and P. S. Yu. Window based tensor analysis on high dimensional and multi aspect streams. ICDM, pages 1076-1080, 2006.
31. A. Tjandra, S. Sakti, S. Nakamura Compressing Recurrent Neural Network with Tensor Train, NIPS 2017
32. L. Tucker, Some mathematical notes on three-mode factor analysis. Psychometrika, 31:279-311, 1966.
33. L. Xiong, *et al.* Temporal collaborative filtering with bayesian probabilistic tensor factorization. SDM 2010.
34. Y. Yang, D. Krompass, V. Tresp Tensor-Train Recurrent Neural Networks for Video Classification. ICML 2017
35. R. Zafarani, H Liu: Users joining multiple sites: Friendship and popularity variations across sites. Information Fusion 28: 83-89 (2016)
36. R. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.
37. http://grouplens.org/datasets/movielens/
38. http://www2.informatik.uni-freiburg.de/cziegler/BX/
39. http://www.public.asu.edu/∼jtang20/datasetcode/ truststudy.htm