TENSOR FUSION: LEARNING IN HETEROGENEOUS AND DISTRIBUTED DATA

by

Umut Şimşekli

B.S., Computer Science and Engineering, Sabancı University, 2008

M.S., Computer Engineering, Boğaziçi University, 2010

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2015

To the memory of my uncle Çetin Şimşekli,
my grandmother Naciye Şimşekli, and
my grandfather Kemal Yaşar Arslankaradede ...

ACKNOWLEDGEMENTS

I would like to thank to my supervisor Assoc. Prof. Ali Taylan Cemgil for all his support in every possible aspect of my journey at Boğaziçi, which took seven full years in total! It is impossible for me to fully express how grateful I am for being a PhD student (well, actually an 'apprentice') of yours; nevertheless, I would like to say that it has been a great feeling that having an academic advisor, a close friend, and sometimes a big brother at the same time. I would also like to thank to my PhD committee and jury, Assoc. Prof. Burak Acar, Prof. Ethem Alpaydın, Assist. Prof. Bert Arnrich, Prof. Ş. İlker Birbil, and Assist. Prof. Murat Saraçlar for the valuable feedback they have provided for this thesis.

I want to thank to my labmates İsmail Arı, Deniz Akyıldız, Doğaç Başaran, Cihan Camgöz, Taha Ceritli, Barış 'Evrim' Demiröz, Beyza Ermiş, Hakan Güldaş, Heysem Kaya, Alp Kındıroğlu, Hazal Koptagel, Barış Kurt, Orhan 'Gudi' Sönmez, Çağatay Yıldız, and all the other members of Perceptual Intelligence and Media Lab. for their help, support, and friendship. It is not possible to mention everyone here, but I would like to thank Prof. Lale Akarun, Assist Prof. Albert Ali Salah, Assist. Prof. Mehmet Gönen and the faculty members of CMPE for their help and suggestions.

I have been fortunate enough to collaborate with many great researchers. I would like to thank John Hershey, Andre Holzapfel, Jonathan Le Roux, Antoine Liutkus, Figen Öztoprak, Sertan Şentürk, Tuomas Virtanen, Y. Kenan Yılmaz, and the members of Gravi for having patience to collaborate with me.

A large part of my life has always been dedicated to music. I would like to thank to my fellow bandmates Erencan 'Teğmen' Ereren, Toros 'Maestro' Energin, Murat 'Başkan' Kirkit, Ozan 'Albay' Özcan, Nurtun 'Paşa' Şarman, and Nilgün 'Nünü' Yavaşoğlu just for being themselves. In all these things we have been through, I have collected more than enough stories to tell my grandchildren.

The last round is reserved for the family. My parents, Dr. Fatma and Dr. Coşkun Şimşekli, and my sister (Dr-to-be) Duygu Şimşekli, I want to thank to you all for making me what I am now and thank you for always supporting me no matter what the topic is. I feel extremely lucky to have a family like you. I want to thank to my best friend, my wife-to-be Gül Varol for being the source of joy and peacefulness of my life. A little smile of yours is enough for me to forget all the problems. Another big amount of thanks will go to the most relaxed couple that ever existed on Earth, Tolga and Cansu Birdal. Thank you for being so close to me even though there are thousands of kilometers in between.

Finally, I would like to thank to the Scientific and Technological Research Council of Turkey (TÜBİTAK). This thesis has been supported by the Ph.D. scholarship (2211) from TÜBİTAK.

ABSTRACT

TENSOR FUSION: LEARNING IN HETEROGENEOUS AND DISTRIBUTED DATA

In this thesis, we focus on *coupled matrix and tensor factorization models*; that provide a good modeling accuracy – practicality trade off for modeling large-scale and/or heterogeneous data that are collected from diverse sources. Our main concern in this thesis will be to develop inference methods for coupled tensor factorization models. We will first develop a rigorous tensor factorization notation, that aims to cover all possible model topologies and coupled factorization models. Our notation highlights the *partially separable* structure of tensor factorization models, which paves the way for developing parallel and distributed inference algorithms. Secondly, we will develop novel methods for making inference in coupled tensor factorization models. The proposed methods can be separated into three groups. In the first set of methods, we will focus on optimization-based approaches for making maximum likelihood and a-posteriori estimation of the latent variables. The second group of methods builds up on the first group and jointly estimates the relative weights and divergence functions, which play important role in coupled factorization models. Finally, in the third group, we will focus on full Bayesian inference, where we will develop several Markov Chain Monte Carlo methods for sampling from the posterior distributions of the latent variables. We will evaluate our methods on several challenging applications. We will develop novel factorization models for addressing challenging audio processing applications. We will also evaluate our distributed inference methods on large-scale link prediction applications, where we will report successful results in all of these applications.

ÖZET

TENSÖR TÜMLEŞTİRME: AYRI CİNSTEN VE DAĞITIK VERİLERDE ÖĞRENME

Bu tezde, büyük çaplı ve/ya farklı kaynaklardan toplanan ayrık veriler için iyi bir modelleme doğruluğu ve uygulanabilirlik ödünleşimi sağlayan *bağlaşımlı matris ve tensör ayrışımı modelleri* üzerinde yoğunlaşıyoruz. Bu tezdeki asıl amacımız bağlaşımlı tensör ayrışım modelleri için çıkarım yöntemi geliştirmek olacaktır. İlk olarak, olabilecek bütün model topolojilerini kapsamayı hedefleyen titiz bir tensör ayrışım simgelemi geliştireceğiz. Simgelemimiz, tensör ayrışım modellerinde paralel ve dağıtık çıkarım yapmaya olanak sağlayan *kısmen ayrışabilir* yapıyı vurgulamaktadır. İkinci olarak, bağlaşımlı ayrışım modellerinde çıkarım yöntemleri geliştireceğiz. Önerdiğimiz yöntemler üç ana grupta toplanabilir. İlk gruptaki yöntemlerde, enbüyük olabilirlik ve enbüyük sonsal kestirim için eniyileme tabanlı yöntemler üzerinde duracağız. İkinci grup yöntemler bağlaşımlı ayrışım modellerinde önemli bir rol oynayan *ilgili ağırlık* ve *ıraksayları* da beraber olarak kestirmektedir. Son olarak üçüncü grupta tam Bayesçi çıkarım yöntemleri üzerinde yoğunlaşacağız ve saklı değişkenlerin sonsal dağılımlarından örnek çekme amacıyla birçok Markov Zinciri Monte Carlo yöntemi geliştireceğiz. Önerdiğimiz yöntemleri birçok zorlu uygulamada sınayacağız. Bilhassa, ses işleme alanında görülen birçok zorlu uygulama için yeni ayrışım modelleri geliştireceğiz. Ayrıca, dağıtık çıkarım yöntemlerimizi büyük çaplı bağlantı kestirimi problemlerinde sınayacağız ve bu uygulamaların hepsi için başarılı sonuçlar göstereceğiz.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$\mathcal{B}$	Set of index configurations of a data block
$\mathcal{BI}$	Binomial distribution
B_s	Number of blocks in part s
$\mathcal{C}$	Set of index configurations
$d_p(\cdot)$	β divergence
$D_\nu(\cdot)$	Divergence function for X_ν
$\mathcal{G}$	Gamma distribution
$\mathcal{GG}$	Generalized gamma distribution
i_k	k^{th} index
I	Set of all indices
$\mathbb{I}$	Identity matrix
I_α	Set of indices of Z_α
$I_{0,\nu}$	Set of indices of X_ν
$\mathcal{IG}$	Inverse gamma distribution
$K(\cdot)$	Normalizing constant of the Tweedie distribution
M	Memory parameter of L-BFGS
$\mathcal{M}$	Multinomial distribution
M_ν	Binary mask for X_ν
$\mathcal{N}$	Gaussian distribution
N_x	Number of observed tensors
N_z	Number of latent factors
$\mathbb{P}$	Probability
p_ν	Power parameter for X_ν
$\mathcal{PO}$	Poisson distribution
$R_{\nu,\alpha}$	Coupling matrix
S	Number of parts
$\mathcal{S}$	Stable distribution
$\mathcal{S}\alpha\mathcal{S}$	Symmetric α-stable distribution

S_ν	Source tensor for X_ν
$\mathcal{T}$	Transition kernel of a Markov chain
$\mathcal{U}$	Uniform distribution
X_ν	Observed tensor
$\hat{X}_\nu$	Mean parameter for X_ν
Z_α	Latent factor
α	Characteristic exponent
β	Index parameter of β-divergence
γ	Step size parameter
$\Gamma(\cdot)$	Gamma function
$\delta(\cdot)$	Kronecker delta function
$\Delta(\cdot)$	Tensor valued collapse operation
ϵ	Step size
η	Step size parameter
θ	L-BFGS parameter
κ_ϕ	Prior parameter for the dispersion
λ_α	Index configuration for the collapsed model
Λ_ν	Parameter tensor for X_ν
μ'_s	Mean of the full conditional distribution of the sources
ν	Observed tensor index
ξ	Stochastic noise
Ξ	L-BFGS memory matrix
π	Element of a partition
ρ	L-BFGS parameter
σ_α	Variance of the proposal distribution
σ'_s	Variance of the full conditional distribution of the sources
Σ_α	Shape parameter
τ_ϕ	Prior parameter for the dispersion
ϕ_ν	Dispersion for X_ν
Φ	L-BFGS memory matrix

Φ_α	Inverse scale parameter
Ψ_α	Noise tensor for Z_α
$\Omega_{s,b}$	Data block
∇	Gradient
∂	Partial derivative

LIST OF ACRONYMS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
αMF	α-Stable Matrix Factorization
BFGS	Broyden–Fletcher–Goldfarb–Shanno Algorithm
BPMF	Bayesian Probabilistic Matrix Factorization
CGF	Cumulant Generating Function
CP	Canonical Polyadic Decomposition
CP	Compound Poisson Distribution
CPU	Central Processing Unit
DIGD	Distributed Incremental Gradient Descent
DSGLD	Distributed Stochastic Gradient Langevin Dynamics
EM	Expectation Maximization
GCTF	Generalized Coupled Tensor Factorization
GD	Gradient Descent
GPU	Graphics Processing Unit
GSL	GNU Scientific Library
HAMSI	Hessian Approximated Multiple Subsets Iteration
ICM	Iterative Conditional Modes
IGD	Incremental Gradient Descent
IS-NMF	Itakura-Saito NMF
KL	Kullback-Leibler
L-BFGS	Limited Memory BFGS
LD	Langevin Dynamics
MAP	Maximum A-Posteriori
MAPS	MIDI Aligned Piano Sounds
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MDCT	Modified Discrete Cosine Transform

MIDI	Musical Instrument Digital Interface
MF	Matrix Factorization
MH	Metropolis Hastings
MKL	Multiple Kernel Learning
ML	Maximum Likelihood
MSD	Million Song Dataset
MSE	Mean Squared Error
MUR	Multiplicative Update Rules
NMF	Non-negative Matrix Factorization
NMFD	Non-negative Matrix Factor Deconvolution
PARAFAC	Parallel Factorization
PSGLD	Parallel Stochastic Gradient Langevin Dynamics
RMSE	Root Mean Squared Error
ROC	Receiver Operating Characteristics
SADA	Space Alternating Data Augmentation
SAR	Signal-to-Artifact Ratio
SDR	Signal-to-Distortion Ratio
SGD	Stochastic Gradient Descent
SGLD	Stochastic Gradient Langevin Dynamics
SIR	Signal-to-Interference Ratio
SNR	Signal-to-Noise Ratio
STFT	Short-time Fourier Transform
QN	Quasi Newton

1. INTRODUCTION

We are living in the era of Big Data. Thanks to the current technological infrastructure, massive amounts of data are continuously produced and the cost of storing these data gets cheaper everyday. This circumstance has opened new horizons for many fields such as social media analysis [1], genomics [2], finance [3], audio processing [4], computer vision [5], and retail sales [6].

The growth in the size of the data has brought new challenges. There are two major challenges that rise with large-scale data [7]. The first one is the computational challenge: the 'traditional' methods become impractical as the size of the data gets larger. One would need efficient, *parallel* and *distributed* algorithms in order to be able to handle large amounts of data.

In order to address this challenge, numerous algorithms have been proposed for processing large-scale data within reasonable computational requirements. These methods include but not limited to the incremental/stochastic optimization methods [8–13], incremental/stochastic quasi-Newton methods [14–16], stochastic variational methods [17–19], and 'approximate' Monte Carlo methods [20–26].

The second challenge that appear in processing large-scale data is handling data-heterogeneity. The data are often collected from multiple sources at different time points, using different technologies. *Fusing* different but related data can improve the performance drastically, provided the different modes of the data contain sufficiently rich information and a proper model is established for jointly modeling the different modes.

Various research fields have focused on the data-heterogeneity problem, such as transfer learning [27–29], multiple-kernel learning [30–32], and coupled matrix and tensor factorizations [33–36]. Each of these fields has different application-specific concerns and therefore approaches the problem from a different perspective, if not

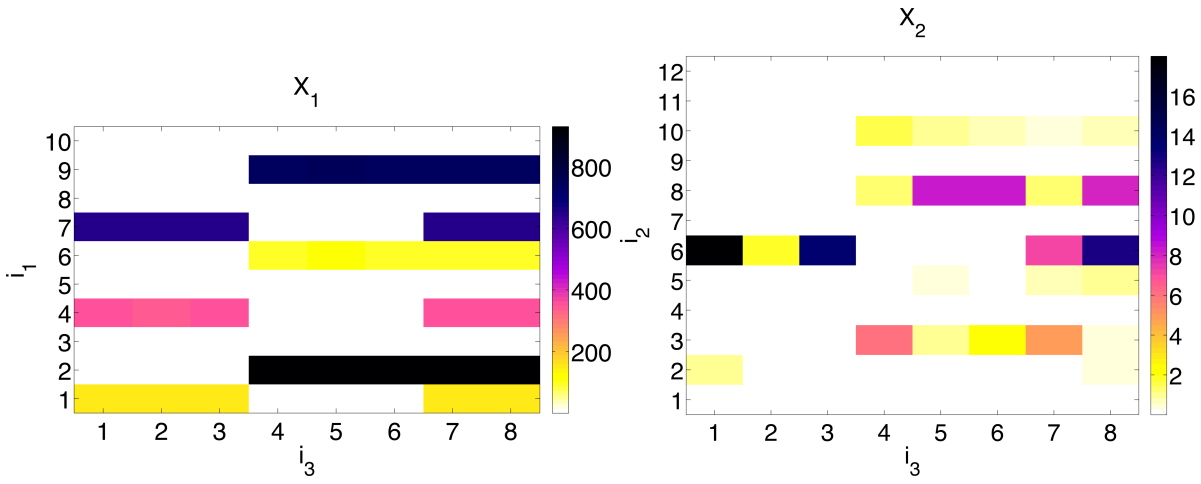


Figure 1.1. Example observed matrices. X_1 is the customer vs product matrix and X_2 is the monthly-sales vs product matrix.

completely different. We will give an overview of these methods in Section 1.5.

The main focus of this thesis will be on *coupled matrix and tensor factorizations* for large-scale and/or heterogeneous data. Coupled factorization methods are useful in various application areas such as audio processing [36–39], computational psychology [40, 41], bioinformatics [42–44], or link prediction [45, 46], where heterogeneous information from diverse sources are available and need to be combined for arriving at useful predictions. Examples of such situations are abound: for example for product recommendation, a product-buyer rating matrix can be enhanced with demographic information from the customer and connectivity information from a social network. In musical audio processing, one example is having a large collection of annotated audio data and symbolic music score information. The common theme in all such applications is the data fusion problem.

1.1. Example: Coupled Matrix Factorization

Let us consider a simple example, where we would like to model the unknown underlying structure of two observed matrices $X_1(i_1, i_3)$ and $X_2(i_2, i_3)$ that are depicted

in Figure 1.1. In this model, X_1 can be considered as the 'customer vs product' matrix, so that $X_1(i_1, i_3)$ denotes the total amount of money that the customer i_1 has spent on the product i_3. On the other hand, in the same context X_2 would contain the 'monthly-sales vs product' information, so that $X_2(i_2, i_3)$ is the total number of times that the product i_3 is sold during the month i_2, where $i_2 \in \{1, 2, \ldots, 12\} \equiv \{\text{January}, \ldots, \text{December}\}$. As we will illustrate in our experiments, this kind of datasets often appear in link prediction or recommender system applications, where the aim is to recommend new products to the customers by making use of all the information provided by X_1 and X_2.

The data matrices X_1 and X_2 share the index i_3 (i.e., they both contain the information of the same products), therefore we can assume that they are related. Firstly, let us take a closer look at each matrix one by one. It can be clearly seen that there are two patterns in X_1, where the first pattern is *active* when $i_3 = 1, 2, 3, 7, 8$, and the second pattern is active when $i_3 = 4, 5, 6, 7, 8$. We can interpret this as there are three groups of products, where the first group consists of the products 1, 2, and 3, the second group consists of the products 4, 5, and 6, and the third group consists of the products 7 and 8. We can observe that each group of products is always sold to the same customers and the amounts that the customers have spent on these products are very similar.

The main idea in factorization-based modeling is based on exploiting the fact that, even though there are three different groups of products, the columns of X_1 can be approximated as a linear combination of only two different patterns: for products 1, 2, 3, only the first pattern is active, for products 4, 5, 6, only the second pattern is active, whereas the products 7 and 8 can be expressed as a linear combination of these two patterns. In other words, we assume matrices like X_1 are *low-rank noisy* matrices and therefore we can approximate them by making use of low-rank approximations. For this example, we can decompose X_1 by using a rank 2 approximation, given as

follows:

$$X_1(i_1, i_3) \approx \sum_{i_4} Z_1(i_1, i_4) Z_3(i_4, i_3) \tag{1.1}$$

where, Z_1 and Z_3 are rank 2 matrices, i.e. $i_4 = 1, 2$. In this modeling strategy, Z_1 is typically called as the *dictionary* matrix, since its columns contain the patterns that occur in X_1. Similarly, Z_3 is called as the *activation* matrix, since its rows determine when the patterns in Z_1 are activated.

Now, let us take a look at X_2 in a similar fashion. Even though it has more rows than X_1 and their scales are very different, we can still observe that X_2 has a similar underlying structure: there are two major patterns that are synchronously activated with the patterns in X_1. However, X_2 has much more variation than X_1 as we can understand from its patterns to be not as crisp as the patterns of X_1. Nevertheless, we can still model X_2 by following a similar approach to the one presented in Equation 1.1 and come up with the following *coupled* model:

$$X_1(i_1, i_3) \approx \sum_{i_4} Z_1(i_1, i_4) Z_3(i_4, i_3), \quad X_2(i_2, i_3) \approx \sum_{i_4} Z_2(i_2, i_4) Z_3(i_4, i_3) \tag{1.2}$$

where, X_2 is decomposed into Z_2 and Z_3, where Z_3 simultaneously activates the patterns in Z_1 and Z_2. The factor Z_3 is the *shared factor* in both decompositions, making the overall model coupled. We call the factors Z_1 and Z_2 as *local factors* as they are only related to specific observations. This modeling strategy is also useful for modeling multi-modal time series, where i_3 would correspond to time and X_1 and X_2 would be two different modes of the same data. In this sense, Z_1 and Z_2 contain the bases for each mode and Z_3 forms the common activation structure over time.

The aim in this model is to estimate the latent factors Z_1, Z_2, and Z_3 given X_1 and X_2, as illustrated in Figure 1.2. In this problem, we need to solve the following

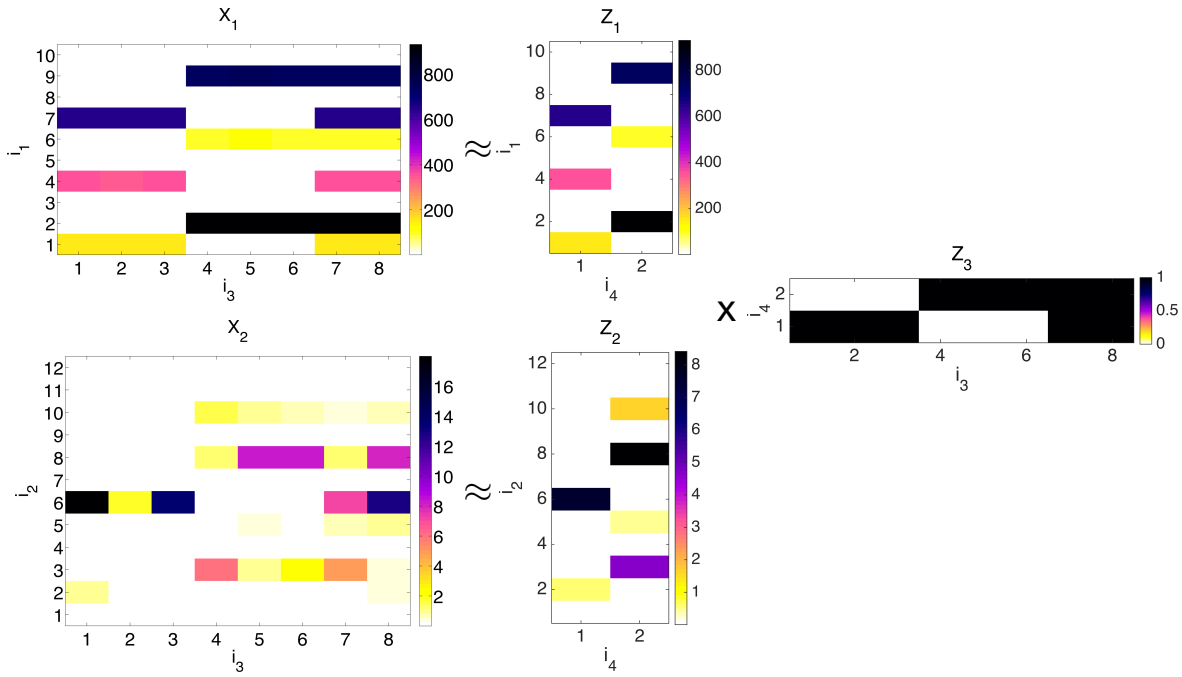


Figure 1.2. Example decomposition. Here the aim is to estimate the factors Z_1, Z_2, and Z_3, given the observed matrices X_1 and X_2.

optimization problem:

$$(Z_1^\star, Z_2^\star, Z_3^\star) = \underset{Z_1,Z_2,Z_3}{\arg\min} \left[\frac{1}{\phi_1} D_1(X_1||Z_1 Z_3) + \frac{1}{\phi_2} D_2(X_2||Z_2 Z_3) + \lambda r(Z_1, Z_2, Z_3) \right] \quad (1.3)$$

where D_1 and D_2 are *divergence functions* measuring the approximation error, the *dispersion parameters* ϕ_1 and ϕ_2 are the relative weights for the error in the approximation to each observed tensor, and $r(\cdot)$ is a regularization term.

Note that, a possible modeling approach for this model would be forming a so called 'concatenated' model, $X_{1,2} \approx Z_{1,2} Z_3$ where $X_{1,2} = [X_1; X_2]$ and $Z_{1,2} = [Z_1; Z_2]$, and trying to minimize $D(X_{1,2}||Z_{1,2} Z_3)$. This approach assumes that all the divergence functions and the dispersion parameters are exactly the same ($D_1 \equiv D_2$, $\phi_1 = \phi_2$); an assumption which would be violated in several real-world applications (including this example: each entry of X_1 is a real number whereas X_2 contains count data, besides

the scales of X_1 and X_2 are very different). On the other hand, as we will exemplify in the next chapters, in some factorization models it will not be possible to form a concatenated model due to model structures, therefore modeling the observations separately will be a requirement.

1.2. An Overview of the Inference Methods Used in the Thesis

As we will describe in more detail in the next chapter, the optimization problem given in Equation 1.3 can also be formulated as a maximum a-posteriori estimation problem that is given as follows:

$$(Z^{\star}_{1:3}) = \underset{Z_1,Z_2,Z_3}{\arg\max} \Big[\log \mathbb{P}(X_1; Z_1, Z_3, \phi_1) + \log \mathbb{P}(X_2; Z_2, Z_3, \phi_2) + \log \mathbb{P}(Z_1, Z_2, Z_3)\Big] \tag{1.4}$$

where $\mathbb{P}(Z_1, Z_2, Z_3)$ is the prior distribution of the latent factors, where $\log \mathbb{P}(Z_1, Z_2, Z_3)$ is equal to $-r(Z_1, Z_2, Z_3)$ up to constants. In this setting, minimizing $\frac{1}{\phi_1} D_1(X_1|\cdot)$ and $\frac{1}{\phi_2} D_2(X_1|\cdot)$ would be equivalent to maximizing $\log \mathbb{P}(X_1; \cdot)$ and $\log \mathbb{P}(X_2; \cdot)$, respectively.

In this thesis, depending on the application, we will be interested in finding the maximum likelihood (ML) or a-posteriori (MAP) estimates of the factors (as formulated in Equation 1.4) and characterizing the full posterior distribution of the latent factors $\mathbb{P}(Z_1, Z_2, Z_3|X_1, X_2)$ via full Bayesian inference. In the sequel, we will provide a brief overview of the methods that will be used throughout this thesis.

In the subsequent sections, in order to illustrate the methods, we will consider a simple model where we observe N *independent* observations $x = \{x_1, \ldots, x_N\}$, each of them being generated from the following probabilistic generative model:

$$z \sim \mathbb{P}(z) \tag{1.5}$$

$$x_i|z \sim \mathbb{P}(x_i|z), \quad \forall i = 1, 2, \ldots, N \tag{1.6}$$

where z is a latent variable whose posterior distribution is of interest.

1.2.1. Point Estimation via Optimization

In maximum a-posteriori estimation, our aim is to find the mode of the posterior distribution by solving the following optimization problem:

$$z^\star = \arg\max_z \mathbb{P}(z|x) = \arg\max_z [\mathbb{P}(x|z)\mathbb{P}(z)] = \arg\max_z [\log \mathbb{P}(x|z) + \log \mathbb{P}(z)]. \quad (1.7)$$

Equivalently, we can cast this problem as a cost minimization problem that is given as follows:

$$z^\star = \arg\min_z [f(z) + r(z)] \quad (1.8)$$

where $f(z) = -\sum_{i=1}^{N} \log \mathbb{P}(x_i|z)$ is the loss function, $r(z) = -\log \mathbb{P}(z)$ is the regularization term, and the total cost is $f(z) + r(z)$.

Gradient descent (GD) is a well-known optimization algorithm that is based on first order information. GD aims to find a local minimum of a cost function by iteratively taking steps in the negative direction of the gradient, given as follows:

$$z^{(t+1)} = z^{(t)} - \epsilon^{(t+1)} \Big(\nabla f(z^{(t)}) + \nabla r(z^{(t)}) \Big) \quad (1.9)$$

where t denotes the iteration number, $\epsilon^{(t)}$ is the step-size (also called as the learning rate), and ∇ is the gradient. In Section 3.1, we will develop a GD algorithm for making inference in small- and medium-scale coupled factorization models. Besides, we will also present an adaptive GD algorithm which would be suitable for non-negative factorizations.

As we will illustrate in our experiments, GD-based approaches can be very powerful in various applications. However, when the size of the observed data increases

(i.e., N gets larger), GD-based methods become impractical since they need to evaluate the gradient of $f(z)$ at each iteration, which requires a complete pass on all the data points. As opposed to GD, incremental gradient descent (IGD) and stochastic gradient descent (SGD) algorithms make use of a *noisy* gradient that is computed over a subset of the data points, instead of the whole data set. Accordingly, the update equation can be written as follows:

$$z^{(t+1)} = z^{(t)} - \epsilon^{(t+1)}\Big(\nabla f_{\Omega^{(t)}}(z^{(t)}) + \nabla r(z^{(t)})\Big) \tag{1.10}$$

where $\Omega^{(t)} \subset \{1, \ldots, N\}$ and

$$\nabla f_{\Omega}(z) = -\frac{N}{|\Omega|}\sum_{i\in\Omega} \nabla \log \mathbb{P}(x_i|z). \tag{1.11}$$

Here $|\Omega|$ denotes the number of elements in Ω. IGD and SGD differ in the way that they select the subsample $\Omega^{(t)}$ at each iteration. These algorithms provide great computational advantages over GD since usually $|\Omega| \ll N$. Provided certain conditions are met, these algorithms guarantee $z^{(t)}$ to converge to a local optimum [8,47]. In Section 3.2, we will develop a parallel and distributed IGD algorithm for making inference in large-scale coupled factorization models where the observations could be distributed among many nodes.

Due to their simplicity and nice theoretical properties, GD, SGD, and IGD are widely used in practical applications. However, since they are solely based on first order information, they can suffer from slow convergence. On the contrary, Newton's method utilizes second order information to circumvent such problems by applying the following update rules:

$$z^{(t+1)} = z^{(t)} - (H^{(t)})^{-1}\Big(\nabla f(z^{(t)}) + \nabla r(z^{(t)})\Big) \tag{1.12}$$

where $H = \nabla^2(f(z) + r(z))$ is the Hessian matrix. Even though this algorithm enjoys far better properties in terms of convergence speed, it requires huge computational

requirements: at each iteration, we need to estimate H and solve a large linear system, which make the method impractical except for very small-scale problems.

Quasi-Newton (QN) methods aim to make use of the second order information in an efficient way by replacing H with an approximate Hessian $\hat{H}$ which is often computed by using the past gradients that are computed throughout the optimization procedure:

$$z^{(t+1)} = z^{(t)} - \epsilon^{(t+1)}(\hat{H}^{(t)})^{-1}\Big(\nabla f(z^{(t)}) + \nabla r(z^{(t)})\Big). \tag{1.13}$$

However, these methods still suffer from computational complexity in large-scale settings where computing the gradients on the whole data set and storing them in the memory would not be possible.

By borrowing ideas from incremental optimization algorithms, in Section 3.3, we will develop a parallel and distributed *incremental* QN algorithm for large-scale coupled factorizations where the gradients and the approximate Hessian is computed over subsamples of the dataset:

$$z^{(t+1)} = z^{(t)} - \epsilon^{(t+1)}\hat{H}^{-1}_{\Omega^{(t)}}\Big(\nabla f_{\Omega^{(t)}}(z^{(t)}) + \nabla r(z^{(t)})\Big). \tag{1.14}$$

Since it incorporates second order information, this method is also expected to have nice convergence properties, while at the same time it would be computationally efficient since it makes use of incremental gradients.

1.2.2. Full Bayesian Inference via Markov Chain Monte Carlo

Maximum likelihood and a-posteriori estimation methods provide us useful and practical tools that can be used in various applications. However, in certain cases, they are prone to over-fitting since they fall short at capturing uncertainties that arise in the inference process. Instead of aiming to obtain single point estimates of the latent variables, full Bayesian inference aims to characterize the full posterior distribution over

the latent variables. Apart from being able to handle the uncertainties and therefore being robust to over-fitting, full Bayesian inference has many advantages over the point estimation methods in various tasks such as the model selection problem.

Monte Carlo methods are a set of numerical techniques to estimate expectations of the form:

$$\langle \varphi(z) \rangle_{\pi(z)} = \int \varphi(z)\pi(z)dz \approx \frac{1}{M}\sum_{t=1}^{M} \varphi(z^{(t)}) \tag{1.15}$$

where $\langle \varphi(z) \rangle_{\pi(z)}$ denotes the expectation of the function $\varphi(z)$ under the distribution $\pi(z)$ and $z^{(t)}$ are independent samples drawn from the target distribution $\pi(z)$, that will be the posterior distribution $\mathbb{P}(z|x)$ in our case. Under mild conditions on the test function φ, this estimate converges to the true expectation as M goes to infinity. The challenge here is obtaining independent samples from a nonstandard target density π.

Markov Chain Monte Carlo (MCMC) techniques generate subsequent samples from a Markov chain defined by a transition kernel $\mathcal{T}$, that is, one generates $z^{(t+1)}$ conditioned on $z^{(t)}$ as follows:

$$z^{(t+1)} \sim \mathcal{T}(z|z^{(t)}). \tag{1.16}$$

The transition kernel $\mathcal{T}$ does not need to be formed explicitly in practice; we only need a procedure that samples a new configuration, given the previous one. Perhaps surprisingly, even though these subsequent samples are correlated, Equation 1.15 remains still valid, and estimated expectations converge to their true values when number of samples M goes to infinity, provided that $\mathcal{T}$ satisfies certain ergodicity conditions. In order to design a transition kernel $\mathcal{T}$ such that the desired distribution is the stationary distribution, that is,

$$\pi(z) = \int \mathcal{T}(z|z')\pi(z')dz' \tag{1.17}$$

various strategies can be applied, where the most popular MCMC strategy being the Metropolis-Hastings (MH) algorithm [48]. In Section 5.1, we will develop block and collapsed Gibbs samplers for sampling from the posterior distribution of the latent factors in small- and medium-scale coupled factorization models. In Section 5.2, we will develop a stochastic gradient based parallel and distributed MCMC method that can gracefully scale up to large scale problems.

1.3. The Objectives and the Contributions of the Thesis

In this thesis, we aim to address the following problems:

(i) Maximum likelihood and a-posteriori estimation of the latent factors: The latent factors Z_1, Z_2, and Z_3 are clearly the most important variables in the model. These variables are particularly useful for analyzing the observed matrices or predicting missing parts of the observed matrices. High quality estimation of these variables is key to the success of an application.

(ii) Estimation of the dispersions: The dispersion parameters ϕ_1 and ϕ_2 play an important role in coupled factorizations as they form the balance between the approximation error to X_1 and X_2, for example observations may have been recorded using different and unknown scales, as the case in our running example. Typically, such weight parameters are selected manually [36, 39] and data is assumed to be suitably preprocessed. In a statistical setting, these relative weights are directly proportional to the observation noise variances and can be estimated directly from data.

(iii) Automatic selection of the divergences: Squared Euclidean distance is commonly used in tensor models, implicitly related to a conditionally Gaussian noise assumption. However, heavy-tailed noise distributions are often needed for robust estimation and more specific noise models are needed for sparse data, where Gaussian assumptions fall short. Choosing suitable divergence functions D_1 and D_2 becomes even more critical in coupled models due to the data heterogeneity, where X_1 and X_2 may have different statistical characteristics. In such cases, it is useful to choose a specific divergence for each observed matrix, where we call

total cost functions such as Equation 1.3 as *mixed divergences*.

(iv) Full Bayesian inference for the latent factors: Maximum likelihood and a-posteriori estimation of the latent factors provide us useful and practical tools that can be used in various applications. However, in certain cases, they are prone to over-fitting since they fall short at capturing uncertainties that arise in the inference process. Instead of aiming to obtain single point estimates of the latent variables, full Bayesian inference aims to characterize the full posterior distribution over the latent variables. Apart from being able to handle the uncertainties and therefore being robust to over-fitting, full Bayesian inference has many advantages over the point estimation methods in various tasks such as the model selection problem.

(v) Distributed and parallel inference: Modern computing infrastructure comprises of systems with hybrid architectures and the data may be distributed across several computers where each computer has multiple processors such as a multicore system or a graphics processing unit (GPU). Coupled tensor models are naturally adopted for such hybrid architectures. For example, X_1 and X_2 can be distributed in a large-scale setting. This motivates the development of distributed and parallel methods for inference.

(vi) Handling arbitrary model topologies: So far, we have motivated the open questions on the example model of Equation 1.2. However, in applications, as we will also demonstrate in Chapters 6, 8, and 7, one often needs to develop custom model topologies, where either the observed objects or the latent factors have multiple entities and cannot be represented without loss of structure using a matrix. To have this modeling flexibility for real world data sets that may consist of several tensors and require custom models, we would like to develop an algorithmic framework to handle a broad variety of model topologies.

<u>Our main contributions are as follows:</u>

(i) We rigorously develop the tensor factorization notation of [35], that aims to cover all possible model topologies and coupled factorization models. Our notation highlights the *partially separable* structure of tensor factorization models, which paves the way for developing parallel and distributed inference algorithms.

(ii) We develop two novel, inherently parallel optimization algorithms for maximum and a-posteriori estimation of the latent variables. As we demonstrate in our applications, these algorithms are particularly suited for distributed applications.

(iii) For jointly inferring the dispersions and the divergences, we develop two novel methods. The first method is focused on a particular family of noise distributions, whereas the second method is more general.

(iv) We develop novel Markov Chain Monte Carlo (MCMC) methods for making full Bayesian inference in the models. For moderate-sized data, we develop 'block' and 'collapsed' Gibbs samplers. For large-scale applications, we develop an inherently parallel stochastic gradient-based MCMC method, that can gracefully scale up to large-scale, distributed problems.

(v) We develop several novel factorization models that aim to solve various challenging problems. Our main application focus will be on audio processing. We report successful results in several applications.

(vi) Finally, we develop a novel factorization model with α-stable observations, that is particularly suited for impulsive or corrupted data that appear in several domains such as audio processing. We develop a novel MCMC method for sampling from the posterior distributions of the latent variables in this model.

1.4. Factorization-Based Data Modeling

In this section, we will describe the basics of factorization based modeling, and describe extensions such as coupled tensor factorizations and nonnegative decompositions.[1]

In many applications, data can be represented as a matrix, for example, the spectrogram of an audio signal (frequency vs time), a dataset of images (pixel coordinates vs instances), word frequencies among different documents (words vs documents), and the adjacency structure of a graph (nodes vs nodes) to name a few. Here the indices of the matrix correspond to the entities, and the matrix elements describe a relation between the two entities. Matrix Factorization (MF) models are one of the most widely

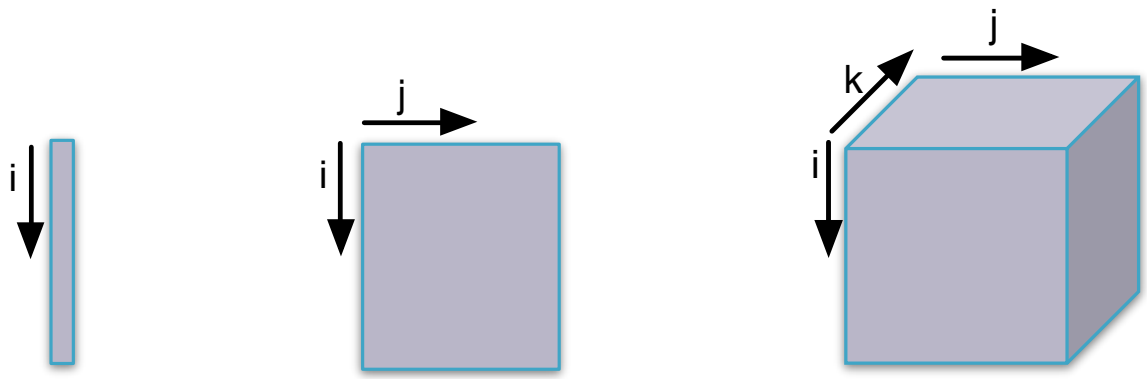
[1] This section is based on the material published in [49].

used methods for analyzing the data that involve two entities [50–53]. The goal in these models is to calculate a factorization of the form:

$$X_1(i,j) \approx \hat{X}_1(i,j) = \sum_k Z_1(i,k) Z_2(k,j) \tag{1.18}$$

where X_1 is the given data matrix, $\hat{X}_1$ is an approximation to X_1, and Z_1, and Z_2 are factor matrices to be estimated. Even though we have a single observed matrix in this model, we use a subscript in X_1 since we will consider factorization models that involve more than one observed matrix or tensor, later in this section. Here, X_1 is expressed as the product of Z_1 and Z_2, where Z_1 is considered as the *dictionary* matrix and Z_2 contains the corresponding *weights* (or activations). From another perspective, X_1 is approximated as the sum of inner products of the columns of Z_1 and the rows of Z_2, as illustrated at the top of Figure 1.4. Note that, if Z_1 would have been fixed, the problem would have been equivalent to basis regression where the weights (expansion coefficients) Z_2 are estimated [54]. In contrast, in matrix factorization the dictionary (the set of basis vectors) is estimated along with the coefficients. This modeling strategy has been shown to be successful in various fields including signal processing, finance, bioinformatics, and natural language processing [51].

Matrix factorization models are applicable when the observed data encapsulates the relation of two different entities (e.g., i and j in Equation 1.18). However, when the data involves multiple entities of interest, such as ternary or higher order relations it cannot be represented without loss of structure by using matrices. For example a multichannel sound library of several instances may be represented in the time-frequency domain conveniently as an object with several entities, say the power at each (frequency, time, channel, instance). One could in principle 'concatenate' each spectrogram across time and instances to obtain a big matrix, say (frequency $\times$ channel, time $\times$ instance) but this representation would obscure important structural information – compare simply with representing a matrix with a column vector. Hence one needs naturally multiway tables, the so-called *tensors*, where each element is denoted by $T(i,j,k,\dots)$. Here, T is the tensor and the indices $i,j,k,\dots$ are the entities. The number of distinct



(a) (b) (c)

Figure 1.3. Illustration of a) a vector $X(i)$: an array with one index b) a matrix $X(i,j)$ an array with two indices c) a tensor $X(i,j,k)$: an array with three or more indices. In this study, we refer vectors as tensors with one mode and matrices as tensors with two modes.

entities dictates the mode of a tensor. Hence a vector and a matrix are tensors of mode one and two respectively. Tensors are illustrated in Figure 1.3 and we will give a more precise and compact definition in Chapter 2.

For modeling multiway arrays with more than two entities the canonical polyadic decomposition [55, 56] (also referred as, CP, PARAFAC, or CANDECOMP) is one of the most popular factorization models. The model, for three entities, is defined as follows:

$$X_2(i,m,r) \approx \hat{X}_2(i,m,r) = \sum_k Z_1(i,k) Z_3(m,k) Z_4(r,k) \tag{1.19}$$

where the observed tensor X_2 is decomposed as a product of three different matrices. Analogous to MF models, this model approximates X_2 as the sum of 'inner products' of the columns of Z_1, Z_3, and Z_4 as illustrated at the bottom of Figure 1.4. This model has been shown to be useful in chemometrics [57], psychometrics [55], and signal processing [51].

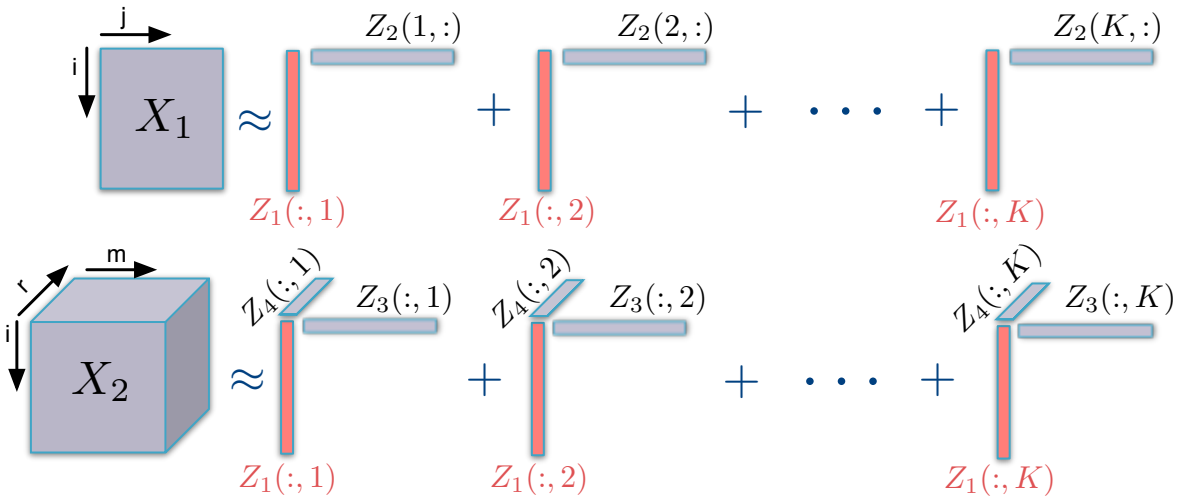


Figure 1.4. Coupled MF-PARAFAC illustration. The observed matrix X_1 is approximated as the sum of inner products of the columns of Z_1 and the rows of Z_2. Similarly, X_2 is approximated as the sum of 'inner products' of the columns of Z_1, Z_3, and Z_4. The overall model is coupled since the matrix Z_1 is shared in both factorizations. Here K denotes the size of the index k: $k \in \{1, \ldots, K\}$.

Tucker model [58] is another important model for analyzing tensors with three modes, which is a generalization of the PARAFAC model. The model is defined as follows:

$$X_3(i,j,k) \approx \hat{X}_3(i,j,k) = \sum_p \sum_q \sum_r Z_1(i,p) Z_2(j,q) Z_3(k,r) Z_4(p,q,r) \tag{1.20}$$

where X_3 is expressed as the product of three matrices ($Z_{1:3}$) and a 'core tensor' (Z_4). When the core tensor Z_4 is chosen as super diagonal ($Z_4(p,q,r) \neq 0$ only if $p = q = r$), Tucker decomposition reduces to PARAFAC.

1.4.1. Coupled Factorization Models

In certain applications, information from different sources are available and need to be combined for obtaining more accurate predictions [37, 40, 42, 45, 59]. In musical

audio processing, one example is having a large collection of annotated audio data and a collection of symbolic music scores as side information. Similarly, in product recommendation systems, a customer-product rating matrix can be enhanced with connectivity information from a social network and demographic information from the customer. For these problems, a single factorization model would not be sufficient for exploiting all the information in the data and we need to develop more comprehensive modeling strategies in order to be able to combine different data sources in a single factorization model. Such models are called as *coupled tensor factorization* models, where the aim is to simultaneously factorize multiple observed tensors that share a set of latent factors.

Let us consider an example coupled matrix-tensor factorization model where two observed tensors X_1 and X_2 are collectively decomposed as

$$\begin{aligned} X_1(i,j) &\approx \hat{X}_1(i,j) = \sum_k Z_1(i,k) Z_2(k,j) \\ X_2(i,m,r) &\approx \hat{X}_2(i,m,r) = \sum_k Z_1(i,k) Z_3(m,k) Z_4(r,k) \end{aligned} \tag{1.21}$$

where X_1 is decomposed by using an MF model and X_2 is decomposed by using a PARAFAC model. The factor Z_1 is the 'shared factor' in both decompositions, making the overall model coupled. Figure 1.4 illustrates this model. In our experiments, we will illustrate the usefulness of various factorization models on audio processing applications.

1.4.2. Non-Negative Tensor Factorizations

So far, we have described some of the most important matrix and tensor factorization models. However, even if two factorization models have the exact same topology (e.g., both models are MF models with the same number of parameters), depending on the constraints placed over the latent factors, the factorizations might have completely different interpretations. For instance, in the MF models, one option is not to restrict the factors by not placing any constraints over them. On the other hand, we can have

orthogonality constraints on the factors, where the factorization would turn into the principal component analysis.

Even if we place highly restrictive constraints such as orthogonality, the estimated factors would be dense and their physical interpretations in applications would be quite limited as long as their elements are allowed to take any positive and negative values. In this study, we will mainly consider *non-negative* factorization models [51], where we will restrict all the elements of the factors to be non-negative. Here, the non-negativity constraint implicitly imposes an *additive* structure on the model, where the contributions of the latent factors are always added since there will not be any cancellations due to negative values, as opposed to the aforementioned cases. Therefore, this strategy promotes sparsity on the factors since most of the entries in the factors would be close to zero in order the model to be able to fit the data, and more importantly the estimated factors will have physical interpretations that might be essential in many fields, such as audio processing. On the other hand, as we will describe in more detail in Chapters 2 and 5, by modeling tensors with probabilistic tensor factorization models, we essentially decompose the parameters of a probabilistic model that are non-negative by definition (e.g., the intensity of a Poisson distribution or the mean of a gamma distribution) and are constructed as the sum of non-negative sources [52]. In this modeling strategy, the non-negativity constraint on the factors is rather a necessity than an option.

Moreover, in our audio processing applications, we will model the energies of signals in the time-frequency domain that are known as the magnitude or power spectra. For modeling these spectra, the non-negativity constraint turns out to be very natural, since realistic sounds can be viewed as being composed of purely additive components. For example, music signals consist of multiple instruments, and the signal of each instrument consists of multiple notes played by the instrument. Speech signals consist of basic units such as phonemes and words. Cancellation of sounds happens only intentionally and in very specific scenarios, for example in echo cancellation systems.

1.5. Related Fields

Various other research fields have considered the data-heterogeneity problem. In this section, we will briefly describe two of the most important research fields, namely multiple kernel learning and transfer learning, that aim to model heterogeneous data in a systematic manner.

1.5.1. Multiple Kernel Learning

Multiple Kernel Learning (MKL) algorithms handle heterogeneous data by combining more than one kernel instead of using a single kernel in kernel machines such as support vector machines. Let us consider the support vector machine, which aims to find the linear discriminant function of the form $f(x) = w^\top \Phi(x) + b$, where x is $N \times D$ training data (N samples, D dimensions), and $\Phi(\cdot)$ is a mapping function. The optimal discriminant function is obtained by solving the following optimization problem:

$$\begin{aligned} \min_{w,\xi,b} \quad & \frac{1}{2}||w||_2^2 + C\sum_i \xi_i \\ \text{subject to} \quad & y_i(w^\top \Phi(x_i) + b) \geq 1 - \xi_i \quad \forall i \end{aligned} \tag{1.22}$$

where y represents the class labels ($y_i \in \{-1, 1\}$), b is the bias term, w denotes the weights, ξ is a collection of slack variables, and C is a trade-off parameter. The dual of this problem can be obtained by using the Lagrangian dual function, that is given as follows:

$$\begin{aligned} \max_{\alpha} \quad & \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j \underbrace{\Phi(x_i)^\top \Phi(x_j)}_{\kappa(x_i, x_j)} \\ \text{subject to} \quad & \sum_i \alpha_i y_i = 0, \quad C \geq \alpha_i \geq 0, \quad \forall i \end{aligned} \tag{1.23}$$

where α is a vector containing dual variables and $\kappa(\cdot)$ is the kernel function. In practice, the kernel function $\kappa(\cdot)$ is usually selected as the linear, polynomial, or the Gaussian kernel [30]. There are also application-specific kernel functions such as [60, 61].

MKL algorithms use multiple kernel functions instead of a single kernel function:

$$\kappa_\eta(x_i, x_j) = f_\eta\Big(\kappa_1(x_i, x_j), \kappa_2(x_i, x_j), \ldots, \kappa_K(x_i, x_j)\Big) \tag{1.24}$$

where $f_\eta(\cdot)$ is the combination function and $\kappa_i(\cdot)$ denotes the kernel functions to be combined. As investigated in detail in [30], combining multiple kernels can serve two main purposes: 1) different kernels would correspond to different similarity measures for the data, instead of picking a single kernel, we can combine the kernels in order to get a better similarity measure, 2) we can use different kernels for modeling different parts of the data, if different parts of the data are coming from multiple sources (i.e., they are heterogeneous). MKL algorithms aim to estimate the combination function $f_\eta(\cdot)$ along with the parameters of the kernel functions $\kappa_i(\cdot)$. MKL techniques have been shown to be useful for many applications such as emotion prediction [62], image retrieval [63], and bioinformatics [31]. A comprehensive overview of MKL algorithms is provided in [30].

Even though it might not be clear at a first sight, Potluru has showed that there are close connections between SVMs and factorization models [64]. We can indeed show that the objective function in Equation 1.23 can be rewritten as follows:

$$\min_{\alpha} \quad \frac{1}{2}\Big(\sum_{i\in A}\sum_{j\in A}\rho_{ij} - 2\sum_{i\in B}\sum_{j\in A}\rho_{ij} + \sum_{i\in B}\sum_{j\in B}\rho_{ij}\Big) - \sum_i \alpha_i \tag{1.25}$$

where we define $\rho_{ij} = \alpha_i\alpha_j\kappa(x_i, x_j)$, $A = \{i|y_i = 1\}$, and $B = \{i|y_i = -1\}$. By making use of the definition of $\kappa(x_i, x_j)$, we can rewrite Equation 1.25 as follows:

$$\min_{\alpha} \quad \frac{1}{2}\|\Phi(x_A)^\top\alpha_A - \Phi(x_B)^\top\alpha_B\|_2^2 - \sum_i \alpha_i \tag{1.26}$$

where x_A and x_B denotes the data samples that belong to different classes, and α is split into α_A and α_B accordingly. We can view the first term of this problem as a

factorization model with the following form

$$Z \approx \begin{bmatrix} \Phi(x_A) & -\Phi(x_B) \end{bmatrix}^\top \begin{bmatrix} \alpha_A & \alpha_B \end{bmatrix} \tag{1.27}$$

where each element of the matrix Z is set to zero and the second term in Equation 1.26 acts as a regularization term.

For certain choices for the combination function $f_\eta(\cdot)$ and the kernels $\kappa_k(\cdot)$ defined in Equation 1.24, the MKL problem can also be casted as a factorization problem. However, in the general case the MKL problem drifts apart from a MF problem.

1.5.2. Transfer Learning

Majority of the machine learning algorithms assume that the training and test data are in the same feature space and have the same underlying distribution. However, in some cases, the training data for a certain problem can be insufficient and we might have a sufficiently large set of data from a different but related problem, where the this data might have different statistical properties and might even lie in a different space. Transfer learning methods aim to *transfer* the information learned from a source task to a target task. In this section, we will consider the transfer learning examples provided in [27].

Let us first consider the supervised feature learning problem given in [28]. Here, the aim is to learn *higher level* features by using a source data and target data whose class labels are all known. The authors proposed an 'inductive' transfer learning approach, where the aim was to optimize the following optimization problem:

$$\begin{aligned} \min_{U,A_s,A_t} \quad & D(y_s||A_sUX_s) + D(y_t||A_tUX_t) + \gamma(||A_s||_{2,1} + ||A_s||_{2,1}) \\ \text{subject to} \quad & U \text{ is orthogonal} \end{aligned} \tag{1.28}$$

where y_s and y_t are the class labels, X_s and X_t are the data matrices, A_s and A_t

are the parameter matrices of the source and target, respectively. Here, the aim is to estimate the matrices U, A_s, and A_t. A_s and A_t will be the higher level features and U is responsible for the information transfer. It is easy to verify that this problem is actually a coupled factorization problem of the form of Equation 1.3, where the model can be rewritten as $y_s \approx A_s U X_s$ and $y_t \approx A_t U X_t$ with additional penalty terms.

Another example for transfer learning is the unsupervised feature construction problem given in [29]. Here, the aim is again to learn higher level features by using a source and target data. However, in this case the class labels y_s and y_t are not known. For this problem, the authors proposed a two-step algorithm, where the first stage solves the following optimization problem:

$$\begin{aligned} \min_{A_s,B} \quad & ||X_s - A_s B||_2^2 + \beta ||A_s||_1 \\ \text{subject to} \quad & ||B_j|| < 1, \quad \forall j \end{aligned} \tag{1.29}$$

where B will form a basis for both source and target data and A_s is the features for the source data. When we estimate the matrices A_s and B, we proceed to the next step, where we solve the following optimization problem:

$$\min_{A_t} \quad ||X_t - A_t B||_2^2 + \beta ||A_t||_1 \tag{1.30}$$

where B is fixed for this problem. By learning the basis B on the source data and fixing it from then on, we transfer information from source to the target. Similar to the supervised transfer learning example, this problem can also be seen as a factorization problem, where we first factorize the matrix $X_s \approx A_s B$, then we factorize the matrix $X_t \approx A_t B$.

There are many other forms of transfer learning, such as transferring parameters, transferring relational knowledge, transferring instances, transferring feature representations. A comprehensive survey of transfer learning is provided in [27].

1.6. Organization of the Thesis

This thesis is organized as follows. In Chapter 2, we describe our tensor factorization notation in detail and define our main probabilistic model for tensor factorizations. In Chapter 3, we describe three methods for obtaining the maximum likelihood and a-posteriori estimates of the latent factors; the first method is suited for moderate-sized data, whereas the remaining ones are suited for large-scale applications. In Chapter 4, we develop two methods for jointly estimating the mixed divergences along with the latent factors. In Chapter 5, we develop two different Monte Carlo methods for sampling from the full posterior distribution of the latent factors, where one of the methods can be used in large, distributed settings. In Chapters 6, 7, and 8, we present our experimental results on various applications. Finally, Chapter 9 concludes this thesis.

2. PROBABILISTIC MODELING OF COUPLED TENSOR FACTORIZATIONS

As demonstrated in Chapter 1, one often needs to develop custom model topologies, where either the observed objects or the latent factors can have multiple entities and cannot be represented without loss of structure using a matrix. To have this modeling flexibility for real world data sets that may consist of several tensors and require custom models, we would like to deal with a broad variety of tensor factorization models. In this chapter, we will rigorously develop the non-standard tensor notation of [35], that aims to cover all possible model topologies and coupled factorization models.

In our notation, a tensor is an N-way array, where we refer vectors as tensors with one index and matrices as tensors with two indices. We will denote tensors with capital letters, such as A, with its elements denoted by $A(i_1, i_2, \ldots, i_N)$. Here, A has N distinct indices $i_1, i_2, \ldots, i_N$. We let $I = [N]$ to be the index set of A, where $[N]$ denotes the set $\{1, 2, \ldots, N\}$. Each index i_k for $k \in I$ runs from 1 to its cardinality, denoted by s_k; i.e., we have $i_k \in \{1 \ldots s_k\}$, alternatively, $i_k \in [s_k]$. Each element of A is a real number and we write $A \in \mathbb{R}^{s_1 \times \cdots \times s_N}$.

Example 2.1. *Suppose we have a 3-way tensor $A(i_1, i_2, i_3)$ where $i_1 \in \{1, 2\}$, $i_2 \in \{1, 2, 3\}$ and $i_3 \in \{1, 2\}$. Then, the index set is $I = \{1, 2, 3\}$, $N = 3$, and the index sizes are $s_1 = 2$, $s_2 = 3$, $s_3 = 2$; hence $A \in \mathbb{R}^{2 \times 3 \times 2}$.*

In order to be able to handle a broad variety of tensor models and avoid unnecessary model specific notation, we index the elements of tensors with *index configurations*. An index configuration v is an N-tuple from the product space of the domains of all indices defined as:

$$v \in \mathcal{C}_I(I) \equiv \prod_{k \in I} [s_k] = [s_1] \times [s_2] \times \cdots \times [s_N].$$

We will call the set $\mathcal{C}$ as the set of index configurations. Given an index configuration v, we will write $v(k)$ to refer specifically to the value of the index i_k, i.e., $v(k) = i_k$. Given the index set I, a tensor element $A(i_1, i_2, \ldots, i_N)$ is denoted more compactly by $A(v)$.

Often, we need to iterate over configurations on a particular subset $I_\alpha \subset I$. Given a particular $v \in \mathcal{C}_I(I)$, we define v_α as the index configuration such that

$$v_\alpha(k) = v(k) = i_k$$

for all $k \in I_\alpha$ and, $v_\alpha(k) = 1$ for all $k \notin I_\alpha$. That is,

$$v_\alpha \in \mathcal{C}_I(I_\alpha) = \prod_{k \in I} [s_k^{\mathbb{1}(k \in I_\alpha)}] = [s_1^{\mathbb{1}(1 \in I_\alpha)}] \times [s_2^{\mathbb{1}(2 \in I_\alpha)}] \times \cdots \times [s_N^{\mathbb{1}(N \in I_\alpha)}] \tag{2.1}$$

where $\mathbb{1}(x) = 1$ if x is true, and $\mathbb{1}(x) = 0$ otherwise. We will call $\mathcal{C}_I(I_\alpha)$ as the set of index configurations of I_α with respect to the domain I where $I_\alpha \subset I$.

Example 2.2. *Following Example 2.1, the set of index configurations $\mathcal{C}_I(I)$ is as follows:* $\mathcal{C}_I(I) = \{(1,1,1), (1,1,2), (1,2,1) \ldots, (2,3,1), (2,3,2)\}$.

We define the *tensor contraction* as the operation of summing a tensor over a subset of its indices. Often, we are required to contract a tensor F with index set J over a subset of its indices to obtain a tensor $\hat{X}$ with index set $I_0 \subset J$. We define the index configurations $v \in \mathcal{C}_J(J)$, $v_0 \in \mathcal{C}_J(I_0)$, and $\bar{v}_0 \in \mathcal{C}_J(\bar{I}_0)$ where $\bar{I}_0 = J \setminus I_0$ denotes the indices that are not present in $\hat{X}$. We write

$$\hat{X}(v_0) = \sum_{\bar{v}_0} F(v_0 \cup \bar{v}_0) = \sum_{\bar{v}_0} F(v),$$

where the union is defined as $(v_0 \cup \bar{v}_0) \in \mathcal{C}_J(I_0 \cup \bar{I}_0)$ such that $(v_0 \cup \bar{v}_0)(k) = v(k)$ for all k such that $i_k \in I_0 \cup \bar{I}_0$ and $(v_0 \cup \bar{v}_0)(k) = 1$ otherwise.

Example 2.3. *Suppose we wish to sum the tensor A in Example 2.1 over i_2 to obtain*

Table 2.1. Illustration of different factorization models in the GCTF notation. Here N_x is the number of observed tensors, N_z is the number of latent factors, I is the set of all indices in the model, $I_{0,\nu}$ are the set of indices of X_ν, I_α are the set of indices of Z_α, R is the coupling matrix.

	N_x	N_z	I	$I_{0,\nu}$	I_α	R
MF (Eq.1.18)	1	2	$\{i,j,k\}$	$\{i,j\}$	$\{i,k\}$, $\{k,j\}$	$[1,1]$
PARAFAC (Eq.1.19)	1	3	$\{i,j,k,r\}$	$\{i,j,k\}$	$\{i,r\}$, $\{j,r\}$, $\{k,r\}$	$[1,1,1]$
TUCKER (Eq.1.20)	1	4	$\{i,j,k,p,q,r\}$	$\{i,j,k\}$	$\{i,p\}$, $\{j,q\}$, $\{k,r\}$,$\{p,q,r\}$	$[1,1,1,1]$
MF-PARAFAC (Eq.1.21)	2	4	$\{i,j,k,m,r\}$	$\{i,j\}$, $\{i,m,r\}$	$\{i,k\}$, $\{k,j\}$, $\{m,k\}$,$\{r,k\}$	$\begin{bmatrix}1&1&0&0\\1&0&1&1\end{bmatrix}$

the tensor $\hat{X}$. Here, the result $\hat{X}$ would be a tensor with two indices i_1 and i_3, with the index set as $I_0 = \{1,3\}$. In traditional notation, the contraction would be denoted as

$$\hat{X}(i_1, i_3) = \sum_{i_2} A(i_1, i_2, i_3).$$

Instead of dropping index i_2, we write the result of the contraction as

$$\hat{X}(i_1, 1, i_3) = \sum_{i_2} A(i_1, i_2, i_3)$$

adopting the convention that $\hat{X}(i_1, 1, i_3)$ and $\hat{X}(i_1, i_3)$ refer to the same object. Hence the summation over the index i_2 is equivalent to computing

$$\hat{X}(i_1, 1, i_3) = A(i_1, 1, i_3) + A(i_1, 2, i_3) + A(i_1, 3, i_3)$$

for each possible configurations of the pair i_1, i_3. This is exactly the elements of the sets of index configurations of I_0 given as

$$\mathcal{C}_I(I_0) = \{(1,1,1), (1,1,2), (2,1,1), (2,1,2)\}.$$

We have $\bar{I}_0 = I \setminus I_0 = \{2\}$ *and the set of index configurations of* $\bar{I}_0$ *is:*

$$\mathcal{C}_I(\bar{I}_0) = \{(1,1,1),(1,2,1),(1,3,1)\}.$$

Similarly, we define the *tensor product* as the operation of multiplying two tensors. Two tensors Z_1 and Z_2 with index sets I_1 and I_2 can be mutiplied to obtain the tensor product G on the index set $J = I_1 \cup I_2$. We write

$$G(w) = Z_1(v_1)Z_2(v_2)$$

where $w = v_1 \cup v_2$. More generally, for index sets $I_\alpha \subset J$ where $J = \cup_\alpha I_\alpha$ for $\alpha \in [N_z]$, we let $w \in \mathcal{C}_J(J)$ and $v_\alpha \in \mathcal{C}_J(I_\alpha)$. The product is a tensor G with the index set J, and Z_α are a collection of tensors, each with the index set I_α. When $w = v_1 \cup v_2 \cup \cdots \cup v_{N_z}$, we write

$$G(w) = \prod_{\alpha \in [N_z]} Z_\alpha(v_\alpha) = Z_1(v_1)Z_2(v_2)\ldots Z_{N_z}(v_{N_z}).$$

The generalized coupled tensor factorization (GCTF) framework is a statistical model for multiple observed tensors X_ν for $\nu \in [N_x]$. Each observed tensor X_ν is approximated by a model output tensor $\hat{X}_\nu$ that is obtained by the product of some latent tensors Z_α for $\alpha \in [N_z]$. The model is defined as follows:

$$X_\nu(u_\nu) \approx \hat{X}_\nu(u_\nu) = \sum_{\bar{u}_\nu} \prod_\alpha Z_\alpha(v_\alpha)^{R^{\nu,\alpha}} \tag{2.2}$$

Here, $u_\nu \in \mathcal{C}_I(I_{0,\nu})$ denotes the index configuration of X_ν and $v_\alpha \in \mathcal{C}_I(I_\alpha)$ denotes the index configuration of Z_α, where $I_{0,\nu}$ and I_α denote the index sets of an observed tensor X_ν and a latent factor Z_α, respectively. The set of all indices of a GCTF model is denoted by the index set $I = \{1, 2, \ldots, N\}$; it is the union of all indices used in the

Table 2.2. The correspondence between the elements of relational databases and the GCTF framework.

Relational Database	**GCTF Model**	**Notation**
View	Observed tensor ν	X_ν
Table	Latent tensor α	Z_α
Primary keys of a view	Visible index set of the ν'th observed tensor	$I_{0,\nu}$
All keys of all views	Visible index set	$\cup_\nu I_{0,\nu}$
Primary keys of a table	Set of latent indices of the α'th latent tensor	I_α
All keys of all tables	Set of latent indices	$\cup_\alpha I_\alpha$
A unique key combination of a view	Index configuration	u_ν
A unique key combination of a table	Index configuration	v_α
A data field of a view	Observed tensor Element	$X_\nu(u_\nu)$
A data field of a table	Latent tensor Element	$Z_\alpha(v_\alpha)$

model, both the indices of observed and latent tensors:

$$I = \left(\bigcup_\nu I_{0,\nu}\right) \cup \left(\bigcup_\alpha I_\alpha\right). \tag{2.3}$$

R is an $N_x \times N_z$ matrix with binary entries (0 or 1) that describes the coupling structure of a GCTF model, where each entry $R_{\nu\alpha}$ specifies if the model output tensor $\hat{X}_\nu$ is a function of Z_α:

$$R_{\nu\alpha} = \begin{cases} 1 & X_\nu \text{ is connected to } Z_\alpha \\ 0 & X_\nu \text{ is not connected to } Z_\alpha \end{cases}. \tag{2.4}$$

Finally, $\bar{u}_\nu \in \mathcal{C}_I(\bar{I}_{0,\nu})$ denote the index configurations that $\hat{X}_\nu$ is contracted on, where

$$\bar{I}_{0,\nu} = \left(\bigcup_{\alpha, R_{\nu\alpha}=1} I_\alpha\right) \setminus I_{0,\nu}. \tag{2.5}$$

The factors Z_α are called *local* if they are connected to a single observed tensor ($\sum_\nu R_{\nu\alpha} = 1$) and *shared* if they are connected to multiple observed tensors ($\sum_\nu R_{\nu\alpha} >$

1). Table 2.1 lists all the factorization models described earlier in the thesis as specific instances of the GCTF notation.

Our tensor notation is deliberately non-standard as our aim is developing a framework that can handle a broad class of model structures, including standard and application specific ones. The GCTF model can be viewed as a latent variable model for a relational database. Remember that a relational database contains several tables and information is linked across tables via keys, that are shared among tables. Information from distinct tables can be joined by using the shared keys across tables to create views. In this respect, each view is a particular projection of the data, as computed by a distinct query that joins a multiple of database tables. In GCTF, we envisage a process in the reverse direction; here the data is provided to us only by a fixed set of views, and we wish to approximately construct the latent tables of the database, that is to collectively factorize out the tables from the observed views. Analogous to a collection of tables in a relational database, GCTF defines a system of latent tensors Z_α, which share indices. The analogies are given in Table 2.2.

2.1. The Probabilistic Model

GCTF assumes the following probabilistic model on the observed tensors:

$$\mathbb{P}(X_\nu|\hat{X}_\nu, \phi_\nu, p_\nu) = \prod_{u_\nu} \mathcal{TW}_{p_\nu}\big(X_\nu(u_\nu); \hat{X}_\nu(u_\nu), \phi_\nu\big)^{M_\nu(u_\nu)}, \qquad \nu = 1, \ldots, N_x \tag{2.6}$$

Here, M_ν is a binary *mask* tensor of size X_ν, where

$$M_\nu(u_\nu) = \begin{cases} 1 & X_\nu(u_\nu) \text{ is observed} \\ 0 & X_\nu(u_\nu) \text{ is missing} \end{cases}. \tag{2.7}$$

Besides, $\mathcal{TW}$ denotes Tweedie distribution and $\hat{X}_{1:N_x}$ are the model output tensors that are defined in Equation 2.2. Tweedie distribution is an important special case of exponential dispersion models, characterized by three parameters, described in more detail in Appendix B. Tweedie densities $\mathcal{TW}_p(x; \hat{x}, \phi)$ can be written in the following

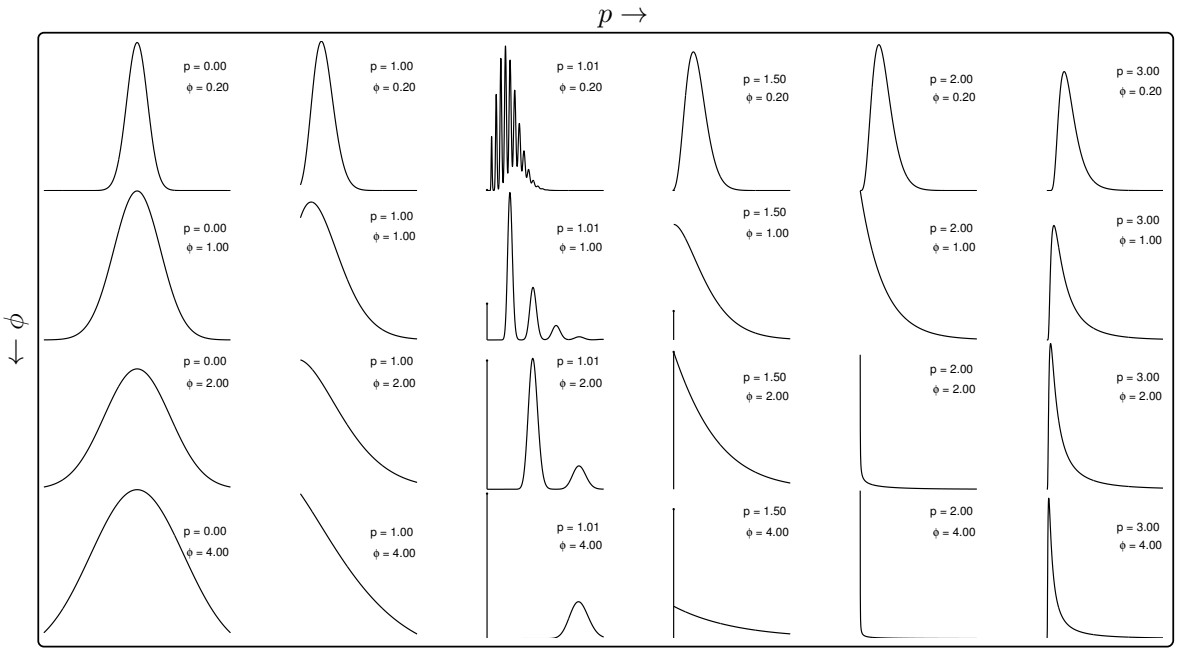


Figure 2.1. Illustration of the Tweedie distribution with different power and dispersion parameters.

moment form:

$$\mathbb{P}(x;\hat{x},\phi,p) = K(x,\phi,p)\exp\left(-\frac{1}{\phi}d_p(x||\hat{x})\right) \tag{2.8}$$

where $\hat{x}$ is the mean, ϕ is the dispersion, p is the power parameter and $d_p(\cdot)$ denotes the β-divergence defined as follows:

$$d_p(x||\hat{x}) = \frac{x^{2-p}}{(1-p)(2-p)} - \frac{x\hat{x}^{1-p}}{1-p} + \frac{\hat{x}^{2-p}}{2-p} \tag{2.9}$$

with $p = 2-\beta$. This model is also known as the *power variance* model, since the variance of the data has the following form: $\mathrm{var}(x) = \phi\hat{x}^p$.

By taking appropriate limits, it is easy to verify that $d_p(\cdot)$ is the Euclidean distance square, information divergence, and Itakura-Saito divergence for $p = 0, 1, 2$, respectively. In the probabilistic counterpart, different choices of p yield well-known important distributions such as Gaussian ($p = 0$), Poisson ($p = 1$), compound Poisson

Table 2.3. Tweedie distributions with corresponding normalizing constants and divergence forms. The general form of the distribution is given in Equation 2.8.

p	**Divergence**	**Distribution**	**Normalizing Constant**	**Divergence Form**
$2-\beta$	β-divergence	Tweedie	$K(x,\phi,p)$	$d_p(x\|\|\hat{x})$
0	Euclidean	Gaussian	$(2\pi\phi)^{1/2}$	$\frac{1}{2}(x-\hat{x})^2$
1	Kullback-Leibler	Poisson	$\frac{e^{x/\phi}\Gamma(x/\phi+1)}{(x/\phi)^{x/\phi}}$	$x\log\left(\frac{x}{\hat{x}}\right)-x+\hat{x}$
2	Itakura-Saito	Gamma	$\Gamma(1/\phi)(e\phi)^{1/\phi}x$	$\frac{x}{\hat{x}}-\log\left(\frac{x}{\hat{x}}\right)-1$
3	–	Inverse Gaussian	$(2\pi x^3\phi)^{1/2}$	$\frac{1}{2}\frac{(x-\hat{x})^2}{x\hat{x}^2}$

$(1<p<2)$, gamma $(p=2)$ and inverse Gaussian $(p=3)$ distributions. Excluding the interval $0<p<1$ for which no exponential dispersion model exists, for all other values of p, one obtains Tweedie stable distributions [65]. Figure 2.1 illustrates the Tweedie distribution for different p and ϕ values. Table 2.3 shows the normalizing constants and the divergence forms for $p\in\{0,1,2,3\}$.

An important property of the Tweedie distribution is that the normalization constant K does not depend on $\hat{x}$; hence it is easy to see that for fixed p and ϕ, solving a maximum likelihood problem for $\hat{x}$ is indeed equivalent to minimization of the β-divergence. For the familiar Gaussian case, we have

$$\mathcal{TW}_0(x;\hat{x},\phi)=(2\pi\phi)^{-1/2}\exp(-\frac{1}{\phi}d_0(x;\hat{x}))$$

where $d_0(x;\hat{x})=(x-\hat{x})^2/2$ and the dispersion is simply the variance. As for all admissible p, we have a similar form; Tweedie models generalize the established theory of least squares linear regression to more general noise models (restricted to identity link functions).

For regularization and incorporating prior knowledge, we place prior distributions over the latent factors. Depending on the application, we might consider different prior

distributions on the latent factors. For example, we can choose an exponential prior over the latent factors that can be used for all the cases of the power parameter p, shown as follows:

$$Z_\alpha(v_\alpha) \sim \mathcal{E}(Z_\alpha(v_\alpha); A_\alpha(v_\alpha)) \tag{2.10}$$

where $\mathcal{E}$ denotes the exponential distribution. We can also choose more sophisticated priors for individual cases of p. For instance, in the case of Poisson observations ($p = 1$), we can choose a gamma prior model

$$Z_\alpha(v_\alpha) \sim \mathcal{G}(Z_\alpha(v_\alpha); A_\alpha(v_\alpha), B_\alpha(v_\alpha))$$

where $\mathcal{G}$ denotes the gamma distribution. We further regularize ϕ and assume an inverse gamma prior on ϕ:

$$\phi \sim \mathcal{IG}(\phi, \tau_\phi, \kappa_\phi) \tag{2.11}$$

The definitions of the distributions that are mentioned in this thesis are given in Appendix A.

In this probabilistic setting, the power parameter p_ν determines the divergence function (e.g., D_1 or D_2 in Equation 1.3), the dispersion ϕ_ν determines the relative weight of the cost, and the mean parameter $\hat{X}_\nu$ is the output of the desired factorization as given in Equation 2.2.

Once we observe the tensors $X_{1:N_x}$, we would like to make inference in the probabilistic model defined in Equation 2.6. In the next chapter, we will focus on estimating the latent factors $Z_{1:N_z}$, given the dispersion $\phi_{1:N_x}$ and power parameters $p_{1:N_x}$. In particular, we will develop methods for obtaining point estimates, such as maximum likelihood (ML) or maximum a-posteriori (MAP):

(i) ML: $Z^{\star}_{1:N_z} = \underset{Z_{1:N_z}}{\arg\max} \log\Big[\mathbb{P}(X_{1:N_x}|Z_{1:N_z})\Big]$

(ii) MAP: $Z^{\star}_{1:N_z} = \underset{Z_{1:N_z}}{\arg\max} \log\Big[\mathbb{P}(X_{1:N_x}|Z_{1:N_z})\mathbb{P}(Z_{1:N_z})\Big]$.

In Chapter 4, apart from the estimation of the latent factors, we will develop methods for joint estimation of all the parameters including the dispersion and power parameters:

$$(Z_{1:N_z}, p_{1:N_x}, \phi_{1:N_x})^{\star} = \underset{Z_{1:N_z}, p_{1:N_x}, \phi_{1:N_x}}{\arg\max} \log \mathbb{P}(Z_{1:N_z}, p_{1:N_x}, \phi_{1:N_x}|X_{1:N_x}). \tag{2.12}$$

Finally, in Chapter 5, we will develop Monte Carlo methods for sampling from the full posterior distribution of the latent factors: $\mathbb{P}(Z_{1:N_z}|X_{1:N_x})$.

2.2. Extension: Alpha-stable Matrix Factorization

The GCTF framework with Tweedie observations provides very flexible and powerful modeling tools. However, when the data is impulsive (i.e. it may contain outliers) and it is not necessarily non-negative (i.e., the values can be also negative) the only Tweedie model we can use is the Gaussian distribution, which is not well-suited to such impulsive data since the distribution is not heavy-tailed.

In this section, we will focus on a matrix factorization (MF) model with a particular observation model and present αMF: a matrix factorization model that makes use of a family of heavy-tailed distributions as the observation model, so called the α-stable distributions. As we will describe in Section 2.2.1, similar to the Tweedie distributions, stable distributions have a rich structure and cover a broad range of noise distributions, where several important distributions appear as special cases. Stable distributions have been used in signal processing, especially in robust time-series modeling [66–70]. However, to the best of our knowledge, this is the first study to develop a MF framework with α-stable observations. Note that, αMF can be easily extended to the general case of coupled tensor factorization given in Equation 2.2. However, for notational simplicity, we will stick to the MF model throughout this section.

In this section, after a brief introduction to α-stable distributions, we describe αMF in detail. In Section 5.1.3, we will develop a Gibbs sampler for sampling from the posterior distributions of the latent variables.

2.2.1. α-Stable Distributions

Stable distributions are heavy-tailed distributions and appear as the limiting distributions in the generalized central limit theorem [71]. The probability density function of a stable distribution, $\mathcal{S}(x; \alpha, \beta, \sigma, \mu)$ cannot be written in closed-form except for certain special cases; however, the characteristic function of the distribution can be written as follows:

$$\varphi(\omega) = \int \exp(i\omega x)\mathcal{S}(x; \cdot)dx = \exp(\psi_{\alpha,\beta}(\omega) + i\mu\omega) \tag{2.13}$$

where Stable distributions are characterized by four parameters:

(i) $\alpha \in (0, 2]$ is called the characteristic exponent and determines the tail thickness of the distribution. As this parameter gets smaller, the distribution will be heavier-tailed, and therefore the observations will be more impulsive.

(ii) $\beta \in [-1, 1]$ is called the skewness parameter and determines whether the distribution is left- or right-skewed. The distribution is called symmetric α-stable ($\mathcal{S}\alpha\mathcal{S}$) if $\beta = 0$.

(iii) $\sigma \in (0, \infty)$ is called the scale or the dispersion parameter. It measures the spread of the random variable around its mode.

(iv) $\mu \in (-\infty, \infty)$ is the location parameter.

As special cases of the stable distributions, we obtain the Gaussian distribution ($\alpha = 2$, $\beta = 0$), the Cauchy distribution ($\alpha = 1$, $\beta = 0$), and the Lévy distribution ($\alpha = 0.5$, $\beta = 1$).

α-stable distributions are readily extended to the case of vectors, and in particular to complex random variables $x \in \mathbb{C}$. In this thesis, we will make use of the complex

isotropic α-stable distribution, which is shortly noted as $\mathcal{S}\alpha\mathcal{S}_c(x;\sigma)$, that reduces to $\mathcal{S}(x;\alpha,0,\sigma,0)$ in the real case [71, 72].

2.2.2. The Alpha-Stable Factorization Model

In this section, we describe the α-Stable Matrix Factorization (αMF) model. αMF models all the entries of an $i_1 \times i_2$ complex matrix X as independent and $\mathcal{S}\alpha\mathcal{S}_c$ distributed with dispersion parameter decomposed as follows:

$$X|W,H,\alpha \sim \prod_{fn} \mathcal{S}\alpha\mathcal{S}_c\Big(X(f,n);[\sum_k W(f,k)H(k,n)]^{1/\alpha}\Big). \tag{2.14}$$

with $N_x = 1$, $N_z = 2$, $Z_{1:2} \equiv \{W,H\}$, and $i_{1:3} \equiv \{f,n,k\}$. In order to preserve conjugacy, we assume generalized gamma priors on the latent factors:

$$\begin{aligned} W(f,k)|\alpha &\sim \mathcal{GG}(W(f,k);a_w,b_w,-2/\alpha) \\ H(k,n)|\alpha &\sim \mathcal{GG}(H(k,n);a_h,b_h,-2/\alpha), \end{aligned} \tag{2.15}$$

Finally, we assume uniform prior on α: $\alpha \sim \mathcal{U}(\alpha;(0,2]))$.

3. MAXIMUM LIKELIHOOD AND A-POSTERIORI ESTIMATION: OPTIMIZATION-BASED APPROACHES

In this chapter, we will focus on estimating the latent factors $Z_{1:N_z}$, given the dispersion $\phi_{1:N_x}$ and power parameters $p_{1:N_x}$. ML estimation of the factors $Z_{1:N_z}$ reduces to the problem of minimizing the β-divergence between the observations and the product of the latent factors, given as follows:

$$\begin{aligned} &\text{minimize} \quad \sum_{\nu} \sum_{u_\nu} \frac{1}{\phi_\nu} d_{p_\nu}\big(X_\nu(u_\nu) || \sum_{\bar{u}_\nu} \prod_{\alpha} Z_\alpha(v_\alpha)^{R^{\nu,\alpha}}\big) \\ &\text{subject to} \quad Z_\alpha(v_\alpha) \in C, \quad \forall \alpha \in [N_z], v_\alpha \in \mathcal{C}_I(I_\alpha) \end{aligned} \tag{3.1}$$

where, C is a constraint set, typically chosen as the non-negative real numbers $\mathbb{R}_+$. Here, the power parameter p_ν determines the cost function to be used for X_ν and the dispersion parameter ϕ_ν determines the relative weight of the approximation error to X_ν. ML and MAP estimation of the latent factors Z_α can be achieved via iterative methods, by fixing all factors $Z_{\alpha'}$ for $\alpha' \neq \alpha$ but one Z_α and updating in an alternating fashion.

In this chapter we will describe three methods for obtaining the point estimates. We will first present a gradient descent-based approach where we will derive multiplicative update rules that resemble practical implementations when the data and factors are non-negative. Then, we will develop two distributed incremental methods for large-scale applications.

3.1. Gradient Descent and Multiplicative Update Rules

Gradient descent (GD) is a well-known optimization algorithm that is based on first order information. GD aims to find a local minimum of a cost function by taking steps in the negative direction of the gradient. In order to estimate the latent factors,

we can iteratively apply the following GD updates, given as follows:

$$Z_{\alpha}^{(t)} = Z_{\alpha}^{(t-1)} - \epsilon^{(t)} \nabla_{Z_{\alpha}} \mathcal{L}(X_{1:N_x}) \tag{3.2}$$

where t denotes the iteration number and $\nabla_{Z_{\alpha}} \mathcal{L}(\cdot)$ is the gradient of the objective function defined in Equation 3.1. In this section, we will derive a GD algorithm for GCTF in detail.

Before deriving the partial derivatives with respect to Z_{α}, let us write down the following derivatives that will become handy at the final derivation step. The general form the derivative of the β divergence with respect to $\hat{x}$ is given as follows:

$$\frac{\partial d_p(x||\hat{x})}{\partial \hat{x}} = -x\hat{x}^{-p} + \hat{x}^{1-p} = \frac{\hat{x} - x}{\hat{x}^p}$$

Similarly, the derivative of the β-divergence $d_{p_\nu}(X_\nu(u_\nu); \hat{X}_\nu(u_\nu))$ with respect to an element of the model output tensor $\hat{X}_\nu(u_\nu)$ is given as follows:

$$\frac{\partial d_{p_\nu}(X_\nu(u_\nu); \hat{X}_\nu(u_\nu))}{\partial \hat{X}_\nu(u_\nu)} = \frac{\hat{X}_\nu(u_\nu) - X_\nu(u_\nu)}{\hat{X}_\nu(u_\nu)^{p_\nu}} \tag{3.3}$$

By using Equation 3.3, we can obtain the partial derivatives that are required in the GD algorithm as follows:

$$\begin{aligned}
\frac{\partial \mathcal{L}(X_{1:N_x})}{\partial Z_\alpha(v_\alpha)} &= \sum_\nu \frac{1}{\phi_\nu} \sum_{u_\nu} \frac{\partial d_{p_\nu}(X_\nu(u_\nu); \hat{X}_\nu(u_\nu))}{\partial \hat{X}_\nu(u_\nu)} \frac{\partial \hat{X}_\nu(u_\nu)}{\partial Z_\alpha(v_\alpha)} \\
&= \sum_\nu \left[R^{\nu,\alpha} \frac{1}{\phi_\nu} \sum_{\bar{v}_\alpha} \left(\frac{\hat{X}_\nu(u_\nu) - X_\nu(u_\nu)}{\hat{X}_\nu(u_\nu)^{p_\nu}} \right) \prod_{\alpha' \neq \alpha} Z_{\alpha'}(v_{\alpha'})^{R^{\nu,\alpha}} \right]
\end{aligned} \tag{3.4}$$

It is easy to verify that Equation 3.4 can be re-written in the following form:

$$\nabla_{Z_\alpha} \mathcal{L}(X_{1:N_x}) = \sum_\nu \left[R^{\nu,\alpha} \phi_\nu^{-1} \Delta_{\alpha,\nu}(\hat{X}_\nu^{1-p_\nu}) \right] - \sum_\nu \left[R^{\nu,\alpha} \phi_\nu^{-1} \Delta_{\alpha,\nu}(\hat{X}_\nu^{-p_\nu} \circ X_\nu) \right] \tag{3.5}$$

where the function $\Delta_{\alpha,\nu}(\cdot)$ is defined as follows:

$$\Delta_{\alpha,\nu}(A) = \left[\sum_{u_\nu \cap \bar{v}_\alpha} A(u_\nu) \sum_{\bar{u}_\nu \cap \bar{v}_\alpha} \prod_{\alpha' \neq \alpha} Z_{\alpha'}(v_{\alpha'})^{R^{\nu,\alpha'}} \right]. \tag{3.6}$$

When the model is non-negative, one possible option for enforcing non-negativity is to use projections where each $Z_\alpha(v_\alpha)$ is set to $\max(0, Z_\alpha(v_\alpha))$ at the end of each iteration. Another popular approach for developing optimization methods for non-negative models is called the multiplicative update rules (MUR). Provided all the factors and observed tensors are non-negative, we can observe that in the right hand side of the Equation 3.5 both terms are non-negative. By making use of this observation, MUR makes use of the following adaptive step size in order to preserve the non-negativity of the latent factors during the estimation process:

$$\epsilon^{(t)} = \frac{Z_\alpha^{(t-1)}}{\sum_\nu \left[R^{\nu,\alpha} \phi_\nu^{-1} \Delta_{\alpha,\nu}(\hat{X}_\nu^{1-p_\nu}) \right]} \tag{3.7}$$

where the division is element-wise. By plugging this step size into Equation 3.2, we obtain the multiplicative update rules as follows:

$$Z_\alpha \leftarrow Z_\alpha \circ \frac{\sum_\nu R^{\nu,\alpha} \phi_\nu^{-1} \Delta_{\alpha,\nu}(\hat{X}_\nu^{-p_\nu} \circ X_\nu)}{\sum_\nu R^{\nu,\alpha} \phi_\nu^{-1} \Delta_{\alpha,\nu}(\hat{X}_\nu^{1-p_\nu})}, \tag{3.8}$$

where $\circ$ is the element-wise product and the division operator is also element-wise. For MAP inference, the derivation procedure is the same up to adding one more term to the overall objective function. The monotonicity of these update rules for $N_x = 1$ is analyzed in [73].

For MAP inference, the objective given in Equation 3.1 will have an additional regularization term that involves Z_α. The resulting algorithm for MAP inference turns out to be similar to the ML schema. For exponential priors over the factors, the update

equation becomes a simple modification:

$$Z_\alpha \leftarrow Z_\alpha \circ \frac{\sum_\nu R^{\nu,\alpha}\phi_\nu^{-1}\Delta_{\alpha,\nu}(\hat{X}_\nu^{-p_\nu} \circ X_\nu)}{A_\alpha + \sum_\nu R^{\nu,\alpha}\phi_\nu^{-1}\Delta_{\alpha,\nu}(\hat{X}_\nu^{1-p_\nu})}. \tag{3.9}$$

For other conjugate priors, the update rules have similar forms [33].

The benefit of GD and MUR is that they can be applied on any tensor factorization model and are rather simple to implement. However, these methods are computationally extensive and do not scale to to large-scale problems.

3.2. Distributed Incremental Gradient Descent

In this section, we will be concerned with distributed and parallel inference in coupled tensor factorization models. We will develop a parallel and distributed algorithmic framework for GCTF where we will solve the optimization problem given in Equation 3.1 in large-scale setting. Here, we envision a distributed-data/distributed-processing scenario, where each observed tensor X_ν may reside at a different site (for example a cluster or a multicore machine) and each site has multiple processors with parallel, whilst limited, computational capacity. Our approach to optimization is inspired by the distributed and parallel stochastic gradient descent method for matrix factorizations (MF) in [12, 13, 74]. These methods make use of the conditional independence structure of factorization models, where the data is carefully segmented into mutually disjoint blocks that can be processed in parallel. On the other hand, our approach is geared towards the distributed data scenario and we never need to transmit any data across sites, that would be crucial for certain privacy critical applications.

Let us simplify and rewrite the optimization problem in Equation 3.1 by using the iterate u that covers all (ν, u_ν) pairs as follows:

$$\min_{Z_{1:N_z}} \quad \sum_{u=1}^{U} f_u(Z_1, \ldots, Z_{N_z}), \tag{3.10}$$

subject to the same constraints and $f_u(\cdot)$ is the appropriate cost function. If U is sufficiently small, more classical algorithms such as MUR can be used here. However, when U is large, operating on the entire cost function (i.e., computing the gradient on the full data tensors) becomes impractical. Besides, in large-scale applications the data is usually distributed among many processing units, which brings motivation to utilize distributed incremental methods.

Incremental gradient descent (IGD) methods are powerful algorithms that operate on a subset of samples $\Omega \subset [U]$ at each iteration, rather than the entire cost function [8]. The idea is to take a small step at each iteration in the opposite direction of the gradient computed on Ω:

$$Z_\alpha(v_\alpha)^{(t)} = Z_\alpha(v_\alpha)^{(t-1)} - \epsilon^{(t)}\Big(\frac{U}{|\Omega^{(t)}|}\sum_{u\in\Omega^{(t)}} \partial f_u(Z_{1:N_z})\Big) \tag{3.11}$$

where $\Omega^{(t)}$ denotes the subset of the data samples that are selected at iteration t and $|\Omega^{(t)}|$ denotes the number of elements in $\Omega^{(t)}$. For convergence, the step size $\epsilon^{(t)}$ must satisfy the following conditions:

$$\sum_{t=0}^{\infty}\epsilon^{(t)} = \infty, \quad \sum_{t=0}^{\infty}(\epsilon^{(t)})^2 < \infty \tag{3.12}$$

A typical choice for the step size is $\epsilon^{(t)} = \mathcal{O}(1/t)$.

The way of selecting the subsets Ω is very important and might result in different computational requirements depending on the problem structure. Here, we exploit the conditional independence structure of coupled tensor factorizations and construct a *partially separable* cost function as follows:

$$\min_{Z_{1:N_z}} \sum_{s=1}^{S}\sum_{b=1}^{B_s}\sum_{u\in\Omega_{s,b}} f_u(Z_{1,b},\ldots,Z_{N_z,b}) \tag{3.13}$$

where s is called as a *part*, b is called as a block, $\Omega_{s,b}$ denotes the data points in part

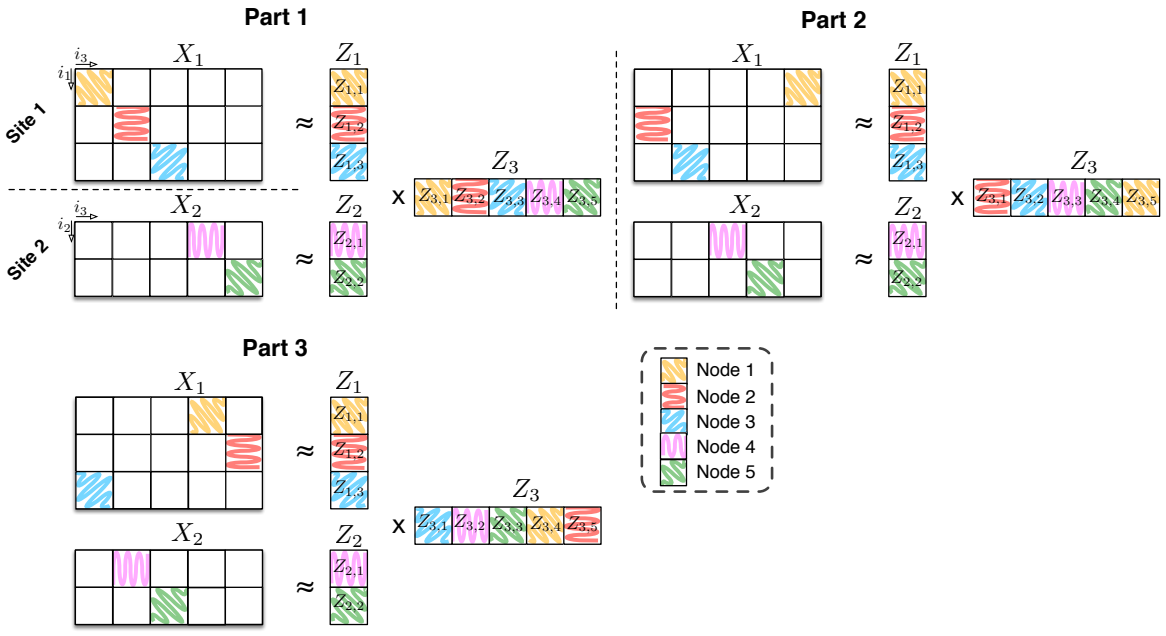


Figure 3.1. Illustration of DIGD on the coupled matrix factorization model given in Equation 1.2, $X_1 \approx Z_1 Z_3$, $X_2 \approx Z_2 Z_3$. There are 2 different sites and 5 nodes in total; the first site has 3 nodes and the second site has 2 nodes. The nodes are represented with different textures.

s and block b, S is the total number of parts, and B_s is the total number of blocks in part s. Example parts and blocks are illustrated in Figure 3.1.

In large-scale applications, these blocks and the corresponding factors $Z_{\alpha,b}$ will be fully distributed. The key point here is to form each part in such a way that the parameter spaces of the blocks in a part become disjoint; enabling these blocks to be updated in parallel. In other words, we aim to iterate sequentially over the index s and process the summation over b in parallel. We now describe how to form such blocks and parts for coupled tensor factorization that supports data locality.

We start by defining a partition for each index i_k. For the observed indices

$(k \in \cup_\nu I_{0,\nu})$, we define the partition

$$P_k = \{(\pi_k)_1, \ldots, (\pi_k)_{N_k}\} \tag{3.14}$$

on the set $[s_k]$, where $(\pi_k)_i$ are non-empty, disjoint subsets of $[s_k]$ and N_k denotes the number of subsets in P_k. We will not need partitions for the hidden indices $(k \notin \cup_\nu I_{0,\nu})$, however for notational consistency, we define a partition with a single element for each hidden index: $P_k = \{(\pi_k)_1\}$, where $(\pi_k)_1 = [s_k]$. Then a *block* of an observed tensor $X_\nu(u_\nu)$ is specified by the set of index configurations defined as follows:

$$u_\nu \in \mathcal{B}_I(I_{0,\nu}, \mathbf{b}_{s,b}, \mathbf{t}_{s,b}) = \prod_{k \in I} (\pi_k)_{\mathbf{b}_{s,b}(k)}^{\mathbb{1}(k \in I_{0,\nu}, \mathbf{t}_{s,b}(\nu)=1)} \tag{3.15}$$

where we define $S^1 = S$ and $S^0 = \{1\}$ for the set S, and $\mathbf{b}_{s,b}$ and $\mathbf{t}_{s,b}$ are arrays of integers that fully determine the structure of the block b in part s. Here, $\mathbf{t}_{s,b}(\nu) \in \{0, 1\}$ determines whether the block will contain the elements of X_ν and each $\mathbf{b}_{s,b}(k) \in [N_k]$ determines the element that will be selected from P_k, so that $(\pi_k)_{\mathbf{b}_{s,b}(k)}$ is the $\mathbf{b}_{s,b}(k)^{\text{th}}$ element of P_k. In this sense $\Omega_{s,b} \equiv \bigcup_\nu \mathcal{B}_I(I_{0,\nu}, \mathbf{b}_{s,b}, \mathbf{t}_{s,b})$.

We define the corresponding blocks for the factors $Z_{\alpha,b}$ similarly; we set $Z_{\alpha,b} \equiv \{Z_\alpha(v_\alpha) | v_\alpha \in \bigcup_\nu \mathcal{B}_I(I_\alpha, \mathbf{b}_{s,b}, \mathbf{t}_{s,b})\}$. In practice, the data is usually sparse and partitioning the data into balanced blocks can be an important but a highly nontrivial task. In our applications, we will utilize a simple heuristic approach for data-dependent index partitioning where each observed index is partitioned separately.

A part $\Omega_s = \bigcup_{b=1}^{B_s} \Omega_{s,b}$ is a collection of non-overlapping blocks. Here, for each pair of blocks $b \neq b'$, we have $\mathbf{b}_{s,b}(k) \neq \mathbf{b}_{s,b'}(k)$ for all

$$k \in \bigcup_{\substack{\nu \\ \mathbf{t}_{s,b}(\nu)=1 \\ \mathbf{t}_{s,b'}(\nu)=1}} I_{0,\nu}. \tag{3.16}$$

We construct the parts in such a way that, the collection of the parts will cover the

whole observed data: $\bigcup_s \Omega_s = \bigcup_s \bigcup_b \Omega_{s,b} \equiv \bigcup_\nu \mathcal{C}_I(I_{0,\nu})$.

Example 3.1. *Let us consider the example partitioning given in Figure 3.1. The blocks are obtained by partitioning the observed indices i_1, i_2, i_3 into $N_1 = 3$, $N_2 = 2$, and $N_3 = 5$ pieces, respectively. We have $S = 3$ parts, each of them containing $B_s = 5$ different blocks. In order to obtain the blocks in Part 1, we choose* $\mathbf{b}_{1,1} = [1,1,1,1,1]$, $\mathbf{t}_{1,1} = [1,0]$, $\mathbf{b}_{1,2} = [2,1,2,1,1]$, $\mathbf{t}_{1,2} = [1,0]$, $\mathbf{b}_{1,3} = [3,1,3,1,1]$, $\mathbf{t}_{1,3} = [1,0]$, $\mathbf{b}_{1,4} = [1,1,4,1,1]$, $\mathbf{t}_{1,4} = [0,1]$, $\mathbf{b}_{1,5} = [1,2,5,1,1]$, *and* $\mathbf{t}_{1,5} = [0,1]$. *Similarly, for Part 2 we have* $\mathbf{b}_{2,1} = [2,1,1,1,1]$, $\mathbf{t}_{2,1} = [1,0]$, $\mathbf{b}_{2,2} = [3,1,2,1,1]$, $\mathbf{t}_{2,2} = [1,0]$, $\mathbf{b}_{2,3} = [1,1,3,1,1]$, $\mathbf{t}_{2,3} = [0,1]$, $\mathbf{b}_{2,4} = [1,2,4,1,1]$, $\mathbf{t}_{2,4} = [0,1]$, $\mathbf{b}_{2,5} = [1,1,5,1,1]$, *and* $\mathbf{t}_{2,5} = [1,0]$.

By making use of the parts and their corresponding blocks, we develop a parallel and distributed incremental gradient descent (DIGD) algorithm for coupled tensor factorizations. Since the elements of the partitions are disjoint by definition and there are no common elements in two different blocks, the parameter spaces of the blocks in a part will also be disjoint by construction. This enables parallelism, since each $Z_{\alpha,b}$ can be updated independently:

$$Z_\alpha(v_\alpha)^{(t)} = Z_\alpha(v_\alpha)^{(t-1)} - \epsilon^{(t)}\Big(\frac{U}{|\Omega_{s,b}^{(t)}|} \sum_{u \in \Omega_{s,b}^{(t)}} \partial \log f_u(Z_{1,b}, \ldots, Z_{N_z,b})\Big) \tag{3.17}$$

where $v_\alpha \in \Omega_{s,b}$. For non-negative factors, we use projections by setting

$$Z_\alpha(v_\alpha)^{(t)} \leftarrow \max(0, Z_\alpha(v_\alpha)^{(t)}) \tag{3.18}$$

at each iteration.

Figure 3.2 summarizes DIGD for generalized coupled tensor factorization. One outer iteration of DIGD (lines 3-11) consists of several inner iterations (lines 5-10). After Ω_s is formed, we run the IGD updates in parallel for each block $\Omega_{s,b}$ as shown in Equation 3.17. The inner iterations are continued until all parts are processed; therefore all the entries of the observed tensors are seen. Here, first we assign the blocks of local

Input: $X_{1:N_x}$, R, $p_{1:N_x}$, $\phi_{1:N_x}$, η, γ, $Z^{(0)}_{1:N_z}$
$t \leftarrow 1$
repeat
  $\epsilon_t = (\eta/t)^\gamma$
  for $s = 1, \ldots, S$ **do**
    for $b = 1, \ldots, B_s$ **do in parallel**
      // $\forall \alpha = 1, \ldots, N_z$, $v_\alpha \in \Omega_{s,b}$
      $Z_\alpha(v_\alpha)^{(t)} = Z_\alpha(v_\alpha)^{(t-1)} - \epsilon^{(t)} \Big(\frac{U}{|\Omega^{(t)}_{s,b}|} \sum_{u \in \Omega^{(t)}_{s,b}} \partial f_u(Z_{1,b}, \ldots, Z_{N_z,b}) \Big)$
      // Optional projection step for non-negativity
      $Z_\alpha(v_\alpha)^{(t)} \leftarrow \max(0, Z_\alpha(v_\alpha)^{(t)})$
    $t \leftarrow t + 1$
until *convergence or* $t >$ *max_epochs*

Figure 3.2. DIGD (Distributed incremental gradient descent).

factors to the processors, then we circulate the blocks of the shared factors among the processors (see Figure 3.1). The proposed algorithm is a valid IGD algorithm; provided all the elements of the observed tensors are processed at each iteration, the proof of convergence proceeds in the same lines as in [8].

3.3. Distributed Incremental Quasi-Newton

DIGD provides a significant computational advantage when compared to sequential or batch methods thanks to its inherently parallel structure. However, since it is solely based on the first order information, it can suffer from slow convergence. It is well known in optimization, that second order methods such as Newton's method enjoy far better properties in terms of convergence speed, however are rather impractical for medium to large-scale problems as they typically require the estimation of a Hessian matrix and solving a large linear system. Quasi-Newton methods provide a practical alternative by incorporating local curvature information without inverting a large Hessian matrix. Yet these methods are batch methods that need explicit knowl-

Input: $X_{1:N_x}$, R, $p_{1:N_x}$, $\phi_{1:N_x}$, $z^{(1)} \equiv Z^{(1)}_{1:N_z}$
$t \leftarrow 1$
repeat
  Update $\rho^{(t)}$
  $\xi^{(t,1)} \leftarrow z^{(t)}$
  $H^{(t)} \leftarrow$ An approximate Hessian matrix at $z^{(t)}$
  for $s = 1, \ldots, S$ **do**
    for $b = 1, \ldots, B_s$ **do in parallel**
      $\xi_b^{(t,s)} \leftarrow \arg\min_z Q(z; \xi_b^{(t,s)}, \nabla f_{\Omega_{s,b}}(\xi_b^{(t,s)}), [H^{(t)}]_b, \rho^{(t)})$
    $\xi^{(t,s+1)} \leftarrow \xi^{(t,s)}$
  $z^{(t+1)} \leftarrow \xi^{(t,S+1)}$
  $t \leftarrow t + 1$
until *convergence or* $t >$ *max_epochs*

Figure 3.3. HAMSI (Hessian Approximated Multiple Subsets Iteration).

edge of the exact objective. In this section, we will develop a distributed incremental Quasi-Newton method for making inference in large-scale couple factorization models.

The idea of second order incremental methods has been investigated before. Bertsekas proposed such a method specifically designed for the least squares problem [75]. His proposal is an incremental version of the Gauss-Newton method. An extension of this method for general functions has recently been proposed by Gürbüzbalaban et al. [14]. They have shown linear convergence for the method under strong convexity and gradient growth assumptions. Moreover, their method requires the computation and inversion of exact Hessian matrices of component functions. In another study, an incremental aggregated quasi-Newton algorithm has been proposed, where the main idea is to update the quadratic model of one component function at each iteration [76]. Applying a quasi-Newton method with incremental (or stochastic) gradients is not straightforward as it may cause a *data consistency* problem (cf. [15, 16]).

In this section, we extend DIGD and present an incremental and parallel algorithm that also incorporates (approximate) curvature information for distributed large-scale optimization. Our experiences have confirmed that using second order information can help fast convergence even with incremental gradients. The proposed algorithm uses incremental gradients and incorporates a second order information into the optimization steps. This second order information comes from an approximation to the Hessian of the objective function. As we also work on multiple parts of the data, the algorithm is aptly called Hessian Approximated Multiple Subsets Iteration (HAMSI).

Before describing HAMSI in detail, let us simplify our notation one more time and rewrite the optimization problem in Equation 3.13 as follows:

$$\min_{Z_{1:N_z}} \sum_s \sum_b \sum_{u \in \Omega_{s,b}} f_u(Z_{1,b}, \ldots, Z_{N_z,b}) \equiv \min_z \sum_s \sum_b \sum_{u \in \Omega_{s,b}} f_u(z_b) \tag{3.19}$$

where z^b is defined as $[\mathbf{vec}(Z_{1,b})^\top, \ldots, \mathbf{vec}(Z_{N_z,b})^\top]^\top$ and similarly z is defined as $[z_1^\top, \ldots, z_{S_b}^\top]^\top$. The key idea of the algorithm is to use a local *convex* quadratic approximation at each iteration

$$Q(\xi; z, g, H, \rho) \equiv (\xi - z)^\top g + \frac{1}{2}(\xi - z)^\top H(\xi - z) + \frac{1}{2}\rho\|\xi - z\|^2 \tag{3.20}$$

for step size computation. Here, g denotes the incremental gradient computed on the subset of the data, H is (an approximation to) the Hessian of the objective function. The parameter ρ is crucial not only to bound the step length but also to control the oscillation of the incremental steps.

Figure 3.3 gives the generic form of HAMSI. Here, $\nabla f_{\Omega_{s,b}}(\cdot)$ denotes the gradient that is computed over $\Omega_{s,b}$. We also denote the s^{th} inner iterate of the t^{th} outer iteration with $\xi^{(t,s)}$, and $z^{(t)}$ are the outer iterates. It is important to note that the inner loop in Figure 3.3 (lines 8-9) computes the blocks of each inner step in parallel. The algorithm passes through the subsets of the observed data in a cyclic manner

similar to DIGD. Once a cycle is complete, one outer iteration is finished and the outer iterate is updated (line 11). Note that the same (approximate) matrix $H^{(t)}$ is employed at all inner iterations during the t^{th} cycle. However, the inner iterations use different blocks of $H^{(t)}$ denoted by the submatrix $[H^{(t)}]_b$. The parameter $\rho^{(t)}$ is also constant during the inner iterations and then it is updated with each outer iteration (line 4).

HAMSI assumes that all the elements of the factors can take values in the whole of $\mathbb{R}$. For non-negativity constraints on the latent factors, we can apply two-metric projections as described in [77]. However, currently we do not have a proof of convergence for HAMSI with such projections. We leave it as future work.

Above description of the algorithm overlooks several important implementation details; in particular, how to construct the quadratic approximation and how to solve the corresponding subproblems. Before presenting an implementation, where we exemplify these details, we first show in the next theorem the convergence properties of HAMSI under the quite generic setting given in Figure 3.3. We give a sketch of the proof of the theorem. The details along with the assumptions are given in Appendix C.

Theorem 3.1. *Consider the iterates $z^{(t)}$ of Algorithm HAMSI. Suppose that $\rho^{(t)} \to \infty$ as $t \to \infty$, and $\rho^{(t)}$ increases slowly enough so that $\rho^{(t)} = \mathcal{O}(\|\nabla f(z^{(t)})\|^{-\sigma})$ with $\sigma \in (0.5, 1)$ for large t. Then*

$$\liminf_{t\to\infty} \|\nabla f(z^{(t)})\| = 0.$$

Proof Sketch. The proof of this theorem depends on two intermediate results given by two lemmas. The first one establishes a bound on the difference between the true gradient of a block at $z^{(t)}$ and the evaluated gradients at $\xi^{(t,s)}$. The second lemma gives a bound on the error committed by taking incremental steps at inner iterates $\xi^{(t,s)}$ instead of the exact Newton step at $z^{(t)}$. The final theorem uses the boundedness of the objective function f and obtains the desired result by simple contradiction. □

Note that the condition on $\rho^{(t)}$ can be satisfied by choosing for instance

$$\rho^{(t)} = \max\{\mathcal{O}(t), \|\nabla f(z^{(t)})\|^{-2/3}\}.$$

We give an example implementation of HAMSI in Figure 3.3. Here, the approximate Hessian matrices $H^{(t)}$ are obtained using BFGS quasi-Newton update formula. In particular, the compact form of limited memory BFGS (L-BFGS) [78] is used in inner iterations to form the quadratic models, and to obtain their analytical solutions directly. L-BFGS allows the computation of $(H^{(t)} + \rho^{(t)}\mathbb{I})^{-1}v$ for a given vector v without forming any $V \times V$ matrices, and without any $\mathcal{O}(V^2)$ operations, where $V = \sum_\alpha \prod_k s_k^{[k \in I_\alpha]}$ is the total number of elements in the latent tensors. Moreover, the memory requirement is only $\mathcal{O}(MV)$, where M is the memory size. Quasi-Newton approximations require data pairs; a step in z and the corresponding change in the gradients. A limited memory quasi-Newton algorithm uses a collection of M such pairs to update the approximate Hessian; we denote the corresponding memory matrices of size $V \times M$ as Ξ and Φ. Since our algorithm requires the update to be done in blocks, the memory matrices are also held in blocks (see lines 10 and 11). The computations in lines 11-20 are nothing else but the direct use of compact form formulas; we refer to [78] for details. The algorithm sets the initial Hessian approximation to a multiple of the identity $\theta\mathbb{I}$, this is why incremental gradient steps are taken in the first cycle; i.e. when $t = 1$ (line 20). We should also note that the condition in line 9 ensures that the memory (therefore $H^{(t)}$) is updated once in a cycle, and the differences are computed by using the same component function to provide consistency. Finally, the reduce primitive appearing in the algorithm, accumulates local data from each node and distributes the results back.

Input: $X_{1:N_x}$, R, $p_{1:N_x}$, $\phi_{1:N_x}$, η, γ, $z^{(1)} \equiv Z^{(1)}_{1:N_z}$, M

$t \leftarrow 1$

repeat

  $\rho^{(t)} = (\eta t)^{\gamma}$, $\xi^{(t,1)} \leftarrow z^{(t)}$

  for $s = 1, \ldots, S$ **do**

    for $b = 1, \ldots, B_s$ **do in parallel**

      $g_b^{(t,s)} = \nabla f_{\Omega_{s,b}}(\xi_b^{(t,s)})$

      if $t > 1$ **then**

        if $s = (t - 1 \mod S) + 1$ **then**

          $\Xi_b = \left[\Xi_b(:, 2:M),\ \xi_b^{(t,s)} - \xi_b^{(t-1,s)}\right]$,

          $\Phi_b = \left[\Phi_b(:, 2:M),\ g_b^{(t,s)} - g_b^{(t-1,s)}\right]$

          $W_b = [\Phi_b \ \ \theta\Xi_b]$, $A_b = \Xi_b^{\top}\Phi_b$, $E_b = \Xi_b^{\top}\Xi_b$, $C_b = \Phi_b^{\top}\Phi_b$

          reduce

            $A = \sum_b A_b$, $E = \sum_b E_b$, $C = \sum_b C_b$

          $L_A = \mathrm{tril}(A)$, $D_A = \mathrm{diag}(A)$, $U_A = \mathrm{triu}(A)$

          $V = \begin{pmatrix} C & A^{\top} \\ A & E \end{pmatrix}$, $R = \begin{pmatrix} -D_A & L_A^{\top} \\ L_A & \theta E \end{pmatrix}^{-1}$, $N = (I - \frac{1}{\theta} RV)^{-1}$

          reduce

            $\bar{g} = \sum_b W_b g_b^{(t,s)}$

          $\xi_b^{(t,s)} \leftarrow \xi_b^{(t,s)} - \frac{1}{\rho^{(t)}\theta}(g_b^{(t,s)} + \frac{1}{\theta} W_b N R \bar{g})$

        else

          $\xi_b^{(t,s)} \leftarrow \xi_b^{(t,s)} - \frac{1}{\rho^{(t)}\theta} g_b^{(t,s)}$

    $\xi^{(t,s+1)} \leftarrow \xi^{(t,s)}$

  $z^{(t+1)} \leftarrow \xi^{(t,S+1)}$

  $t = t + 1$

until *convergence or* $t >$ *max_epochs*

Figure 3.4. HAMSI with L-BFGS updates.

4. LEARNING MIXED DIVERGENCES

So far, we have focused on the estimation of the latent factors. However, in coupled models, the success of a method may hinge on a good setting of the dispersion and power parameters, $\phi_{1:N_x}$ and $p_{1:N_x}$, yet manual selection is not straightforward especially when the number of observed tensors, N_x, is large.

In order to illustrate the importance of the dispersion and power parameters, let us consider the following sparse non-negative dictionary learning problem:

$$\min_{Z_1, Z_2 \geq 0} \quad \frac{1}{2} \sum_{i,j} \Big(X_1(i,j) - \sum_k Z_1(i,k) Z_2(k,j) \Big)^2 + \lambda \Big(\sum_{ik} |Z_1(i,k)| \Big) \tag{4.1}$$

where, we would like decrease the approximation error to X_1, while trying to keep the elements of the dictionary matrix Z_1 close to zero as much as possible. Here λ is the sparsity parameter that is often chosen manually.

Even though there is only one observed matrix X_1 in this problem, we can still rewrite this problem as a coupled matrix factorization problem, that is given as follows:

$$X_1(i,j) \approx \sum_k Z_1(i,k) Z_2(k,j) \tag{4.2}$$

$$X_2(i,k) \approx Z_1(i,k) \tag{4.3}$$

where X_2 is a matrix whose entries are all set to zero. We can verify that, maximum likelihood estimation in this coupled model is indeed equivalent to the optimization problem given in Equation 4.1 if we set $p_1 = 0$ and $p_2 = 1$ in the GCTF framework. Therefore, in this setting, the dispersion parameters ϕ_1 and ϕ_2 will play the role of the sparsity parameter. Furthermore, we can also change p_1 and p_2 in order to obtain different loss and regularization functions.

Here, we present a simple experiment where we would like to decompose the

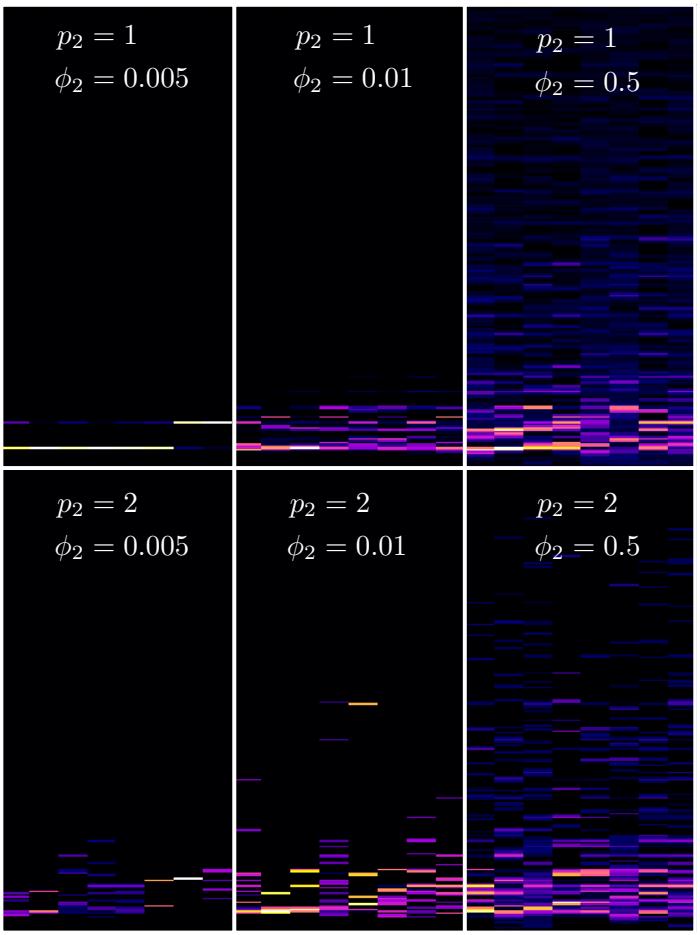


Figure 4.1. Dictionary matrices obtained with different dispersion and power parameters.

magnitude spectrogram (X_1) of a short audio signal by using the model that is defined above. We fix $p_1 = 0$ and $\phi_1 = 1$, and run the MUR algorithm in order to estimate Z_1 and Z_2 where we try several values for p_2 and ϕ_2. Figure 4.1 shows the dictionary matrices that are estimated for different p_2 and ϕ_2 values. It can be clearly seen that, the choice of the dispersion and power parameters drastically changes the structure of the estimated dictionaries. We can also observe that, for smaller ϕ_2 we obtain sparser dictionaries as expected, whereas the choice of p_2 becomes unimportant when ϕ_2 is not set to a reasonable value (0.5).

In practical applications, manual selection of these parameters is not trivial and can be time consuming. One can consider cross validation for selecting these variables in

supervised problem settings, however, this would also be problematic when N_x is large. On the other hand, we cannot use cross validation in unsupervised problem settings. In this chapter, we will focus on developing methods for estimating the dispersion and power parameters for the GCTF framework.

Even though coupled factorization models have been widely studied in several domains, estimation of the relative weights and divergences (here, corresponding to ϕ and p) has not been extensively explored. In [40], a maximum likelihood estimator for ϕ was presented under Gaussian observation models. In [79] a score matching approach was proposed for estimating the power parameter p only for the non-negative matrix factorization model that uses a Tweedie distribution with unitary dispersion ($\phi = 1$) as the observation model. Recently, a score matching approach was proposed in [80] for making inference in Tweedie distributions with $p \geq 0$, where the Tweedie distribution was approximated by an 'exponential divergence' distribution. The authors estimated the dispersion and power parameter by running a grid search procedure on this approximate distribution. Besides the $\beta-$divergence, the authors also enabled automatic selection of α, γ, and Rényi divergences through non-linear transformations. The need for a general and efficient method for coupled factorization models that would handle the whole Tweedie family still prevails.

In this chapter, we will develop two methods for jointly estimating the mixed divergences along with the latent factors in Tweedie models. Firstly, we will focus on a particular Tweedie model, called as the Tweedie compound Poisson distributions that is suitable for sparse observations. Then, we will develop an alternative method that can be applied to the whole Tweedie family.

4.1. Learning the β-Divergence in Tweedie Compound Poisson Models

In this section, we will focus on a particular observation model, called as the compound Poisson distribution, that coincides with the Tweedie model with $1 < p < 2$.[2] Here, we give a compact characterization of the compound Poisson distribution as a

[2]This section is based on the material published in [81].

Tweedie model [65]. The generative model is the compound Poisson distribution is as follows. A random variable x that is the sum of n independent and identically distributed Gamma random variables is compound Poisson distributed, when n is Poisson distributed [65]; formally:

$$x = \sum_{i=1}^{n} g_i \tag{4.4}$$

where n and g_i are

$$n \sim \mathcal{PO}(n; \lambda) \qquad g_i \sim_{\text{iid}} \mathcal{G}(g_i; a, b) \tag{4.5}$$

Here, $\mathcal{PO}$ and $\mathcal{G}$ denote the Poisson and Gamma densities, respectively. The marginal density $\mathbb{P}(x)$ is compound Poisson. More compactly, we can also write $x|n \sim \mathcal{G}(x; an, b)$.

To show the equivalence to the Tweedie, we first note that the cumulant generating function (CGF) $K_u(s)$ of a random variable u with density $\mathbb{P}(u)$ is defined as $K_u(s) = \log G_u(e^s)$ where $G_u(z) = \langle z^u \rangle_{\mathbb{P}(u)}$ is a generating function. From basic probability theory, we know that the generating function of the sum of a random number of iid variables is obtained by nesting as $G_x(z) = G_n(G_g(z))$, where

$$G_n(z) = \exp(\lambda(z-1)) \qquad\qquad G_g(z) = (1 - \log(z)/b)^{-a} \tag{4.6}$$

are generating functions for the Poisson and Gamma densities. By substitution we obtain the CGF of x as

$$K_x(s) = \lambda((1 - s/b)^{-a} - 1). \tag{4.7}$$

Now, we will show that we obtain the same CGF starting from the power variance assumption. We can easily verify that CGF for the exponential dispersion model

(EDM) (see Appendix B, Equations B.1 and B.2) is given by [65, 82]

$$K_x(s;\theta,\phi) = \frac{1}{\phi}\left(\kappa(s\phi+\theta) - \kappa(\theta)\right). \tag{4.8}$$

If we substitute the expression for $\kappa(\theta)$ in (B.4) and then express the result as a function of the expectation parameter $\hat{x}$ by noting that

$$\theta = \frac{\hat{x}^{1-p}}{1-p} \tag{4.9}$$

(as $d\theta/d\hat{x} = v(\hat{x})^{-1} = \hat{x}^{-p}$), we obtain

$$K_x(s;\theta,\phi) = \frac{\hat{x}^{2-p}}{(2-p)\phi}\left(\left(1 - s\phi(p-1)\hat{x}^{p-1}\right)^{\frac{2-p}{1-p}} - 1\right) \tag{4.10}$$

that has the same form as (4.7). By matching term by term, we see that the Tweedie distribution for $1 < p < 2$ is the compound Poisson distribution with the following parameter mapping:

$$\lambda = \frac{\hat{x}^{2-p}}{\phi(2-p)}, \quad a = \frac{2-p}{p-1}, \quad b = \frac{\hat{x}^{1-p}}{\phi(p-1)}. \tag{4.11}$$

By using this mapping, the joint distribution can be written as follows:

$$\begin{aligned}
\mathbb{P}(x,n|\hat{x},\phi,p) =& \mathbb{P}(x|n,\hat{x},\phi,p)\mathbb{P}(n|\hat{x},\phi,p) \\
=& \left[\exp(-\frac{\hat{x}^{2-p}}{(2-p)\phi})\right]^{[n=0]} \Big[\exp(-\frac{n}{p-1}\log(\phi) + n\frac{2-p}{p-1}\log\frac{x}{p-1} \\
& - n\log(2-p) - \log\Gamma(n+1) - \log\Gamma(\frac{2-p}{p-1}n) - \log(x) \\
& - \frac{1}{\phi}\left(\frac{\hat{x}^{1-p}x}{(p-1)} + \frac{\hat{x}^{2-p}}{(2-p)}\right))\Big]^{[n>0]}.
\end{aligned} \tag{4.12}$$

It turns out that $\sum_n \mathbb{P}(x,n|\cdot)$ does not have a closed form. Here, Dunn and Smyth provide numerical methods for approximate computation [82], but we propose here two simpler algorithms. An example pdf of a compound Poisson distribution is given in Figure 4.2.

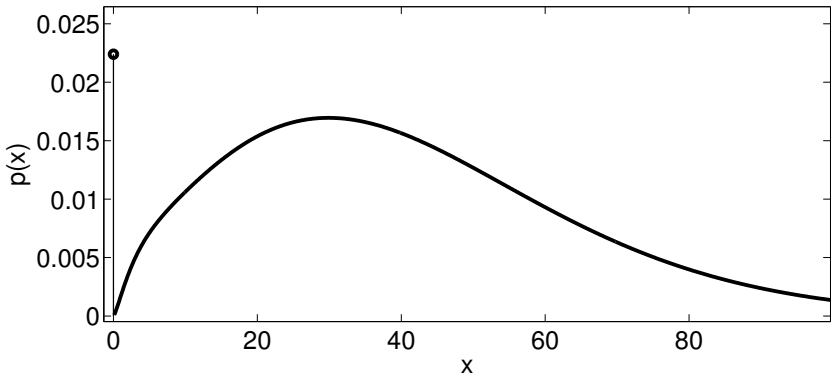


Figure 4.2. The compound Poisson distribution with $p = 1.3$, $\phi = 5$, and $\hat{x} = 40$. Note that the probability mass at zero makes this distribution suitable for sparse positive data.

Since p_ν and ϕ_ν are conditionally independent from $\hat{x}$ and therefore latent the factors, we can make use of the methods described in Chapter 3 in order to estimate the latent factors. Therefore, throughout this chapter we focus on estimation of the dispersion and the power parameters and we stick to a vector notation where we define $x_\nu \equiv \mathbf{vec}(X_\nu)$, $\hat{x}_\nu \equiv \mathbf{vec}(\hat{X}_\nu)$, $m_\nu \equiv \mathbf{vec}(M_\nu)$.

In the next subsections, we present three inference methods for estimating the dispersion and power parameters in Tweedie compound Poisson models. In the first and the second methods we follow a variational approach, where in the third method we integrate out the dispersion parameter and make inference on the marginal distribution.

4.1.1. Variational Approach

In this section, we present two variational methods, namely the Iterative Conditional Modes (ICM) and the Expectation-Maximization (EM) algorithms.

The ICM algorithm iteratively maximizes over the parameters n, ϕ, and p given x and $\hat{x}$. Even though the maximization over n is intractable, we can find the mode n^* by approximating the $\log\Gamma(\cdot)$ functions in (4.12) by using Stirling's approximation,

as proposed in [82]. The mode has the following analytical form:

$$n_\nu^*(i) = \frac{x_\nu(i)^{2-p_\nu}}{(2-p_\nu)\phi_\nu}. \tag{4.13}$$

Maximizing the dispersion parameter ϕ_ν is straightforward, however, since the power parameter p_ν and ϕ_ν are closely related to the variance and may affect each other, it can be necessary to regularize ϕ_ν in order to have a better estimate of p_ν. It is easy to verify that the conjugate prior of the dispersion parameter is the inverse Gamma distribution. Therefore, as we mentioned in Chapter 2, we assume an inverse Gamma prior on ϕ_ν. The optimal dispersion, given the other parameters is as follows:

$$\phi_\nu^* = \frac{\left(\sum_i \frac{m_\nu(i)\hat{x}_\nu(i)^{1-p_\nu}x_\nu(i)}{(p_\nu-1)} + \frac{m_\nu(i)\hat{x}_\nu(i)^{2-p_\nu}}{(2-p_\nu)}\right) + \kappa_\phi}{\frac{\sum_i m_\nu(i)n_\nu^*(i)}{p_\nu-1} + \tau_\phi + 1}. \tag{4.14}$$

Surprisingly, none of the references we are aware of used this conjugate prior. In the next section we will use this property to analytically integrate out the dispersion parameter.

The last step of the ICM algorithm is to compute the maximization over p_ν. Since the optimal p_ν does not have an analytical solution, we consult numerical methods. As the domain of p_ν is limited to $(1, 2)$, we run a simple grid search procedure in order to estimate the power parameter p_ν.

To sum up, at each iteration of the estimation algorithm, we first estimate the factors and compute the mean parameter $\hat{x}_\nu$. Then, we compute the parameters n_ν^* and ϕ_ν^* that are described above, and finally we compute the optimal power parameter p_ν. This procedure is run until convergence.

The EM algorithm is quite similar to the ICM algorithm in algorithmic sense, where we merely replace n_ν^* with the expectation $\langle n_\nu \rangle$ in (4.14). Unfortunately, computing this expectation is also intractable. Therefore, we use a numerical method that is similar to the one proposed in [82]. By using the fact that the conditional distribu-

tion of n_ν is unimodal, we approximate the expectation by numerically computing it around the mode which is defined in (4.13). The rest of the EM algorithm is the same as the ICM algorithm.

4.1.2. Integrating out the Dispersion Parameter

The dispersion parameter plays a key role when we deal with a coupled factorization model. However, when we have only one observed tensor, the dispersion parameter does not contribute to the estimation of the factors in a factorization model as it cancels out in the estimation algorithms.

In this section, we consider the case when $N_x = 1$, where we integrate out the dispersion parameter ϕ and n and make inference on the marginal distribution. When assumed an inverse Gamma prior on ϕ, we obtain the following marginal distribution:

$$\begin{aligned}\mathbb{P}(x,n) = &\left[\exp\Big(\tau_\phi(\log\kappa_\phi - \log(\frac{\hat{x}^{2-p}}{2-p} + \kappa_\phi)\Big)\right]^{[n=0]} \\ &\left[\exp\Big(n\frac{2-p}{p-1}\log\frac{x}{p-1} - n\log(2-p) - \log\Gamma(n+1) - \log\Gamma(\frac{2-p}{p-1}n)\right. \\ &- \log(x) - (\tau_\phi + \frac{n}{p-1})\log(\kappa_\phi + \frac{\hat{x}^{1-p}x}{(p-1)} + \frac{\hat{x}^{2-p}}{(2-p)}) + \tau_\phi\log\kappa_\phi \\ &\left.+ \log\Gamma(\tau_\phi + \frac{n}{p-1}) - \log\Gamma(\tau_\phi)\Big)\right]^{[n>0]}. \end{aligned} \tag{4.15}$$

In order to estimate the power parameter p, we also marginalize out n by using numerical methods. Finally, the optimal p is found by a grid search algorithm, similar to ICM and EM.

4.2. Learning Mixed β-Divergences in the Full Tweedie Model

In this section we present an algorithmic framework for MAP estimation of all the variables without restricting the method to a certain observation model.[3] Our aim

[3]This section is based on the material published in [83] and [84].

is to solve the following optimization problem:

$$\min \sum_{\nu=1}^{N_x} \left[\sum_{i=1}^{s_\nu} \Big(\log K(x_\nu(i), \phi_\nu, p_\nu) + \frac{1}{\phi_\nu} d_p(x_\nu(i) || \hat{x}_\nu(i)) \Big) - \log \mathbb{P}(\phi_\nu) \right] \tag{4.16}$$

with respect to $Z_{1:N_z}$, $\phi_{1:N_x}$, and $p_{1:N_x}$. Here, $s_\nu = \prod_k s_k^{[k \in I_{0,\nu}]}$ is the number of elements in x_ν and $\mathbb{P}(\phi_\nu)$ is the prior distribution of the dispersions.

In order to estimate the latent factors, dispersions, and power parameters jointly, we present an iterative schema where we divide the optimization problem of Equation 4.16 into simpler subproblems. The ultimate method is an ICM algorithm, where each parameter is updated at each iteration given the up-to-date values of the remaining parameters. Here, each iteration t consists of three estimation steps, stated as follows:

$$Z_\alpha^{(t+1)} = \arg\max_{Z_\alpha} \sum\nolimits_{\nu=1}^{N_x} \log \mathbb{P}(x_\nu | Z_{1:N_\alpha}, \phi_\nu^{(t)}, p_\nu^{(t)}), \quad \forall \alpha \tag{4.17}$$

$$\phi_\nu^{(t+1)} = \arg\max_{\phi_\nu} \log \big(\mathbb{P}(x_\nu | \phi_\nu, \hat{x}_\nu^{(t+1)}, p^{(t)}) \mathbb{P}(\phi_\nu) \big), \quad \forall \nu \tag{4.18}$$

$$p_\nu^{(t+1)} = \arg\max_{p_\nu} \log \mathbb{P}(x_\nu | \hat{x}_\nu^{(t+1)}, \phi_\nu^{(t+1)}, p_\nu), \quad \forall \nu \tag{4.19}$$

Given the dispersions and the power parameters, the first problem (Equation 4.17) reduces to the well-known problem of minimizing the β-divergence between the observations X_ν and the model outputs $\hat{X}_\nu$ with respect to $Z_{1:N_z}$. Therefore, we can make use of any of the algorithms presented in Chapter 3.

We first focus on learning the dispersion parameters ϕ_ν for integer values of $p_\nu = \{0, 1, 2, 3\}$. We estimate the optimal dispersion for these distributions by setting the derivative of the log-likelihood to zero and then solving it for ϕ_ν. Note that, since the normalizing constants of the Poisson and the Gamma distributions are complicated (see Appendix B), we replace the terms involving $\log \Gamma(\cdot)$ functions with Stirling's

approximation:

$$\log \Gamma(n) \approx -\frac{1}{2}\log n + n\log n - n. \tag{4.20}$$

Finally, we obtain the optimal dispersion parameters in closed form as follows:

$$\phi_\nu^\star = \frac{\left(\sum_{i=1}^{N} d_{p_\nu}(x_\nu(i)||\hat{x}_\nu(i))\right) + \kappa_\phi}{s_\nu/2 + \tau_\phi + 1}, \quad p \in \{0,1,2,3\} \tag{4.21}$$

where $d_p(\cdot)$ is the β-divergence, defined in Equation 2.9.

Secondly, we focus on the remaining cases of p, where the probability density functions cannot be written in closed-form analytical expressions. However, they can be expressed as infinite series that is defined as follows: [65]

$$\mathcal{TW}_p(x;\hat{x},\phi) = \frac{1}{x\xi_p}\left(\sum_{k=1}^{\infty} V_k(x,p,\phi)\right)\exp\left\{\frac{1}{\phi}\left(\frac{\hat{x}^{1-p}x}{1-p} - \frac{\hat{x}^{2-p}}{2-p}\right)\right\} \tag{4.22}$$

and $\xi_p = 1$ for $p \in (1,2)$ and $\xi_p = \pi$ otherwise. The terms $V_k(\cdot)$ are given in Appendix B.

In order to estimate the dispersions in the compound Poisson and the Tweedie stable distributions, we use a limited memory, bounded quasi-Newton method, namely the L-BFGS-B algorithm [78]. This method requires the gradient of the map objective function that is given as follows:

$$\begin{aligned}\frac{\partial g(\phi_\nu)}{\partial \phi_\nu} =& \frac{1}{\phi_\nu^2}\Big[\sum_{i=1}^{s_\nu}\Big(\frac{\hat{x}_\nu(i)^{2-p_\nu}}{2-p_\nu} + \frac{x_\nu(i)\hat{x}_\nu(i)^{1-p_\nu}}{p_\nu - 1}\Big) + \kappa_\phi\Big] \\ &- \frac{1}{\phi_\nu}\Big[c_p \frac{\sum_{i=1}^{s_\nu}\sum_{k=1}^{\infty} kV_k(x_\nu(i),p_\nu,\phi_\nu)}{\sum_{i=1}^{s_\nu}\sum_{k=1}^{\infty} V_k(x_\nu(i),p_\nu,\phi_\nu)} + \tau_\phi + 1\Big]\end{aligned} \tag{4.23}$$

where $g(\phi_\nu) = -\log \mathbb{P}(x_\nu,\phi_\nu|\hat{x}_\nu,p_\nu)$ and $c_p = 1-p_\nu$ for $p_\nu < 0$ and $c_p = 1/(p_\nu - 1)$ otherwise.

Input: $X_{1:N_x}$, R, $Z^{(0)}_{1:N_z}$, $\phi^{(0)}_{1:N_x}$, $p^{(0)}_{1:N_x}$

repeat

  // Run MUR, DIGD, or HAMSI until convergence

  $Z^{(t)}_{\alpha} = \arg\max_{Z_\alpha} \sum_{\nu}^{N_x} \log \mathbb{P}(X_\nu | Z^{(t-1)}_{1:N_\alpha}, \phi^{(t-1)}_{\nu}, p^{(t-1)}_{\nu}) \qquad \forall \alpha = 1, \ldots, N_z$

  // If $p_\nu \in \{0, 1, 2, 3\}$, use Equation 4.21, otherwise use Equation 4.23

  $\phi^{(t)}_{\nu} = \arg\max_{\phi} \log\big(\mathbb{P}(X_\nu | \phi, \hat{X}^{(t)}_{\nu}, p^{(t-1)}) \mathbb{P}(\phi)\big) \qquad \forall \nu = 1, \ldots, N_x$

  // Run grid search for the power parameters

  $p^{(t)}_{\nu} = \arg\max_{p} \log \mathbb{P}(X_\nu | \hat{X}^{(t)}_{\nu}, \phi^{(t)}_{\nu}, p) \qquad \forall \nu = 1, \ldots, N_x$

  $t \leftarrow t + 1$

until *convergence or* $t >$ *max_epochs*

Figure 4.3. Joint inference in coupled factorization models.

The gradient requires two infinite summations to be computed, which is intractable. Here, we utilize efficient numerical methods by following [82] for approximate computation of these summations. This method locates the indices k where the terms V_k make the major contribution to the sum. The infinite sum is then approximated by summing up the terms in the located region. The cases $p \in (1, 2)$ and $p > 2$ is described in [82]; for completeness we explain the method for $p < 0$ in Appendix D.

The last step of the proposed method (Equation 4.19) is to compute the maximum likelihood estimate of the power parameter p. Unfortunately, the optimal p does not have an analytical solution. Similar to Section 4.1, we utilize a grid search procedure in order to estimate the power parameter p given the other parameters. Note that, the proposed methods are scalable; in large-scale and/or distributed settings, they can be implemented in an embarrassingly parallel fashion as the problem is separable over the indices i. The method is illustrated in Figure 4.2.

5. FULL BAYESIAN INFERENCE VIA MARKOV CHAIN MONTE CARLO

Maximum likelihood and a-posteriori estimation methods provide us useful and practical tools that can be used in various applications. However, in certain cases, they are prone to over-fitting since they fall short at capturing uncertainties that arise in the inference process. Instead of aiming to obtain single point estimates of the latent variables, full Bayesian inference aims to characterize the full posterior distribution over the latent variables. Apart from being able to handle the uncertainties and therefore being robust to over-fitting, full Bayesian inference has many advantages over the point estimation methods in various tasks such as the model selection problem. In this chapter, we will develop two MCMC methods that are based on MH for sampling from the posterior distribution $\mathbb{P}(Z_{1:N_z}|X_{1:N_x})$, namely the Gibbs sampler and parallel stochastic gradient Langevin dynamics.

5.1. Gibbs Sampler

One particularly convenient and simple MH strategy is the Gibbs sampler where one samples each block of variables from the so called full conditional distributions.[4] Deriving a single Gibbs sampler for all the cases of the power parameter p is not straightforward, therefore one needs to focus on individual cases of the Tweedie family. For the Poisson model ($p = 1$), we define the following augmented generative model:

$$\begin{aligned} Z_\alpha(v_\alpha) &\sim \mathcal{G}(Z_\alpha(v_\alpha); A_\alpha(v_\alpha), B_\alpha(v_\alpha)) && \text{factor priors} \\ \Lambda_\nu(v) &= \prod_\alpha Z_\alpha(v_\alpha)^{R_{\nu,\alpha}} && \text{intensities} \\ S_\nu(v)|Z_{1:N_z} &\sim \mathcal{PO}(S_\nu(v); \Lambda(v)) && \text{sources} \\ X_\nu(u_\nu) &= \sum_{\bar{u}_\nu} S_\nu(v) && \text{outputs} \end{aligned} \tag{5.1}$$

[4] This section is based on the material published in [85].

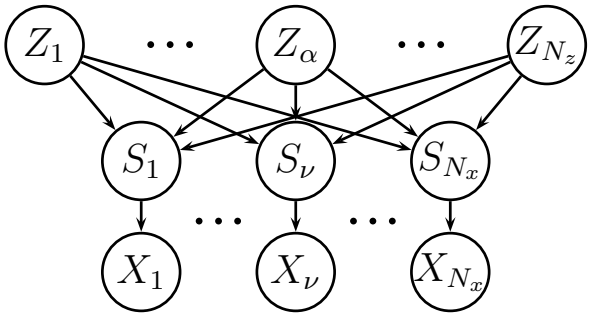


Figure 5.1. Graphical representation of the GCTF framework. The nodes represent the random variables and the arrows represent the conditional independence structure.

In the model defined in Equation 2.6, the observed tensors X_ν directly depend on the latent factors Z_α. Here, we form a so called composite model where we augment the model in Equation 2.6 by defining the intensity tensors Λ_ν and the source tensors S_ν as intermediate layers, where in this model, the observed tensors are deterministic functions of the sources. Figure 5.1 illustrates this model. Note that, the Tweedie models with $p = 0$ and $p = 2$, can also be represented as composite models [86]; however, the other cases have not been explored in the literature.

The Gibbs sampler for the GCTF model with Poisson observations can be formed by iteratively drawing samples from the full conditional distributions as follows:

$$S_\nu^{(t+1)} \sim \mathbb{P}(S|Z_{1:N_z}^{(t)}, X_\nu, \Theta) \qquad \nu = 1 \dots N_x \tag{5.2}$$

$$Z_\alpha^{(t+1)} \sim \mathbb{P}(Z_\alpha|S_{1:N_x}^{(t)}, Z'_{\neg\alpha}, X_{1:N_x}, \Theta) \qquad \alpha = 1 \dots N_z \tag{5.3}$$

where $Z'_{\neg\alpha}$ denotes the most recent values of all the factors but Z_α, Θ denotes the prior distribution parameters $\{A_\alpha, B_\alpha\}_{\alpha=1}^{N_z}$, and the full conditionals are defined as:

$$\mathbb{P}(S_\nu|Z_{1:N_z}, X_\nu, \Theta) = \prod_{u_\nu} \mathcal{M}\left(S_\nu(u_\nu, \bar{U}_\nu); \frac{\Lambda_\nu(u_\nu, \bar{U}_\nu)}{\hat{X}_\nu(u_\nu)}\right) \tag{5.4}$$

Input: $X_{1:N_x}$, R

Initialize factors: $Z_\alpha^{(0)} \sim \mathcal{G}(Z_\alpha; A_\alpha, B_\alpha)\ \forall \alpha = 1, \ldots, N_z$

repeat

// Compute the intensity and parameter tensors: $\forall \nu = 1, \ldots, N_x,\ v \in \mathcal{C}_I(I)$

$\Lambda_\nu(v) = \prod_\alpha Z_\alpha(v_\alpha)^{R_{\nu,\alpha}}$

$\hat{X}_\nu(u_\nu) = \sum_{\bar{u}_\nu} \Lambda_\nu(v)$

// Sample Sources: $\forall \nu = 1, \ldots, N_x,\ \forall u_\nu \in \mathcal{C}_I(I_{0,\nu})$

$S_\nu(u_\nu, \bar{U}_\nu)^{(t)} \sim \mathcal{M}\left(S_\nu(u_\nu, \bar{U}_\nu); \frac{\Lambda_\nu(u_\nu, \bar{U}_\nu)}{\hat{X}_\nu(u_\nu)}\right)$

// Sample Factors: $\forall \alpha = 1 \ldots N_z,\ v_\alpha \in \mathcal{C}_I(I_\alpha)$

$Z_\alpha^{(t)}(v_\alpha) \sim \mathcal{G}\left(Z_\alpha(v_\alpha); \Sigma_\alpha(v_\alpha), \Phi_\alpha(v_\alpha)\right)$

until $t >$ *max_epochs*

Figure 5.2. Block Gibbs Sampler.

$$\mathbb{P}(Z_\alpha | S_{1:N_x}, Z_{\neg\alpha}, X_{1:N_x}, \Theta) = \prod_{v_\alpha} \mathcal{G}\left(Z_\alpha(v_\alpha); \Sigma_\alpha(v_\alpha), \Phi_\alpha(v_\alpha)\right) \tag{5.5}$$

where

$$\Sigma_\alpha(v_\alpha) = A_\alpha(v_\alpha) + \left[\sum_\nu R_{\nu,\alpha} \left(\sum_{\bar{v}_\alpha} S_\nu(v)\right)\right] \tag{5.6}$$

$$\Phi_\alpha(v_\alpha) = B_\alpha(v_\alpha) + \left[\sum_\nu \left(\sum_{\bar{v}_\alpha} \prod_{\alpha' \neq \alpha} Z_{\alpha'}(v_{\alpha'})^{R_{\nu,\alpha'}}\right)\right] \tag{5.7}$$

Here, $\mathcal{M}$ denotes the multinomial distribution. Verbally, given a particular instance of observed indices u_ν, the full conditional of S_ν is a multinomial distribution over all the latent index configurations $\bar{U}_\nu \equiv \mathcal{C}_I(\bar{I}_{0,\nu})$. The pseudo-code is given in Figure 5.1. Note that, the algorithm presented in [53] is a special case of the presented sampling algorithm.

5.1.1. Partially Collapsed Gibbs Sampling via Space Alternating Data Augmentation

Sampling the latent sources in a 'block' fashion as in Equation 5.4, i.e., jointly sampling $S_\nu(u_\nu, \bar{U}_\nu)$, can be problematic in practice since it could yield computational inefficiencies and requires sampling from degenerate distributions when $p = 2$. To address these problems, we develop a *partially collapsed* Gibbs sampler by using space alternating data augmentation (SADA).

SADA was first presented in [87] for making inference in Gaussian mixture models. In [88], Fevotte et al. presented an MCMC procedure with SADA for making inference in composite models including NMF. In this section we will generalize this procedure for tensor factorization models. However, deriving a SADA sampler for coupled factorizations is not straightforward, therefore in this section we focus on the case where a single tensor is observed ($N_x = 1$).

The main idea behind SADA is sampling each slice of the sources from their marginal distribution instead of sampling all the slices from their full conditional at the same time. This approach significantly reduces the memory requirements of a sampler since it only requires storing $s_{0,1}$ elements instead of $\left[\left(\prod_{\alpha=1}^{N_z} s_\alpha\right)\left(\prod_{\nu=1}^{N_x} s_{0,\nu}\right)\right]$ elements of the latent components at each iteration of the sampling procedure.

Directly applying the SADA algorithm to tensor factorizations is not straightforward since the index structure for different models can lead to different conditional independence structures. Therefore, we rewrite the original model (2.2) as a collection of several 'marginal' models, one for each latent factor Z_α as follows:

$$\hat{X}_1(u_1) = \sum_{\bar{u}_1 \cap v_\alpha} \underbrace{Z_\alpha(v_\alpha) \sum_{\bar{u}_1 \cap \bar{v}_\alpha} \prod_{\alpha' \neq \alpha} Z_{\alpha'}(v_{\alpha'})}_{\equiv \Lambda_{1,\alpha}(\lambda_\alpha)} \tag{5.8}$$

where $\lambda_\alpha = v \setminus (\bar{u}_1 \cap \bar{v}_\alpha)$. We also define $S_{1,\alpha}(\lambda_\alpha)$ by using the additivity property of

Input: X_1 $(N_x = 1)$

Initialize factors: $Z_\alpha^{(0)} \sim \mathcal{G}(Z_\alpha; A_\alpha, B_\alpha)\ \forall \alpha = 1, \ldots, N_z$

repeat

  for $\alpha = 1 \ldots N_z,\ v_\alpha \in \mathcal{C}_I(I_\alpha)$ **do**

    $\hat{X}_1(u_1) = \sum_{\bar{u}_\nu} \prod_\alpha Z_\alpha(v_\alpha)$

    // Sample Slices of Sources: $u_1 \in \mathcal{C}_I(I_{0,1})$

    $\Lambda_{1,\alpha}(v_\alpha \cup u_1) = \sum_{\bar{v}_\alpha \cap \bar{u}_1} \prod_{\alpha'} Z'_{\alpha'}(v_{\alpha'})$

    $S_{1,\alpha}^{(i)}(v_\alpha \cup u_1) \sim \mathcal{BI}\left(S_{1,\alpha}; X_1(u_1), \frac{\Lambda_{1,\alpha}(v_\alpha \cup u_1)}{\hat{X}(u_1)}\right)$

    // Sample Factors:

    $Z_\alpha^{(t)}(v_\alpha) \sim \mathcal{G}\left(Z_\alpha(v_\alpha); \Sigma_\alpha(v_\alpha), \Phi_\alpha(v_\alpha)\right)$

until $t >$ *max_epochs*

Figure 5.3. SADA Sampler.

Poisson distribution:

$$S_{1,\alpha}(\lambda_\alpha) \sim \mathcal{PO}(S_{1,\alpha}(\lambda_\alpha); \Lambda_{1,\alpha}(\lambda_\alpha)) \tag{5.9}$$

$$X_1(u_1) = \sum_{\bar{u}_1 \cap v_\alpha} S_{1,\alpha}(\lambda_\alpha). \tag{5.10}$$

In the SADA algorithm, each slice of $S_{1,\alpha}$ is drawn from its marginal distribution and then each Z_α is drawn by conditioning on $S_{1,\alpha}$. Curious reader is referred to [88] for a detailed description and the proof of convergence for the matrix case. The pseudocode is given in Figure 5.1.1. Note that the $\mathcal{BI}$ symbol in the pseudocode refers to the Binomial distribution.

5.1.2. Marginal Likelihood Estimation with Chib's Method

The marginal likelihood of the observed data under a tensor factorization model $\mathbb{P}(X_{1:N_x})$ is often necessary for certain problems such as model selection. This quantity can be estimated from the Gibbs output and it is known as the Chib's method [89].

This method is applied in [53] for NMF; here we generalize it in order to estimate the marginal likelihood for the GCTF framework with $N_x = 1$.

Suppose the Gibbs sampler has been run until convergence and we have N samples for each variable. The marginal likelihood is defined as:

$$\mathbb{P}(X_1) = \frac{\mathbb{P}(S_1, Z_{1:N_z}, X_1)}{\mathbb{P}(S_1, Z_{1:N_z}|X_1)}. \tag{5.11}$$

This equation holds for all points $(S_1, Z_{1:N_z})$. Provided that the distribution is unimodal, a good candidate point in the configuration space is a configuration near the mode $(\tilde{S}_1, \tilde{Z}_{1:N_z})$. The numerator (the full joint distribution) is straightforward to evaluate. We can expand the denominator as follows:

$$\begin{aligned}\mathbb{P}(\tilde{S}_1, \tilde{Z}_{1:N_z}|X_1) =& \mathbb{P}(\tilde{Z}_1|\tilde{Z}_{2:N_z}, \tilde{S}_1)\mathbb{P}(\tilde{Z}_2|\tilde{Z}_{3:N_z}, \tilde{S}_1)\dots\mathbb{P}(\tilde{Z}_{N_z-1}|\tilde{Z}_{N_z}, \tilde{S}_1)\mathbb{P}(\tilde{Z}_{N_z}|\tilde{S}_1)\mathbb{P}(\tilde{S}_1|X_1) \\ =& \mathbb{P}(\tilde{S}_1|X_1)\mathbb{P}(\tilde{Z}_1|\tilde{Z}_{2:N_z}, \tilde{S}_1)\prod_{\alpha=2}^{N_z}\mathbb{P}(\tilde{Z}_\alpha|\tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1),\end{aligned}$$

where $\mathbb{P}(\tilde{Z}_{N_z}|\tilde{Z}_{N_z+1}, \tilde{S}_1) = \mathbb{P}(\tilde{Z}_{N_z}|\tilde{S}_1)$. The ordering of the variables at this expansion step can be changed, however without loss of generality we assume that the ordering is $Z_1 \dots Z_{N_z}$.

The term $\mathbb{P}(\tilde{Z}_1|\tilde{Z}_{2:N_z}, \tilde{S}_1)$ is full conditional, so it is available for the Gibbs sampler. We can also approximate $\mathbb{P}(\tilde{S}_1|X_1)$ as:

$$\mathbb{P}(\tilde{S}_1|X_1) = \int dZ_{1:N_z}\, \mathbb{P}(\tilde{S}_1|Z_{1:N_z}, X_1)\mathbb{P}(Z_{1:N_z}|X_1) \tag{5.12}$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\mathbb{P}(\tilde{S}_1|Z^{(t)}_{1:N_z}, X_1). \tag{5.13}$$

Evaluating the term $\mathbb{P}(\tilde{Z}_\alpha|\tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1)$ is more complicated. Firstly, we start by rewrit-

ing the term $\mathbb{P}(\tilde{Z}_{N_z}|\tilde{S}_1)$ as:

$$\mathbb{P}(\tilde{Z}_{N_z}|\tilde{S}_1) = \int dZ_{1:N_z-1}\, \mathbb{P}(\tilde{Z}_{N_z}|Z_{1:N_z-1}, \tilde{S}_1)\mathbb{P}(Z_{1:N_z-1}|\tilde{S}_1) \tag{5.14}$$

The first term here is again full conditional. However, we do not have samples from the distribution $\mathbb{P}(Z_{1:N_z-1}|\tilde{S}_1)$ since the sampler gives us samples from $\mathbb{P}(Z_{1:N_z-1}|X_1)$. The solution is approximating this term by running the Gibbs sampler M more iterations and clamping S_1 at $\tilde{S}_1$: $(Z_{1:N_z}^{(N+m)}) \sim \mathbb{P}(Z_{1:N_z}|S_1 = \tilde{S}_1)$. The estimate is as follows:

$$\mathbb{P}(\tilde{Z}_{N_z}|\tilde{S}_1) \approx \frac{1}{M}\sum_{m=N+1}^{N+M} \mathbb{P}(\tilde{Z}_{N_z}|Z_{1:N_z-1}^{(m)}, \tilde{S}_1). \tag{5.15}$$

We can apply the same idea to the rest of the terms in (5.12) by clamping some of the factors and running the sampler M more iterations for each $\alpha = (N_z - 1), \ldots, 2$. The resulting estimation is as follows:

$$\begin{aligned}\mathbb{P}(\tilde{Z}_\alpha|\tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1) &= \int dZ_{1:\alpha-1}\, \mathbb{P}(\tilde{Z}_\alpha|Z_{1:\alpha-1}, \tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1)\mathbb{P}(Z_{1:\alpha-1}|\tilde{Z}_{\alpha+1:N_z}\tilde{S}_1) \\ &\approx \frac{1}{M}\sum_{m=l_\alpha}^{u_\alpha} \mathbb{P}(\tilde{Z}_\alpha|Z_{1:\alpha-1}^{(m)}, \tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1)\end{aligned} \tag{5.16}$$

where l_α and u_α denote the first and the last indices of the drawn samples while $\mathbb{P}(\tilde{Z}_\alpha|\tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1)$ is being estimated and they are defined as $l_\alpha = N + (N_z - \alpha)M + 1$ and $u_\alpha = N + (N_z - \alpha + 1)M$.

After replacing the terms in (5.12) with their estimates that are defined in (5.13) and (5.16), Chib's method estimates the marginal likelihood as follows:

$$\log \mathbb{P}(X_1) = \log \mathbb{P}(\tilde{S}_1, \tilde{Z}_{1:N_z}, X_1) - \log \mathbb{P}(\tilde{S}_1, \tilde{Z}_{1:N_z}|X_1) \tag{5.17}$$

$$\begin{aligned}&\approx \log \mathbb{P}(\tilde{S}_1, \tilde{Z}_{1:N_z}, X_1) - \log \mathbb{P}(\tilde{Z}_1|\tilde{Z}_{2:N_z}, \tilde{S}_1) - \log \sum_{i=1}^{N} \mathbb{P}(\tilde{S}_1|Z_{1:N_z}^{(t)}, X_1) \\ &\quad - \sum_{\alpha=2}^{N_z} \log \sum_{m=l_\alpha}^{u_\alpha} \mathbb{P}(\tilde{Z}_\alpha|Z_{1:\alpha-1}^{(m)}, \tilde{Z}_{\alpha+1:N_z}, \tilde{S}_1) + \log(K-1)MN.\end{aligned} \tag{5.18}$$

5.1.3. Gibbs Sampler for Alpha-Stable Matrix Factorization

In this section, we develop a Gibbs sampler for sampling from the posterior distributions of the latent variables that appear in the αMF model. Firstly, by following a similar idea that we described in Section 5.1, we make use of an equivalent formulation to Equation 2.14 by using augmentation, which leads to the following composite model:

$$\begin{aligned} S(f,n,k)|W,H,\alpha &\sim \mathcal{S}\alpha\mathcal{S}_c\Big(S(f,n,k);[W(f,k)H(k,n)]^{1/\alpha}\Big) \\ X(f,n) &= \sum_k S(f,n,k) \end{aligned} \tag{5.19}$$

where $S(:,:,k)$ are called the latent sources. Since we need to sample from the conditional distributions of the latent variables, we express αMF as conditionally Gaussian by making use of the product property of the stable distributions [68, 69], as follows:

$$\begin{aligned} \Phi(f,n,k)|\alpha &\sim \mathcal{S}\Big(\Phi(f,n,k);\frac{\alpha}{2},1,2(\cos\frac{\pi\alpha}{4})^{2/\alpha},0\Big) \\ S(f,n,k)|\Phi,W,H,\alpha &\sim \mathcal{N}_c\Big(S(f,n,k);0,\Phi(f,n,k)[W(f,k)H(k,n)]^{2/\alpha}\Big) \\ X(f,n) &= \sum_k S(f,n,k), \end{aligned} \tag{5.20}$$

where $\mathcal{N}_c$ denotes the complex isotropic Gaussian distribution and Φ is the *impulse* variable. This formulation will allow us to analytically derive the conditional distributions of S, W, and H. Besides, now we can clearly see the impulsive structure of the model, where the variances of the Gaussian observations are modulated by infinite variance stable random variables, whose impulsiveness is controlled by α. The graphical representation of αMF is given in Figure 5.4.

The full conditional distributions of W and H are given as follows:

$$\mathbb{P}(W(f,k)|\alpha,S,H,\Phi) = \mathcal{GG}(W(f,k);a'_w,b'_w,-2/\alpha) \tag{5.21}$$

$$\mathbb{P}(H(k,n)|\alpha,S,W,\Phi) = \mathcal{GG}(H(k,n);a'_h,b'_h,-2/\alpha) \tag{5.22}$$

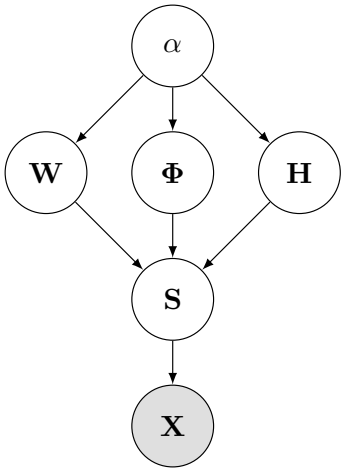


Figure 5.4. The graphical model αMF. The nodes represent the random variables, the arrows determine the conditional independence structure, and the shaded the nodes represent the observed variables.

where

$$a'_w = a_w + \frac{N}{2}, \quad b'_w = (b_w^{2/\alpha} + \sum_n \frac{|S(f,n,k)|^2}{2H(k,n)^{2/\alpha}\Phi(f,n,k)})^{\alpha/2}, \tag{5.23}$$

$$a'_h = a_h + \frac{F}{2}, \quad b'_h = (b_h^{2/\alpha} + \sum_n \frac{|S(f,n,k)|^2}{2W(f,k)^{2/\alpha}\Phi(f,n,k)})^{\alpha/2}. \tag{5.24}$$

To sample the latent sources, it is possible to develop a 'block' Gibbs sampler similar to the one that we developed in Section 5.1, where we would need to sample $S(f,n,:)$ jointly at each iteration. However, this approach requires sampling from a degenerate multivariate Gaussian and could yield computational inefficiencies in certain cases. Therefore, we utilize a SADA sampler (see Section 5.1.1) for sampling S where we draw samples from the marginal conditional distribution of $S(f,n,k)$ instead of the full conditional distribution, given as follows:

$$\mathbb{P}(S(f,n,k)|\alpha,W,H,\Phi,X) = \mathcal{N}_c(S(f,n,k);\mu'_s,\sigma'_s) \tag{5.25}$$

where

$$\mu'_s = g_{fnk} X(f,n), \tag{5.26}$$

$$\sigma'_s = (1 - g_{fnk})[W(f,k)H(k,n)]^{2/\alpha}\Phi(f,n,k), \tag{5.27}$$

$$g_{fnk} = \frac{[W(f,k)H(k,n)]^{2/\alpha}\Phi(f,n,k)}{\sum_{k'}[W(f,k')H(k',n)]^{2/\alpha}\Phi(f,n,k')}. \tag{5.28}$$

Unfortunately, the full conditional distributions of α and Φ cannot be derived analytically, therefore we resort to Metropolis Hastings (MH) algorithm for sampling from their full conditional distributions. The MH algorithm generates samples from $\pi(\alpha) = \mathbb{P}(\alpha|\cdot)$ in two steps. First, it generates a random sample α' from a *proposal* distribution $\alpha' \sim q(\alpha'|\alpha^{(t)})$, then computes an acceptance probability $a(\alpha^{(t)} \to \alpha')$ and draws a uniform random number $u \sim \mathcal{U}([0,1])$. If $u < a(\alpha^{(t)} \to \alpha')$, it accepts the sample and set $\alpha^{(t+1)} = \alpha'$; otherwise it rejects the sample and sets $\alpha^{(t+1)} = \alpha^{(t)}$. The acceptance probability is given as follows:

$$a(\alpha \to \alpha') = \min\{1, \frac{q(\alpha|\alpha')\pi(\alpha')}{q(\alpha'|\alpha)\pi(\alpha)}\}. \tag{5.29}$$

Here, we choose a symmetric proposal distribution $q(\alpha'|\alpha) = \mathcal{N}(\alpha'; \alpha, \sigma_\alpha^2)$, that would explore the state space of α by a random walk. The acceptance probability for α then becomes:

$$a(\alpha \to \alpha') = \min(1, \frac{\mathbb{P}(S, W, H, \Phi, \alpha')}{\mathbb{P}(S, W, H, \Phi, \alpha)}) \tag{5.30}$$

Evaluating this probability requires stable densities to be evaluated twice at each epoch. Therefore, we have developed a fast numerical method for evaluating stable densities by making use of their power series representation [69, 90]. The details of this method is given in Appendix D.

We follow a similar procedure for Φ, where we choose the prior distribution of $\Phi(f,n,k)$ as its proposal distribution: $q(\Phi(f,n,k)) = \mathbb{P}(\Phi(f,n,k))$. Accordingly, the

acceptance probability simplifies and we obtain the following expression:

$$a(\Phi(f,n,k) \rightarrow \Phi'(f,n,k)) = \min(1, \frac{\mathcal{N}_c(S(f,n,k); 0, [W(f,k)H(k,n)]^{2/\alpha}\Phi'(f,n,k))}{\mathcal{N}_c(S(f,n,k); 0, [W(f,k)H(k,n)]^{2/\alpha}\Phi(f,n,k))}). \tag{5.31}$$

5.2. Parallel and Distributed Stochastic Gradient Markov Chain Monte Carlo for Large-Scale Problems

Despite the well known advantages, Monte Carlo methods are typically not the method choice in large scale MF problems as they are perceived to be computationally very demanding. Indeed, the approaches that are presented in the previous section require passing over the whole data set at each iteration, which makes the methods impractical for large data sets. Recently, alternative approaches have been proposed to scale-up MCMC inference to large-scale regime. An important attempt was made by Welling and Teh [22], where the authors combined the ideas from a gradient-based MCMC method, so called the Langevin dynamics (LD) [91] and the popular optimization method, stochastic gradient descent (SGD) [47], and developed a scalable MCMC framework called as the stochastic gradient Langevin dynamics (SGLD). Unlike conventional batch MCMC methods, SGLD requires to 'see' only a small subset of the data per iteration similar to SGD. With this manner, SGLD can handle large datasets while at the same time being a valid MCMC method that forms a Markov Chain asymptotically sampling from the target density. Approximation analysis of SGLD has been studied in [92] and [93]. Several extensions of SGLD have been proposed. Ahn et al. [23] made use of the Fisher information besides the noisy gradients, Patterson and Teh [25] applied SGLD on the probability simplex. Chen et al. [24] and Ding et al. [94] considered second order Langevin dynamics and made use of the momentum terms, extending the vanilla SGLD.

In this section, we develop a parallel and distributed MCMC method for sampling from the full posterior of the latent factors. Our approach is carefully designed for factorization models and builds upon the generic distributed SGLD (DSGLD) frame-

work that was proposed in [26] where separate Markov chains are run in parallel on different subsets of the data that are distributed among worker nodes. When applied to factorization models, DSGLD results in computational inefficiencies since it cannot exploit the conditional independence structure of them. Besides, DSGLD requires all the latent variables (i.e., $Z_{1:N_z}$) to be synchronized once in a couple of iterations which introduces a significant amount of communication cost. On the other hand, for large problems it may not even be possible to store the latent variables in a single machine; one needs to distribute the latent variables among the nodes as well.

We develop a parallel and distributed variant of SGLD tailored for GCTF, that we call Parallel SGLD (PSGLD). PSGLD has very favorable scaling properties with increasing data size, remarkably up to the point that the resulting sampler is computationally not much more demanding than an optimization method such as DIGD (see Chapter 3). Reminiscent to DSGD, PSGLD achieves high performance by exploiting the conditional independence structure of factorization models for sub-sampling the data in a systematic manner as to allow parallelization. The main advantages of PSGLD can be summarized as follows:

(i) Due to its inherently parallel structure, PSGLD is faster than SGLD and Gibbs sampler by several orders of magnitude while being as accurate.
(ii) As we will illustrate in our experiments, PSGLD can easily be implemented in both shared-memory and distributed architectures. This makes the method suitable for very large data sets that might be distributed among many nodes.
(iii) Unlike DSGLD, which requires to communicate all the latent factors among the worker nodes, PSGLD communicates only small parts of the shared factors. This drastically reduces the communication cost for large problem sizes.
(iv) We show that, the probability distribution of the samples generated by PSGLD converges to the Bayesian posterior.

We note that, a DSGLD-based, distributed matrix factorization (MF) framework has been independently proposed by Ahn et al. [95], where the authors focus on a particular MF model, called as the Bayesian probabilistic matrix factorisation

(BPMF) [96]. In this study, we develop our method on GCTF, in which we obtain several observation models that have been used in important MF models (such as BPMF, Poisson-NMF [50], Itakura-Saito NMF [52]) as special cases. On the other hand, since [95] is based on DSGLD, the latent factors are not distributed and need to be synchronized throughout the iterations, which might cause inefficiency in memory and communication time.

5.2.1. Parallel Stochastic Gradient Langevin Dynamics for Factorization Models

In the last decade, SGD has become very popular due to its low computational requirements and convergence guarantee. SGLD brings the ideas of SGD and LD together in order to generate samples from the posterior distribution in a computationally efficient way. In algorithmic sense, SGLD is identical to SGD except that it injects a Gaussian noise at each iteration. For GCTF models, SGLD iteratively applies the following update rules in order to obtain the sample $Z_\alpha(v_\alpha)^{(t)}$:

$$Z_\alpha(v_\alpha)^{(t)} = Z_\alpha(v_\alpha)^{(t-1)} - \epsilon^{(t)}\Big(\frac{U}{|\Omega^{(t)}|}\sum_{u\in\Omega^{(t)}}\partial f_u(Z_{1:N_z})\Big) + \Psi_\alpha(v_\alpha)^{(t)}$$

Here, we make use of the notation presented in Equation 3.10, where the iterate u covers all (ν, u_ν) pairs and $\Omega^{(t)} \subset [U]$ is the subset that is drawn at iteration t. We can clearly see that, SGLD is very similar to the IGD algorithm given in Equation 3.11 in algorithmic sense, apart from the injected noise. The elements of the noise tensor $\Psi_\alpha^{(t)}$ are independently Gaussian distributed:

$$\Psi_\alpha(v_\alpha)^{(t)} \sim \mathcal{N}(\Psi_\alpha(v_\alpha)^{(t)}; 0, 2\epsilon^{(t)}). \tag{5.32}$$

For convergence, the step size $\epsilon^{(t)}$ must satisfy the conditions given in Equation 3.12.

In SGLD, the subset $\Omega^{(t)}$ can be drawn randomly with or without replacement. When dealing with tensor factorization models, instead of sub-sampling the data arbi-

Input: $X_{1:N_x}$, R, $p_{1:N_x}$, $\phi_{1:N_x}$, η, γ, $Z^{(0)}_{1:N_z}$
$t \leftarrow 1$
repeat
  $\epsilon_t = (\eta/t)^\gamma$
  for $s = 1, \ldots, S$ **do**
    for $b = 1, \ldots, B_s$ **do in parallel**
      // $\forall \alpha = 1, \ldots, N_z$, $v_\alpha \in \mathcal{B}_I(I_\alpha, b)$
      $Z_\alpha(v_\alpha)^{(t)} = Z_\alpha(v_\alpha)^{(t-1)} - \epsilon^{(t)}\Big(\frac{U}{|\Omega^{(t)}_{s,b}|} \sum_{u \in \Omega^{(t)}_{s,b}} \partial f_u(Z_{1,b}, \ldots, Z_{N_z,b})\Big) + \Psi_\alpha(v_\alpha)^{(t)}$
      // Optional mirroring step for non-negativity
      $Z_\alpha(v_\alpha)^{(t)} \leftarrow |Z_\alpha(v_\alpha)^{(t)}|$
    $t \leftarrow t + 1$
until *convergence or $t >$ max_epochs*

Figure 5.5. PSGLD (Parallel stochastic gradient Langevin Dynamics).

trarily, we make use of the partitioning approach that we introduced in Section 3.2 for reducing the computational burden drastically. By making use of the notions of parts and blocks, we enable parallelism; given a part Ω_s, the SGLD updates can be applied to different blocks b of the latent factors in parallel:

$$Z_\alpha(v_\alpha)^{(t)} = Z_\alpha(v_\alpha)^{(t-1)} - \epsilon^{(t)}\Big(\frac{U}{|\Omega^{(t)}_{s,b}|} \sum_{u \in \Omega^{(t)}_{s,b}} \partial f_u(Z_{1,b}, \ldots, Z_{N_z,b})\Big) + \Psi_\alpha(v_\alpha)^{(t)} \qquad (5.33)$$

where $v_\alpha \in \Omega_{s,b}$. The pseudo-code of PSGLD is given in Figure 5.2.1.

5.2.2. Convergence Analysis

Since we are making use of a biased sub-sampling schema, it is not clear that the samples generated by PSGLD will converge to the Bayesian posterior. In this section, we will define certain conditions on the selection of the parts and provided these conditions hold, we will show that the probability distribution of the samples

$Z^{(t)}_{1:N_z}$ converges to the Bayesian posterior $\mathbb{P}(Z_{1:N_z}|X_{1:N_x})$.

For theoretical use, we define z as the parameter vector, that contains all the factors:

$$z \triangleq [\mathbf{vec}(Z_1)^\top, \ldots, \mathbf{vec}(Z_{N_z})^\top]^\top \tag{5.34}$$

where $\mathbf{vec}(\cdot)$ denotes the vectorization operator. We also define

$$\mathcal{L}(z^{(t)}) \triangleq \log \mathbb{P}(z^{(t)}) + \sum_{u \in [U]} \log \mathbb{P}(X(u)|z^{(t)})$$
$$\hat{\mathcal{L}}(z^{(t)}) \triangleq \log \mathbb{P}(z^{(t)}) + \frac{U}{|\Omega^{(t)}|} \sum_{u \in \Omega^{(t)}} \log \mathbb{P}(X(u)|z^{(t)})$$

where, we make use of $X(u) = X_\nu(u_\nu)$ since $u \equiv (\nu, u_\nu)$. Then, the stochastic noise is given by

$$\xi^{(t)} = \nabla_z \hat{\mathcal{L}}(z^{(t)}) - \nabla_z \mathcal{L}(z^{(t)}). \tag{5.35}$$

Under the following conditions Theorem 5.1 holds.

Condition 5.1. *The step size $\epsilon^{(t)}$ satisfies Equation 3.12.*

Condition 5.2. *The part $\Omega^{(t)}$ is chosen from nonoverlapping parts whose union covers the whole data tensors $X_{1:N_x}$. The probability of choosing a part $\Omega^{(t)}$ at iteration t is proportional to its size:*

$$\mathbb{P}(\Omega^{(t)} = \Omega) = \frac{|\Omega|}{U}.$$

Condition 5.3. $\mathbb{E}[(\xi^{(t)})^k] < \infty$, *for integer $k \geq 2$.*

Theorem 5.1. *Let $q_t(z)$ be the probability density function of the samples $z^{(t)}$ that are generated by PSGLD. Then, the probability distribution of $z^{(t)}$ converges to the*

Bayesian posterior $\mathbb{P}(z|X_{1:N_x})$*:*

$$\lim_{t\to\infty} q_t(z) = \mathbb{P}(z|X_{1:N_x}). \tag{5.36}$$

Proof Sketch. Under Condition 5.2, we can show that $\hat{\mathcal{L}}$ is an unbiased estimator of $\mathcal{L}$; therefore the stochastic noise $\xi^{(t)}$ is zero-mean:

$$\left\langle \xi^{(t)} \right\rangle = 0.$$

The rest of the proof is similar to [92]. Under conditions 1 and 3, we can show that $q_t(z)$ follows the (multi-dimensional) Fokker-Plank equation and therefore the stationary distribution of $q_t(z)$ is $\mathbb{P}(z|X_{1:N_x}) \propto \exp(-\mathcal{L}(z))$. □

5.2.3. Non-negativity Constraints

In an SGD or IGD framework, the latent factors can be kept in a constraint set by using projections that apply the minimum force to keep the variables in the constraint set, as shown in Equation 3.18. However, since we are in an MCMC framework, it is not clear that appending a projection step to the PSGLD updates would still result in a proper MCMC method. Instead, similar to [25], we make use of a simple mirroring trick, where we replace the negative entries of $Z_\alpha^{(t)}$ with their absolute values. Formally, we let $Z_\alpha(v_\alpha)$ take values in the whole $\mathbb{R}$, however we parametrize the prior and the observation models with the absolute values, $|Z_\alpha(v_\alpha)|$. Since $Z_\alpha(v_\alpha)$ and $-Z_\alpha(v_\alpha)$ will be equiprobable in this setting, we can replace the negative elements of $Z_\alpha^{(t)}$ with their absolute values without violating the convergence guarantee.

6. EXPERIMENTS USING ML AND MAP ESTIMATION METHODS

In this chapter, we will focus on evaluating the methods that are described in Chapter 3. In the subsequent chapters, we will evaluate the methods that we have introduced in Chapters 4 and 5. Most of our experiments are conducted on various audio processing applications, whereas we also evaluate our some of our methods on synthetic data and link prediction experiments.

In this chapter, we will apply our ML and MAP estimation methods for estimating the latent factors on several challenging applications. In particular, we will address two audio processing applications by using the MUR algorithm. In these applications, our primary aim is to demonstrate the advantages of the coupled factorization models. Then, we will present our experimental results on large-scale link prediction experiments where we evaluate HAMSI on a distributed matrix factorization task.

6.1. Audio Restoration

Audio restoration is an important audio processing application where the aim is to restore the missing or corrupted parts of an audio signal. This problem often arises in practice; when network packets are dropped during digital communication in an audio streaming scenario or certain parts of a vinyl recording could be corrupted and the corruption would result in 'click' sounds. The problem is illustrated in Figure 6.1.

Audio restoration problem is often cast to a matrix completion problem, where the aim would be to come up with a model for a corrupted audio spectrum X as follows:

$$M(f,t)X(f,t) \approx M(f,t)\hat{X}(f,t), \tag{6.1}$$

where $\hat{X}$ is an approximation of X and M is the binary mask defined in Equation 2.7,

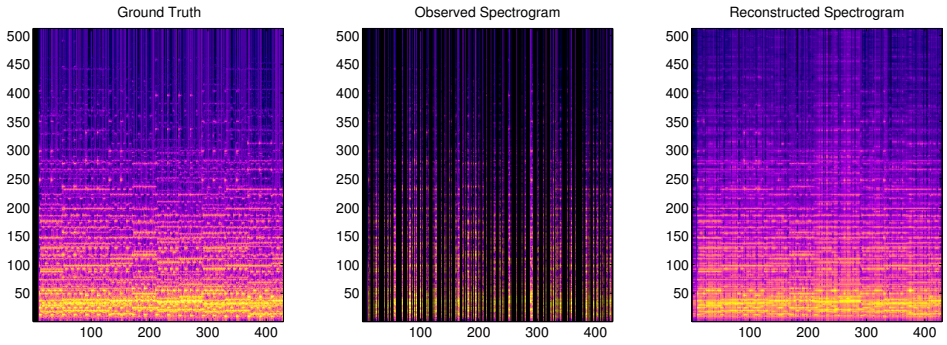


Figure 6.1. Illustration of audio restoration. The aim is to reconstruct the true spectrogram given the observed spectrogram that has missing parts.

denoting whether $X(f,t)$ is missing or not. From the probabilistic point of view, when a data sample is missing, i.e., $M(f,t) = 0$, this is equivalent that the noise variance on $X(f,t)$ is infinite. Therefore, this particular sample does not contribute to the likelihood.

Various audio restoration methods have been proposed in the literature, to name a few [97–99]. The majority of these methods propose different models that are assumed to capture the underlying process of how the audio signals are generated. Impressive results have been reported in these studies, however, these methods have at least one of the two major problems: The first one is that, it is not straightforward to introduce domain specific information to these methods, i.e. the methods that are proposed in [97, 98] both require heavy computational needs. Modifying these methods would yield even slower estimation process while requiring more complex inference schemes. The second problem is, as the case in [99], some methods cannot restore the missing parts if entire frames of audio are missing.

In this section, we will address these problems and illustrate some of the advantages of coupled factorization in audio processing.[5] We present a model for piano spectrogram restoration by using the GCTF, where the main idea is to incorporate different kinds of musical information through coupled factorization while estimating

[5]This section is based on the material published in [100].

the missing parts of the audio: the reconstruction will be aided by an approximate musical score, not necessarily belonging to the played piece, and spectra of isolated piano sounds.

We aim to reconstruct the missing parts of an audio spectrogram of a piano piece $X_1(f,t)$, that represents the short time Fourier transform coefficient magnitude at frequency bin f and time frame t. This is a difficult matrix completion problem since entire time frames (columns of X_1) can be missing, low rank reconstruction techniques are likely to be ineffective.

In order to restore the missing parts in the audio, we form a model that incorporates musical information of chords structures and how they evolve in time. In order to achieve this, we first decompose X_1 by using the NMF model, as follows:

$$X_1(f,t) \approx \hat{X}_1(f,t) = \sum_i D(f,i)E(i,t) \tag{6.2}$$

where D is called as the *spectral dictionary* and E is called as the *excitations*. The main assumption in this modeling strategy is that each column of X_1 would be a noisy realization of a linear combination of the columns of the dictionary matrix D, where the weights of the linear combination is determined by E. Smaragdis and Brown [101] have demonstrated that, provided that model order is properly chosen, the computed factors D and E tend to be semantically meaningful as they correlate well with the intuitive notion of spectral templates of individual notes and a musical score of the corresponding piece, as illustrated in Figure 6.2.

Since, whole columns of X_1 could be missing, we would like to incorporate temporal information to the model. Therefore, we hierarchically decompose the excitation matrix E by using the non-negative matrix factor deconvolution (NMFD) model [102] that aims to express E as a convolution of some basis matrices and their weights, given

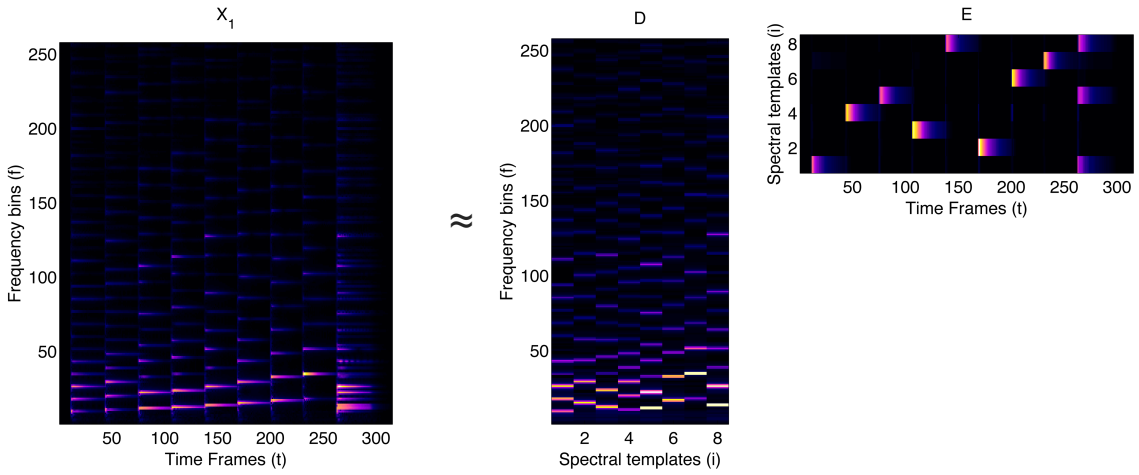


Figure 6.2. An example audio spectrogram decomposed by using NMF. X_1 contains the individual notes and the triad of C major that are played consequently. After the decomposition, D encapsulates the *spectral shapes* of individual notes and E determines the *excitation* patterns of these notes.

as follows:

$$E(i,t) = \sum_{k,\tau} B(i,\tau,k)C(k,t-\tau). \tag{6.3}$$

Since E is similar to a musical score, the basis tensor B would encapsulate both harmonic (in musical sense) and temporal information of the notes that are likely to be used in a musical piece. After replacing E with the decomposed version, we get the following model:

$$\hat{X}_1(f,t) = \sum_{i,\tau,k} D(f,i)B(i,\tau,k)C(k,t-\tau).$$

An example excitation matrix that is decomposed with the NMFD model is illustrated in Figure 6.3.

In order to guide the restoration and provide data-driven regularization, we introduce side information. We provide a score matrix X_2, where $X_2(i,n)$ would denote

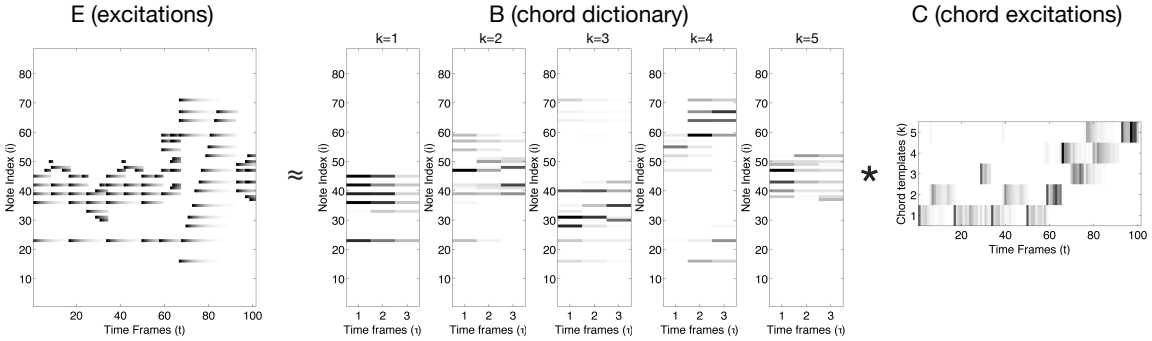


Figure 6.3. An example excitation matrix decomposed by using NMFD. B encapsulates both harmonic and temporal information of the musical score: each slice of B forms a temporal template for the chords that appear in the musical piece. These templates are then activated by the excitation matrix C.

if the note i is played at the time-frame n. This matrix can be possibly obtained from a MIDI file. We also provide an additional audio spectrogram X_3 that contains isolated piano recordings where it is constructed by concatenating isolated recordings corresponding to different notes.

Since we assume X_2 and E would have a similar harmonic structure, we decompose X_2 in the same fashion as we decomposed E:

$$\hat{X}_2(i,n) = \sum_{\tau,k} B(i,\tau,k)G(k,n-\tau)$$

where the musical piece to be reconstructed will share B, possibly played at different times or tempi as modelled by G. Finally, we decompose X_3 by using another NMF model where the spectral dictionary is shared with X_1:

$$\hat{X}_3(f,p) = \sum_{i} D(f,i)F(i,p)T(i,p) \tag{6.4}$$

where T is a $0-1$ matrix, where $T(i,p) = 1(0)$ if the note i is played (not played) during the time frame p and F models the time varying amplitudes of the isolated

notes. Then the ultimate model is given as follows:

$$\hat{X}_1(f,t) = \sum_{i,\tau,k} D(f,i)B(i,\tau,k)C(k,\overbrace{t-\tau}^{d}) \qquad \text{Audio Spectrogram}$$

$$= \sum_{i,\tau,k,d} D(f,i)B(i,\tau,k)C(k,d)Z(d,t,\tau) \tag{6.5}$$

$$\hat{X}_2(i,n) = \sum_{\tau,k} B(i,\tau,k)G(k,\overbrace{n-\tau}^{m}) \qquad \text{Symbolic Music Data (MIDI)}$$

$$= \sum_{\tau,k,m} B(i,\tau,k)G(k,m)Y(m,n,\tau) \tag{6.6}$$

$$\hat{X}_3(f,p) = \sum_{i} D(f,i)F(i,p)T(i,p) \qquad \text{Isolated Notes}$$

$$\tag{6.7}$$

Here, we have introduced new dummy indices d and m, and new (fixed) factors $Z(d,t,\tau) = \delta(d-t+\tau)$ and $Y(m,n,\tau) = \delta(m-n+\tau)$ to express this model in our framework. Figure 6.4 visualises the general structure of the model.

In GCTF notation, we have $N_x = 3$, $N_z = 8$ with $Z_{1:8} \equiv \{D,B,C,Z,G,Y,F,T\}$, and $i_{1:8} \equiv \{f,t,i,\tau,k,d,n,m\}$. The coupling matrix R for this model is defined as follows:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{6.8}$$

In order to evaluate our model, we have conducted several experiments. We have used the MIDI Aligned Piano Sounds (MAPS) piano database [103]: 16 bit 44.1 kHz piano samples are down-sampled to 11.025 Hz and the test files are corrupted by erasing big chunks of samples. In all our experiments the audio is subdivided into frames of 93 milliseconds.

In the experiments, we have used the first 20 seconds of 6 different recordings of 3

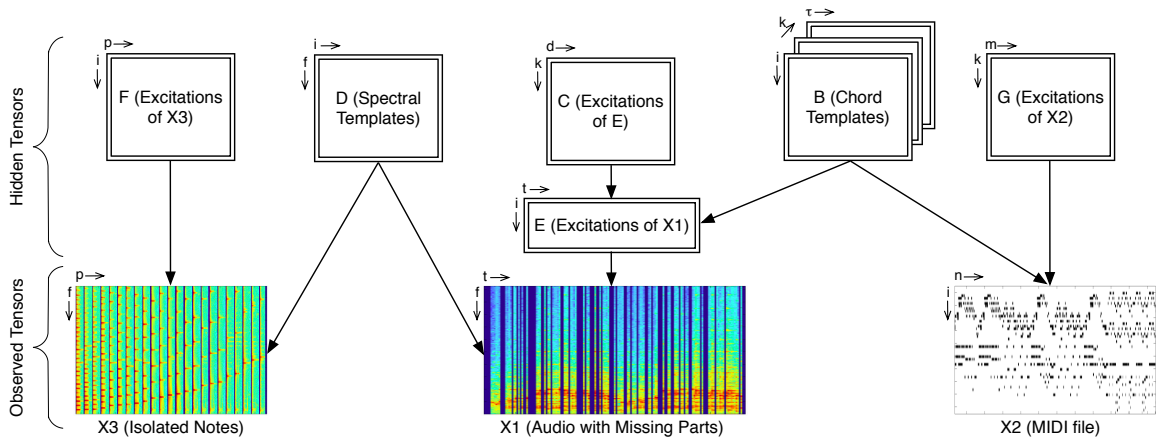
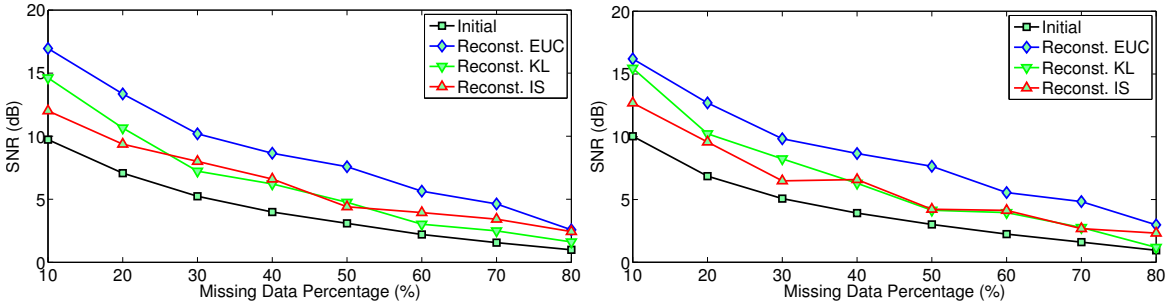


Figure 6.4. General sketch of the audio restoration model. The idea is to incorporate information from the recordings of the instrument and a score of the same genre. The blocks visualize the tensors that are defined in the model and the relation between them. The lower-case letters and arrows near the blocks represent the indices of a particular tensor.

pieces from J. S. Bach. In 2 of these 6 different recordings, the piano samples (X_3) are available for each isolate note. The remaining 4 recordings are from different pianos. In order to obtain the restored version of the corrupted spectra we have simply combined the observed parts of X_1 and the estimated parts of $\hat{X}_1$: $M_1 \circ X_1 + (1 - M_1) \circ \hat{X}_1$, where M_1 is the $0 - 1$ mask that is introduced in Equation 2.7.

In our first experiment, after synthetically corrupting the test files, we have restored them by using their *own* transcriptions as the side information. In the second experiment, we have used transcriptions of *different* pieces. Figure 6.5 illustrates the performance the model for different missing data percentages and different cost functions. For both cases the Euclidean cost function seems to perform better than the others. It can also be observed that, the results of both experiments are similar. One interpretation of this observation is that as long as the musical score (X_2) reflects the chord structure and its temporal evolution of corrupted the audio, it does not necessarily belong to the same piece as $X1$.



(a) First experiment (b) Second experiment

Figure 6.5. Results of the audio restoration experiments. As side information (X_2), we used a) own transcriptions of the test files, b) *different* transcriptions of other test files. Initial SNR is computed by substituting 0 as missing values.

In order to assess the quality of our reconstructions, we measure the signal-to-noise ratio (SNR) between the true and the reconstructed spectrograms. In both cases, we get about 5 dB SNR improvement where 50% of the data is missing; gracefully degrading from 10% to 80% missing data. We believe that the results are encouraging as quite long portions of audio are missing.

6.2. Audio Source Separation

Audio source separation is another key problem in computer music and audio signal processing. The aim is to estimate individual sources from an audio mixture. For instance, in musical signal processing, given an audio recording that contains multiple instruments, one would like to *separate* the signals of the individual musical instruments, that would correspond to different sources in a source separation application. This problem is called underdetermined if the number of channels is less than the number of sources in the mixture. Figure 6.6 illustrates underdetermined source separation problem.

The first approaches to solve underdetermined source separation involved blind source separation techniques [104]. However, audio signals are highly complex, and blind methods fall short in exploiting useful domain specific knowledge. Incorporat-

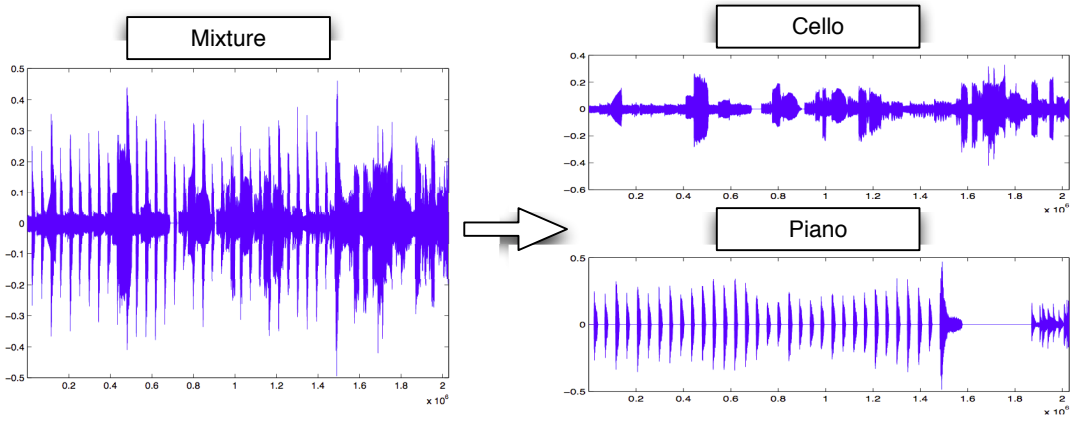


Figure 6.6. Illustration of underdetermined source separation. The aim is to separate individual source signals from an audio mixture.

ing domain specific information via signal models yields to informed source separation methods and there exists several studies that make use of different kinds of information. In the case where the original source signals are known beforehand, [105] presented a method which is based on extracting side information from the original sources and using this information at the source separation process. Another informed source separation method is proposed in [106] which makes use of a temporally aligned transcription of the audio mixture.

In this section, we present two tensor factorization models for informed musical source separation.[6] Both models are based on NMF and the basic idea in our models follows the notion of decomposing the magnitude spectrum of the mixture (X_1) as the multiplication of a spectral dictionary (D) and the corresponding excitations (E), similar to Section 6.1. By this approach, we can obtain the source estimates by Wiener filtering after the factors D and E are estimated:

$$X^{\text{source}} = X_1 \circ \frac{D^{\text{source}} E^{\text{source}}}{\hat{X}_1} \tag{6.9}$$

where D^{source} and E^{source} are the parts of D and E that belongs to the 'source', and $\circ$

[6] This section is based on the material published in [38].

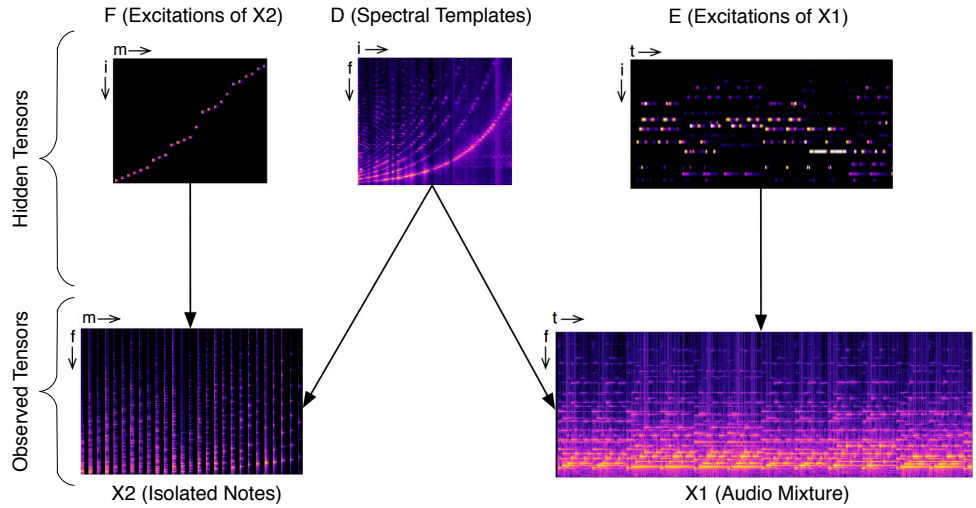


Figure 6.7. General sketch of the first source separation model. The idea is to incorporate information from the recordings of the instruments. The excitation matrix is also restricted by a temporally aligned transcription. The blocks visualize the tensors and the arrows denote the relation between them. The lower-case letters and small arrows near the blocks represent the indices of a particular tensor. Note that N and T matrices are masks that are applied on E and F, respectively.

and $\frac{\cdot}{\cdot}$ denote element-wise product and division.

In our first model, we combine two different NMF models that share the dictionary matrix D. The aim in this model is to incorporate spectral information by coupling the observed mixture with isolated note recordings. Here, the excitation matrix E is further restricted by a temporally aligned transcription of the mixture (N). The model is defined as follows:

$$\hat{X}_1(f,t) = \sum_i D(f,i)E(i,t)N(i,t) \tag{6.10}$$

$$\hat{X}_2(f,m) = \sum_i D(f,i)F(i,m)T(i,m) \tag{6.11}$$

where f is the frequency index, t and m are time frame indices, and i is the index of the spectral templates. Under Euclidean or KL divergences, X_1 is the magnitude

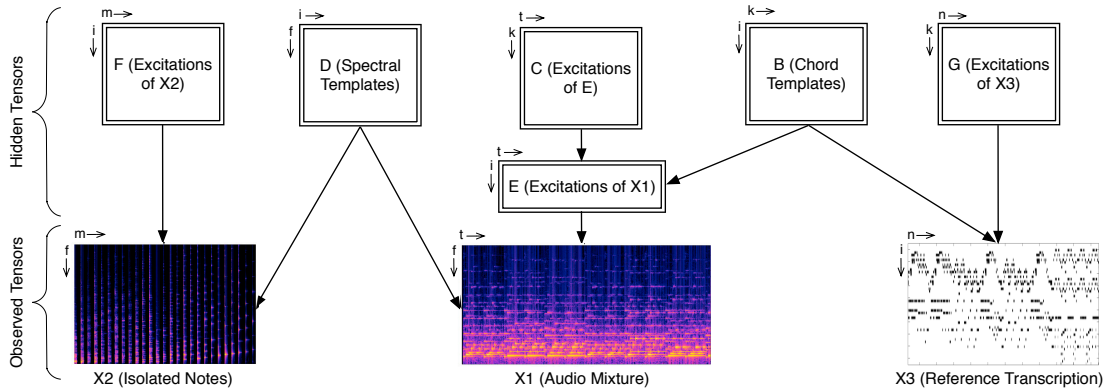


Figure 6.8. General sketch of the second source separation model. The idea is to incorporate spectral information from the recordings of the instruments and harmonic information from an approximate score which is not necessarily aligned. The blocks visualize the tensors and the arrows denote the relation between them. The lower-case letters and small arrows near the blocks represent the indices of a particular tensor.

spectrum of the audio mixture and X_2 is the magnitude spectrum of concatenation of isolated recordings corresponding to different notes. If IS divergence is chosen, both X_1 and X_2 are power spectra. Besides, N is the temporally aligned transcription of the mixture where $N(i,t) = 1(0)$ if the note i is played (not played) during the time frame t. Similarly, T is also a $0-1$ matrix, where $T(i,m) = 1(0)$ if the note i is played (not played) during the time frame m and F models the time varying amplitudes of the isolated notes. Figure 6.7 visualizes the general structure of the model.

In GCTF notation, we have $N_x = 2$, $N_z = 5$ with $Z_{1:5} \equiv \{D, E, N, F, T\}$, and $i_{1:4} \equiv \{f, t, i, m\}$. The coupling matrix R for this model is defined as follows:

$$R = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{6.12}$$

A similar model to this model was proposed in [36] for drum source separation in polyphonic music signals. In that model, the spectral templates are coupled between a polyphonic audio recording and a collection of drum recordings in order to obtain

better drum separation performance.

In our second model, we hierarchically factorize the excitation matrix E as multiplication of a chord dictionary matrix B and its weights C as follows:

$$E(i,t) = \sum_{k} B(i,k)Z(i,k)C(k,t). \tag{6.13}$$

Here the basis matrix B encapsulates the harmonic structure of the music and incorporates additional information to the source separation system. The basic idea behind factorizing the excitation matrix E is to capture the repeated harmonic patterns in the music and form a harmonic basis for the musical piece.

After replacing E with the decomposed version, we get the following model:

$$\hat{X}_1(f,t) = \sum_{i,k} D(f,i)B(i,k)Z(i,k)C(k,t) \tag{6.14}$$

$$\hat{X}_2(f,m) = \sum_{i} D(f,i)F(i,m)T(i,m) \tag{6.15}$$

$$\hat{X}_3(i,n) = \sum_{k} B(i,k)Z(i,k)G(k,n)Y(k,n) \tag{6.16}$$

where X_3 is a score matrix, which can be possibly obtained from a MIDI file: $X_3(i,n)$ is set to a constant value if the i^{th} note is active at time frame n. X_1 and X_3 do not necessarily belong to the same piece, however, in this study we select X_3 as a transcription of X_1.

Furthermore, Z and Y are $0-1$ matrices that allow the model to handle audio mixtures with multiple instruments. $Z(i,k) = 1$ if i^{th} note and k^{th} chord template belong to the same instrument. Similarly, $Y(k,n) = 1$ if the instrument that k^{th} chord template belongs to is active at time n. G models the time varying amplitudes of the chord templates. Figure 6.8 visualizes the general structure of the model.

In GCTF notation, we have $N_x = 3$, $N_z = 8$ with $Z_{1:8} \equiv \{D, B, Z, C, F, T, G, Y\}$,

Table 6.1. Evaluation results of the source separation models. M1 denotes the first model and $M2_d$ denotes the second model where d is the duration of the transcription in seconds. The best results are shown in bold.

	SIR			SAR			SDR		
	$p=0$	$p=1$	$p=2$	$p=0$	$p=1$	$p=2$	$p=0$	$p=1$	$p=2$
M1	12.09	14.82	18.76	9.62	10.23	8.54	7.03	8.55	8.01
$M2_{45}$	7.85	20.08	**21.02**	6.31	**13.25**	6.72	2.06	**12.34**	6.42
$M2_{30}$	7.53	14.51	18.18	6.83	10.97	6.21	2.66	8.66	5.36
$M2_{10}$	6.39	11.17	14.91	8.12	9.35	6.25	2.57	5.37	4.82

and $i_{1:8} \equiv \{f, t, i, k\}$.The coupling matrix R for this model is defined as follows:

$$R = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{6.17}$$

For evaluation of the models, we have synthesized 3 piano and cello duets by using RWC Musical Instrument Sound database [107] and a simple concatenative synthesis algorithm and then we have selected 3 excerpts of 45 seconds from random parts of each piece yielding 9 test cases. In all our experiments the audio is subdivided into frames of 186 milliseconds where the audio spectrum is computed via Modified Discrete Cosine Transform (MDCT).

Since our second model needs only an approximate transcription, we also tested this model on different transcription durations: we used the first 10, 30, and 45 seconds of the transcriptions during the tests.

We have run the inference algorithms for $50 - 75$ iterations for both models and we have used 134 spectral templates that correspond to 88 piano and 46 cello notes. We have also used 80 chord templates for the second model; 50 templates for piano and

30 templates for cello. The factors B, C, D, E, F, and G are initialized randomly and updated during the estimation process. The other factors are clamped to their initial values.

In order to measure the performance of our models, we compute the signal to interference ratio (SIR), signal to artifact ratio (SAR), and signal to distortion ratio (SDR) by using the BSS_{EVAL} toolbox (v3.0) [108]. The evaluation results are given in Table 6.1. It can be observed that, despite the first model uses the temporally aligned transcription, the second model yields a similar performance and it even performs better than the first model for all metrics when KL divergence is chosen. We can also observe that increasing the duration of the transcription that is used in the second model improves the performance of the system which validates the idea behind the model. Some audio examples can be found in `http://www.cmpe.boun.edu.tr/~umut/eusipco2012/`.

6.3. Large-Scale Link Prediction via Distributed Matrix Factorization

In this section, we present the performance of HAMSI on a large-scale, distributed matrix factorization application. We focus on the following problem: ($N_x = 1$)

$$X_1(i,j) \approx \hat{X}_1(i,j) = \sum_k Z_1(i,k) Z_2(k,j) \tag{6.18}$$

where we select $p_1 = 0$ and we do not restrict the factors to be non-negative. Here, we set $i_{1:3} \equiv \{i,j,k\}$.

Figure 6.9 illustrates the partitioning schema that we use in our implementation. Resulting partitions allow us to immediately realize separable subproblems. The memory matrices Ξ_b and Φ_b in Algorithm 3.3 (line 10) are also distributed across the nodes. Thus, we also construct the submatrices Ξ_{1b} and Ξ_{2b} as well as Φ_{1b} and Φ_{2b}.

In our experiments, we consider a distributed architecture that contains two main

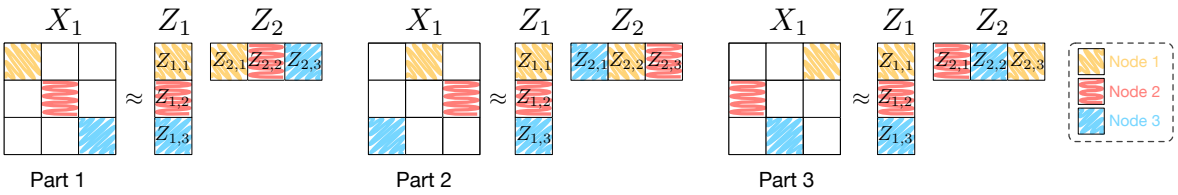


Figure 6.9. Illustration of the *parts* and the *blocks* used in the experiments.

components: (i) data nodes that store the blocks of X_1; (ii) computation nodes that complete the inner HAMSI iterations in parallel. We have chosen to implement HAMSI by a low-level message passing protocol in C using the OpenMPI library. Though, an implementation using higher level distributed computation primitives, like Hadoop MapReduce, is also possible.

In our implementation, each subset has the same number of blocks $B_s = S$ for all s, where S is also the number of available nodes. As illustrated in Figure 6.10, throughout the optimization, the submatrices Z_{1b}, Ξ_{1b} and Φ_{1b} are local to each computation node. On the other hand, at the end of each iteration each node transfers the corresponding submatrices Z_{2b}, Ξ_{2b} and Φ_{2b} to its neighboring node in a cyclic fashion.

We conduct our experiments on a cluster with 15 interconnected computers each of them with 8 Intel Xeon 2.50GHz CPUs and 16 GB of memory. Therefore, provided that the memory is sufficient, we are able to run 120 concurrent processes.

We evaluate HAMSI on three large movie ratings datasets, namely MovieLens 1M, MovieLens 10M, and MovieLens 20M (`grouplens.org`). MovieLens 1M contains 1 million ratings applied to $s_1 = 3883$ movies by $s_2 = 6040$ users, resulting in a sparse data matrix X_1 with 4.3% non-zero entries. For MovieLens 10M, we have 10 million ratings applied to 10681 movies by 71567 users with 1.3% non-zero entries. Finally, in MovieLens 20M, there are 20 million ratings applied to 27278 movies by 138493 users with 0.5% non-zero entries. In all our experiments, we set latent dimension, $s_3 = 50$ and memory size, $M = 5$.

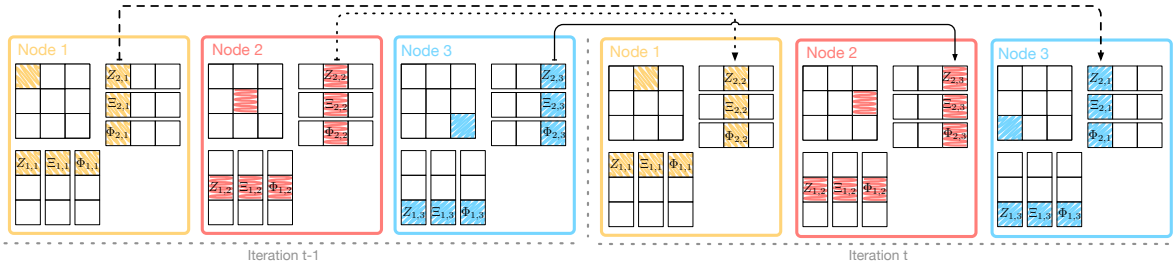


Figure 6.10. Illustration of the communication mechanism with 3 nodes. At each iteration, a node transfers the submatrices Z_{2b}, Ξ_{2b}, and Φ_{2b} to a neighboring node. The submatrices Z_{1b}, Ξ_{1b}, and Φ_{1b} are kept in the same node throughout the optimization.

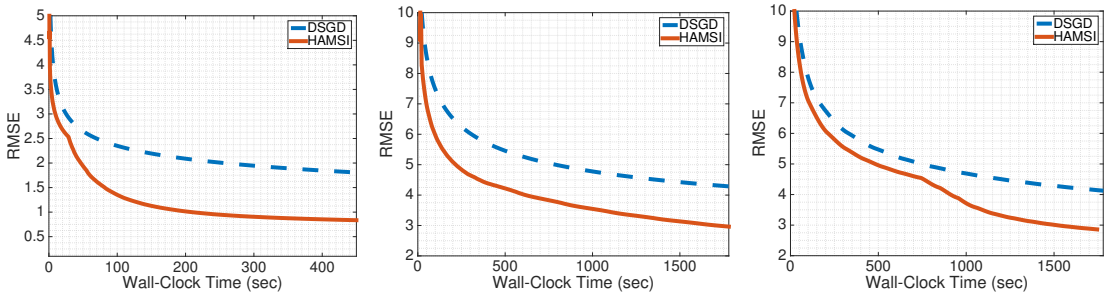
Table 6.2. The list of the parameters in the MovieLens experiments.

	MovieLens 1M		**MovieLens 10M**		**MovieLens 20M**	
	DSGD	HAMSI	DSGD	HAMSI	DSGD	HAMSI
η	$1e-7$	100	$1e-8$	800	$1.2e-8$	500
γ	0.51	0.51	0.6	0.51	0.51	0.51
θ	–	200	–	1000	–	500

In our first set of experiments, we compare HAMSI with the state-of-the-art distributed optimization algorithm for MF, namely, the distributed stochastic gradient descent (DSGD) [12]. In algorithmic sense, DSGD coincides with the DIGD algorithm that we presented in Section 3.2 (with $N_x = 1$, $p_1 = 0$).

In this experiment, on each dataset, we report the root mean squared error (RMSE) between X and $\hat{X}_1$ after running each algorithm for a fixed computation time. RMSE is a standard metric that has been used in link prediction, where the RMSE between X_1 and $\hat{X}_1$ is defined as follows:

$$\text{RMSE}(X_1,\hat{X}_1) = \sqrt{\frac{1}{s_1 s_2}\sum_{ij}\Big(\hat{X}_1(i,j) - X_1(i,j)\Big)^2}. \tag{6.19}$$



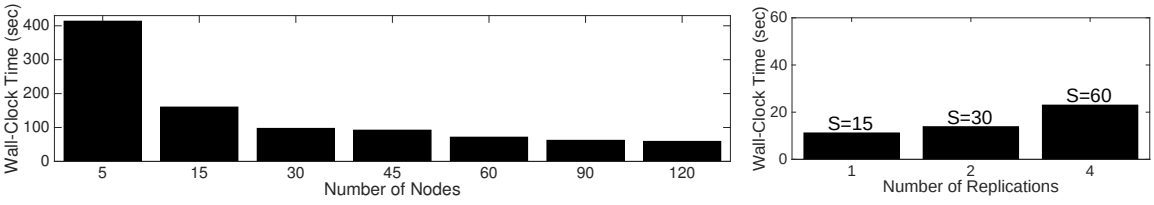
(a) MovieLens 1M, $S = 8$ (b) MovieLens 10M, $S = 15$ (c) MovieLens 20M, $S = 60$

Figure 6.11. RMSE values on MovieLens datasets.

For DSGD, we choose the step-size as $(\eta_{\text{DSGD}}/t)^{\gamma_{\text{DSGD}}}$. For each method, we tried several values for the algorithm parameters (η, γ, θ, η_{DSGD}, γ_{DSGD}) and report the typical results selected among the best performing ones. The list of the selected parameters are given in Table 6.2.

Figure 6.11 shows the RMSE values of HAMSI and DSGD on the three datasets as function of wall-clock time. The parameters used to obtain these results are given in the supplementary document. Since, HAMSI coincides with DSGD until the memory matrices are full (line 20, Algorithm 3.3), we observe the same behavior from both algorithms at the beginning of each experiment. When the memory matrices are filled after M iterations, HAMSI starts to incorporate the approximate curvature information. A single iteration of HAMSI is computationally heavier than DSGD: First, the compact form L-BFGS update requires more computation than simple gradient evaluation. Formally, this overhead is in the order of $\mathcal{O}((M^2 \max(s_1, s_2)s_3)/S^2)$. Second, the communication cost of HAMSI is $\mathcal{O}(s_2 s_3(2M + 1)/S)$ per iteration, whereas the communication cost of DSGD is $\mathcal{O}(s_2 s_3/S)$. However, the use of second order information compensates quickly for this slight increase in computational complexity as it helps HAMSI converge much faster than DSGD. This is clearly seen in Figure 6.11 by the significant gap in RMSE values between the two methods.

In our second experiment, we demonstrate the favorable scalability property of HAMSI. In different computational experiments, we vary the number of nodes from 5



(a) (b)

Figure 6.12. Scalability of HAMSI. a) The size of the data is kept fixed, the number of nodes is increased b) The size of the data and the number of nodes are increased proportionally.

to 120 and run the algorithm for 100 iterations in each setting. Figure 6.12a shows the wall-clock times of HAMSI for different number of computation nodes. The results show that the running time reduces almost linearly as we increase the number of nodes up to $S = 45$. After that the communication overhead seems to dominate and the speed-up becomes marginal.

To illustrate how HAMSI scales with the size of the data, we construct three data matrices, $X_1^{S=15}$, $X_1^{S=30}$ and $X_1^{S=60}$ by artificial replication. The data matrix $X_1^{S=15}$ is the MovieLens 10M, $X_1^{S=30} = h(X_1^{S=15})$ and $X_1^{S=60} = h(X_1^{S=30})$, where $h(X) = [X\ X; X\ X]$. The largest data matrix, $X_1^{S=60}$ is then of size 170896×1145072 with 160 million non-zero entries. For each dataset, the number of computation nodes is also increased in line with the data matrices to keep the data-to-processor ratio constant. Consequently, for each data matrix, the number of computation nodes matches S. Figure 6.12b shows the total running times of HAMSI after 10 iterations for varying data sizes and the number of nodes. The observed results are typical. For the smaller datasets, the wall-clock time stays roughly the same. However, for the larger datasets the communication becomes a key bottleneck due to the increasing traffic on the network.

7. EXPERIMENTS USING DIVERGENCE LEARNING METHODS

In this chapter, we will our evaluate methods that are described in Chapter 4. We will first consider a rather simple case, where we assume that the power parameters are given and estimate the dispersion parameters. Then, we will conduct experiments on the Tweedie compound Poisson model, where we will present an interesting application that aims to predict the lyrics of a song given the feature of the song. Finally, we will present our results where we aim to estimate all the parameters of Tweedie model for all possible values of the power parameter. We will use the MUR algorithm for estimating the factors, unless stated otherwise.

7.1. Dispersion Learning

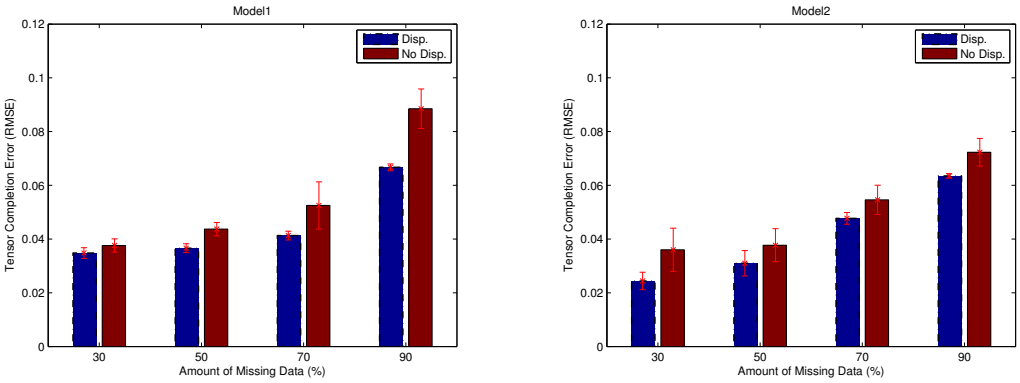
In this section, we evaluate our dispersion estimation method for $p \in \{0, 1, 2, 3\}$, given in Equation 4.21. We will first conduct experiments on synthetic data, then we will apply our method on a real audio processing model.[7]

7.1.1. Synthetic Data

We conduct experiments on two synthetic datasets. First, we illustrate our method on tensor reconstruction model given in Equation 7.1-2, then we use model given in Equation 7.3-5 for the same task. For both of the experiments, we compare two cases. In the first case, we estimate the factors without using the dispersion parameters (i.e. setting $\phi_{1:3} = \phi$) and they do not contribute to the estimation process. In the second case, we also estimate the dispersion parameters by using the proposed method while making inference.

Our method can be used for enhancing missing data recovery when data from

[7]This section is based on the material published in [83].



(a) (b)

Figure 7.1. Illustration of the tensor reconstruction performances of (a) Model 1 and (b) Model2. In both of the figures, completion error of proposed method with (dashed) and without (solid) estimating the dispersion parameters in terms of RMSE under different amounts of missing data are given.

different sources have the same underlying low-rank structure (at least in one mode) but some of the data sets have missing entries. Here, we provide an example to illustrate that how the missing entries can be recovered more accurately when we estimate the dispersion parameters. For both of the datasets, we used our method for data recovery when X_1 has missing entries. We use two coupled tensor factorization models. The first model is given as follows:

$$X_1(i,j,k) \approx \hat{X}_1(i,j,k) = \sum_r A(i,r)B(i,r)C(i,r) \tag{7.1}$$

$$X_2(j,m) \approx \hat{X}_2(j,m) = \sum_r B(j,r)D(k,r) \tag{7.2}$$

where $Z_{1:4} \equiv \{A, B, C, D\}$ and $i_{1:5} \equiv \{i, j, k, m, r\}$. For this model, the simulated data size for observables is $s_1 = 100, s_2 = s_3 = 20, s_4 = 50$ while the latent dimension is $s_5 = 5$. We set the power parameters as $p_1 = 0$, $p_2 = 2$ and true dispersion parameters as $\phi_1 = 0.5$, $\phi_2 = 5$. The hyper-parameters are selected as $\tau_\phi = 10$ and $\kappa_\phi = 1$.

We define our second model as follows:

$$X_1(i,j,k) \approx \hat{X}_1(i,j,k) = \sum_r A(i,r)B(i,r)C(i,r) \tag{7.3}$$

$$X_2(j,m) \approx \hat{X}_2(j,m) = \sum_r B(j,r)D(k,r) \tag{7.4}$$

$$X_3(j,n) \approx \hat{X}_3(j,n) = \sum_r B(j,r)E(n,r) \tag{7.5}$$

where $Z_{1:5} \equiv \{A, B, C, D, E\}$ and $i_{1:6} \equiv \{i, j, k, m, n, r\}$. Here, the simulated data size for observables is $s_1 = 500, s_2 = s_5 = 50, s_3 = s_4 = 200$ while the latent dimension is $s_6 = 5$. This time, we set the power parameters as $p_1 = 1$, $p_2 = 2$, $p_3 = 0$ and the true dispersion parameters as $\phi_1 = 2$, $\phi_2 = 0.7$, $\phi_2 = 1$.

We set different amounts, i.e., $\{30, 50, 70, 90\}\%$, of randomly unobserved elements and report the RMSE scores of ten independent runs for the two cases. We can clearly observe from Figure 7.1 that estimating the dispersion parameters yields better performance on tensor reconstruction.

7.1.2. Drum Source Separation

In this section, we apply our dispersion estimation method on a coupled matrix factorization model for drum separation for professionally recorded audio. This model combines the information that is gathered from the audio mixture, isolated drum sounds and an approximate transcription of drum track of the audio mixture.

Suppose we observe the magnitude spectrum of an audio mixture $X_1(f,t)$, where f and t denote the frequency and time frame indices, respectively. Here, we assume that matrix X_1 is decomposed by using an NMF model:

$$X_1(f,t) \approx \hat{X}_1(f,t) = \sum_i D(f,i)G(i,t) \tag{7.6}$$

where D is the spectral dictionary and G is the corresponding excitation matrix. Since

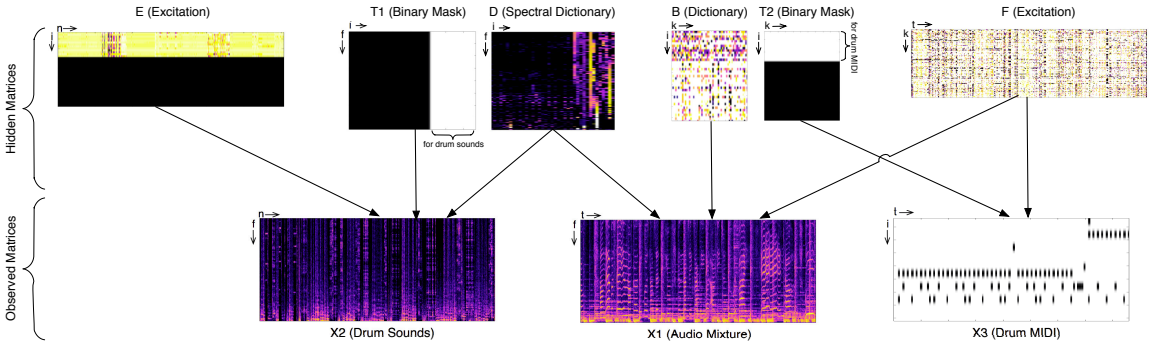


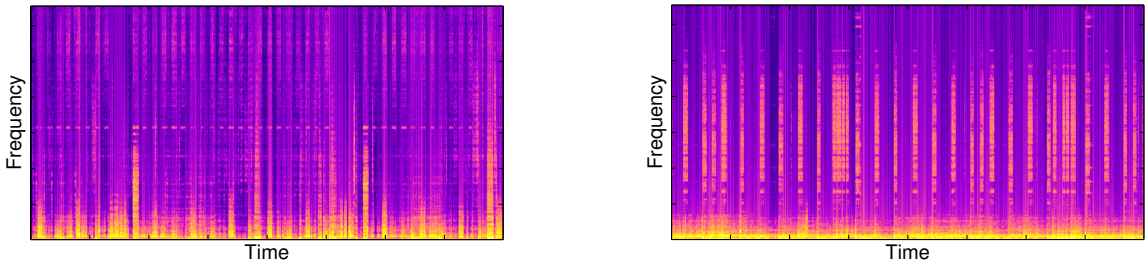
Figure 7.2. Illustration of the drum separation model. The blocks visualize the matrices and the relation between them. The lower-case letters and arrows near the blocks represent the indices of a particular matrix.

the aim of this model is drum track separation, we will assume that some of the spectral templates, say the first I_b columns of D, denoted as $D(:, 1 : I_b)$ model the background sources and the remaining model the drum track. Suppose we observe another magnitude spectrum $X_2(f, n)$, which is obtained from a database of drum sounds. Here f is again the frequency index and n is the time frame index. We can also decompose this matrix using a similar approach:

$$X_2(f, n) \approx \hat{X}_2(f, n) = \sum_i D(f, i)T_1(f, i)E(i, n) \tag{7.7}$$

where D is the same spectral dictionary in Equation 7.6 and E is the excitation matrix for the example drum sounds. Here, T_1 is a pre-determined binary matrix that makes sure that the drum sounds use only the related spectral templates: T_1 takes values of 0 for the background part of the dictionary and 1 otherwise: $T_1(:, 1 : I_b) = 0$ and $T_1(:, I_b + 1 : s_4) = 1$, where s_4 is the number of spectral templates. So far, we have the coupled factorization model of [36], which incorporates the spectral information that is obtained from the drum sounds to the drum separation model.

As we are modeling musical signals, we may also assume the excitation matrix G is composed of the superposition of some certain patterns that repeat over time. With this assumption, we can also factorize the matrix G using another NMF model



(a) (b)

Figure 7.3. Example results of the drum separation experiment. The figures illustrate the reconstructed audio spectra of the drum track X_d, computed via Wiener filtering: $X_d = X_1 \circ \frac{D(:,I_b+1:s_4)B(I_b+1:s_4,:)F}{\hat{X}_1}$. In (a) the dispersion parameters are set to the same value; they do not contribute to the estimation process. In (b) the dispersion parameters are also estimated by the proposed approach.

as follows: $G(i,t) = \sum_k B(i,k)F(k,t)$, where B is the dictionary for the excitations and F denotes the excitations that correspond to this dictionary. By making use of the relation between an excitation matrix and a musical score, we can also couple the matrices B and F with an approximate transcription of the drum track as follows:

$$X_3(i,t) \approx \hat{X}_3(i,t) = \sum_k B(i,k)T_2(i,k)F(k,t). \tag{7.8}$$

Here, $X_3(i,t)$ takes a constant value if a drum event i (e.g. snare hit, hi-hat hit, etc.) is present at time frame t, and becomes 0 otherwise. Furthermore, T_2 is another pre-determined binary matrix similar to T_1, where $T_2(1:I_b,:) = 0$ and $T_2(I_b+1:I,:) = 1$.

Finally, we can define the combined model as follows:

$$\hat{X}_1(f,t) = \sum_{i,k} D(f,i)B(i,k)F(k,t) \qquad \text{Mixture} \tag{7.9}$$

$$\hat{X}_2(f,n) = \sum_i D(f,i)T_1(f,i)E(i,n) \qquad \text{Drum} \tag{7.10}$$

$$\hat{X}_3(i,t) = \sum_k B(i,k)T_2(i,k)F(k,t) \qquad \text{MIDI} \qquad (7.11)$$

where $N_x = 3$, $N_z = 6$, $Z_{1:6} \equiv \{D, B, F, T_1, E, T_2\}$, and $i_{1:5} \equiv \{f, t, n, i, k\}$. Figure 7.2 illustrates this model. The goal is to estimate the latent factors D, B, F, and E as the sources can be separated by Wiener filtering after the factors D, B, and F are obtained (see Equation 6.9). Since we are dealing with audio signals, selecting $p_1 = p_2 = 2$ as Itakura-Saito divergence would be appropriate as suggested in [52]. Besides, we may want to select $p_3 = 1$ as the KL divergence. However, we may wish to give different weights to each observed matrix. In particular, we know that the transcription matrix X_3 is not very accurate, we don't need to fit this matrix precisely. The dispersion parameters play a key role here, as they determine the noise variance of each observed matrix. For this particular example, selecting a large ϕ_3 seems to be an accurate modeling strategy.

We illustrate our dispersion estimation method on our drum source separation model. Here, we conduct our experiments on a famous pop song 'Chasing Pavements' by Adele. Firstly, we estimate the factors without using the dispersion parameters (i.e. setting $\phi_{1:3} = \phi$) and then we also estimate the dispersion parameters by using the proposed method while making inference.

We compute the magnitude spectrum of a 20 second excerpt of the piece and obtain X_1. For X_2, we compute the spectra of the drum sounds that are obtained from the RWC Musical Instrument Sound Database. Finally, we compute X_3 by using an approximate transcription of the drum track of the piece, which can be obtained from online MIDI databases. In our experiments we used $s_4 = 813$ (the number of spectral templates): 13 templates for the drum part and 800 templates for the harmonic part, $s_5 = 100$, and we set the divergence parameters as $p_1 = 2$, $p_2 = 2$, and $p_3 = 1$.

Figure 7.3 visualizes the drum source separation results for the particular experiment. It can be visually observed that estimating the dispersion parameters while estimating the other factors yields better results. We can see that, the high frequency

components of the drum sounds cannot be recovered if the dispersion parameters are fixed to the same number.

7.2. Learning the Divergences in Tweedie Compound Poisson Models

In this section, we will evaluate our methods that are presented in Section 4.1.[8] We will jointly estimate all the variables, where we will restrict $p \in (1, 2)$. Firstly, we will evaluate our methods on modeling symbolic representations for polyphonic music. Secondly, we will define a coupled tensor factorization model and evaluate our methods on prediction of the lyrics of a song from its audio features.

7.2.1. Symbolic Music Modeling

Recent studies suggest that, when designed properly, polyphonic pitch transcription methods with higher level musical models yield better transcription performance [109]. In this section, we present a tensor factorization model for symbolic musical data modeling. This model can be used as a side model for factorization-based audio models.

Symbolic music representation is similar to the sheet representation of music, where symbolic data contain high level musical information, such as note onset times, note durations, and the pitch of the notes that occur in a musical piece. Musical Instrument Digital Interface (MIDI) is one of the standards of symbolic music representation.

One disadvantage of the symbolic representation is that it does not reflect the temporally varying characteristics of the musical notes. We have the information of the velocities at the note onsets, however we cannot obtain the damping structure that the notes naturally have. Therefore, in order to have a better representation, we quantize the time into time-frames and encode the musical information into a matrix $X_1(n, t)$ where n is the note index and t is the time frame index. Here $X_1(n, t)$ simulates the time-varying velocity (volume) of note n during time frame t. For instance, if the note

[8] This section is based on the material published in [81].

n is active at both the time-frame t and $t+1$, then the velocities have the following relation: $X_1(n, t+1) = \alpha X(n, t)$ where $0 < \alpha < 1$. This representation mimics the structure of an excitation matrix of the NMF model.

By construction, only a couple of notes will be active at a given time frame t, therefore X_1 will consist of mostly zeros and some positive values. We can observe that assuming a compound Poisson observation model is quite reasonable as the compound Poisson distribution has a nonnegative probability mass at 0 and a continuous density on positive values.

In this section, we use the NMFD in order to model the modified symbolic musical data. The model is given as follows:

$$X_1(n,t) \approx \hat{X}_1(n,t) = \sum_{\tau,k} D(n,\tau,k)E(k,t-\tau) \tag{7.12}$$

where D is the dictionary tensor and E encapsulates the corresponding excitations. Here $N_x = 1$, $N_z = 2$, $Z_{1:2} \equiv \{D, E\}$, and $i_{1:4} \equiv \{n, t, \tau, k\}$. Apart from using the benefits of the NMF model, this model is also capable of modeling the temporal information of the music.

Since we have only one observed tensor in this model, we can use all three of the inference methods that have been described in Section 4.1. In order to evaluate our methods on modeling the symbolic data, we conduct a restoration experiment similar to the one presented in Section 6.1. We firstly erase some columns (time frames) of the data, then reconstruct the missing parts by using the NMFD model. This reconstruction problem is not trivial as entire time frames (columns of X_1) can be missing.

In these experiments we also use the MAPS database. We use 10 excerpts from 5 different classical music pieces. After generating the X_1 matrices from the symbolic data, we randomly erase some columns of the data which are going to be reconstructed

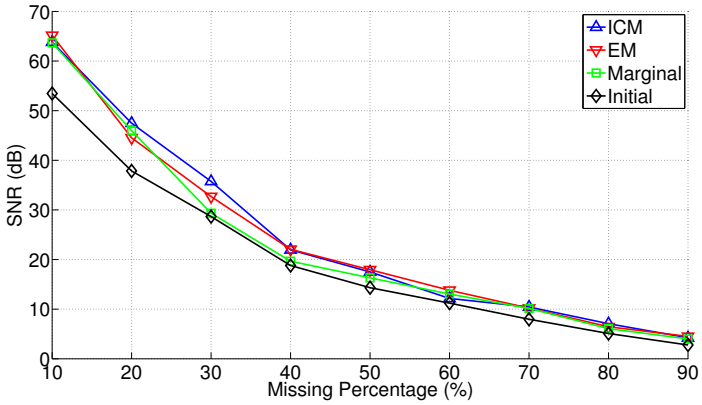
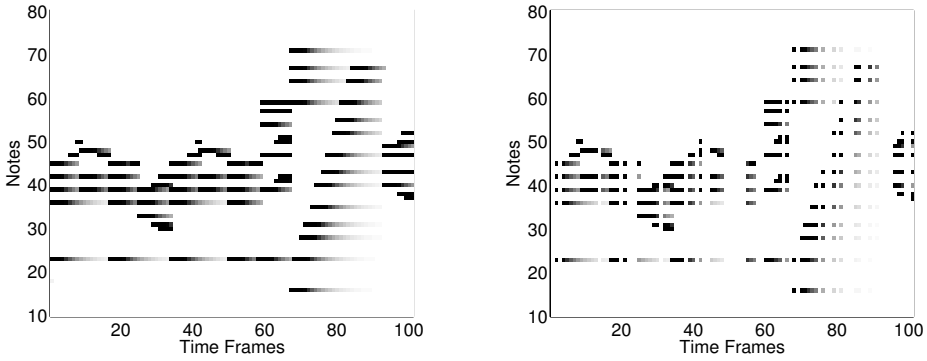


Figure 7.4. Results of the MIDI reconstruction experiments. Initial SNR is computed by substituting 0 as missing values.

later on. In order to obtain the reconstructed symbolic data, we simply combine the observed parts of X_1 and the estimated parts of $\hat{X}_1$: $M_1 \circ X_1 + (1 - M_1) \circ \hat{X}_1$, where M_1 is the binary mask that is introduced in Equation 2.7. We evaluate and compare the performances of our methods by measuring the SNR between the corrupted and the reconstructed symbolic musical data.

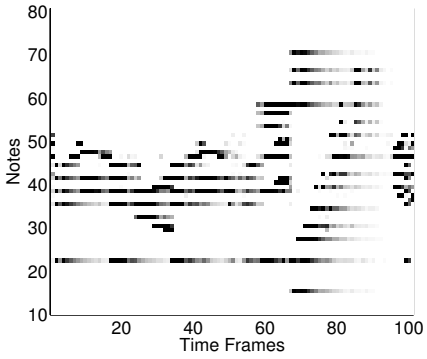
In our experiment settings, the duration of the excerpts is 10 seconds, where we use time frames of 93 milliseconds. We select $\tau_\phi = 5$ and $\kappa_\phi = 3$, $s_3 = 5$, and $s_4 = 50$ for all methods. The results are shown in Figure 7.4.

The results suggest that, the methods always improve the quality of the corrupted symbolic data. The ICM and the EM algorithm give similar results, where the Bayesian method seems to be more sensitive to the missing data than the variational methods. The estimated index parameter p differs for each piece that is reconstructed. Besides, each algorithm finds different p values: the average values for the index parameter are 1.01 (ICM), 1.19 (EM), and 1.26 (Bayesian). For all methods, we get about 4 dB SNR improvement where 50% of the data is missing; gracefully degrading from 10% to 90% missing data. Figure 7.5 visualizes an example reconstruction. It can be observed that the compound Poisson model yields a better reconstruction, where the Gaussian model

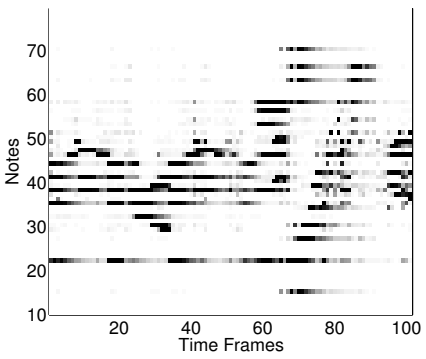


(a) Ground truth

(b) Corrupted data



(c) Reconst. (CP)



(d) Reconst. (Gaussian)

Figure 7.5. Visualization of the symbolic music reconstruction.

introduces spurious notes.

As the results are encouraging even when quite long portions of the data are missing, we can say that modeling the polyphonic music with this approach seems reasonable and might produce good results when used in more complicated models.

7.2.2. Coupled Audio and Lyrics Modeling

In this section, we will illustrate how our approaches can be used with multimodal data. In particular, we will demonstrate that how the index parameter p and the corresponding dispersion ϕ will be estimated under coupled models with mixed observation models where at least one of the observation model is the compound Poisson model.

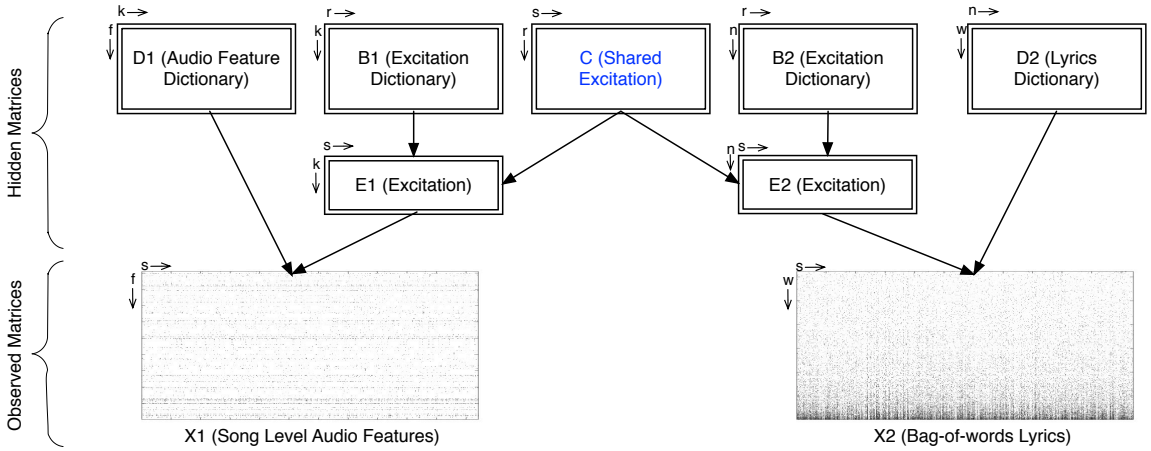


Figure 7.6. Visualization of the coupled factorization model for lyric prediction. The blocks visualize the matrices and the relation between them. The lower-case letters and arrows near the blocks represent the indices of a particular matrix.

In this experiment, we present a coupled matrix factorization model which combines audio features and the lyrics of songs. The aim of this application is to predict the bag-of-words representation of the lyrics of a song given its audio features. This is an interesting application which tries to estimate the keywords that should exist in the lyrics of a song by making use of its audio features and the information from other songs.

Suppose we observe the matrices $X_1 \equiv \{X_1(f,s)\}$ and $X_2 \equiv \{X_2(w,s)\}$, where X_1 contains the song-level audio features and X_2 contains the bag-of-words representation of the lyrics of the songs in their columns. Here, f denotes the audio feature index, s is the song index, w is the word index. We decompose these matrices by using the NMF model as follows:

$$X_1(f,s) \approx \hat{X}_1(f,s) = \sum_k D_1(f,k)E_1(k,s) \tag{7.13}$$

$$X_2(w,s) \approx \hat{X}_2(w,s) = \sum_n D_2(w,n)E_2(n,s) \tag{7.14}$$

where D_1 and D_2 are the dictionary matrices and E_1 and E_2 are the corresponding

excitation matrices. By also assuming a low rank model over the excitation matrices, we hierarchically factorize the excitations by using another NMF model as follows:

$$E_1(k,s) = \sum_r B_1(k,r)C(r,s) \tag{7.15}$$

$$E_2(n,s) = \sum_r B_2(n,r)C(r,s), \tag{7.16}$$

where B_1 and B_2 are the dictionaries for the excitations. With a final assumption that a particular song would use the same columns of the dictionaries B_1 and B_2, we can say that it would have the same excitations. By this approach, we can relate the audio features to the lyrics. We define the ultimate coupled model as follows:

$$\hat{X}_1(f,s) = \sum_{k,r} D_1(f,k)B_1(k,r)C(r,s) \tag{7.17}$$

$$\hat{X}_2(w,s) = \sum_{n,r} D_2(w,n)B_2(n,r)C(r,s). \tag{7.18}$$

Here, $N_x = 2$, $N_z = 5$, $Z_{1:5} \equiv \{D_1, B_1, C, D_2, B_2\}$, and $i_{1:6} \equiv \{f, s, w, k, r, n\}$. Figure 7.6 visualizes this model. Note that, an NMF-based approach is proposed for modeling lyrics in [110] and the authors report successful results.

One can come up with many different applications by using this model; in this study, we focus on the prediction of the lyrics of a song in a bag-of-words representation. It is fairly easy to predict the lyrics of a particular song by using this model: we mark the related parts of the binary mask M_2 (see Equation 2.7) as unobserved, then make predictions by using $\hat{X}_2$.

In our experiments we use the Million Song Dataset (MSD) and the MusiXmatch dataset [4]. The MSD is a free collection of audio features and metadata that are gathered from a large number of music tracks. These features include the key, tempo, time signature, duration, genre tags, year, loudness, and the chroma features of the songs. We use the song level features of random 500 pop songs where we use 2827 features for each song, yielding an audio feature matrix X_1 of size 2827×500.

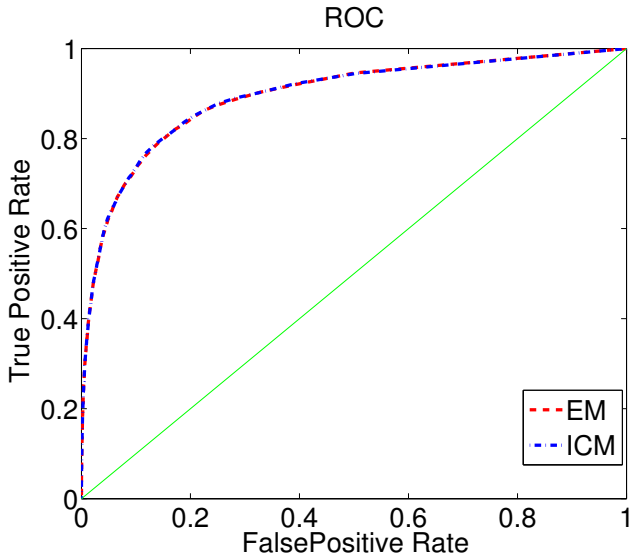


Figure 7.7. The ROC curve belonging to the word detection performance.

The MusiXmatch dataset contains the lyrics of the songs in a bag-of-words representation. This dataset contains more than 230 thousand songs, all being matched with the ones of MSD. Here, we use the number of occurrences of the most common 5000 words of each song, where these 5000 words cover over 92% of all the words in the dataset. We use the same songs that are selected while constructing X_1. Therefore, we have the lyrics matrix X_2 of size 5000×500, where each column of X_2 holds a bag-of-words lyrics of a song.

In our experiment settings, we select $p_1 = 1$ with unitary dispersion, which corresponds to the Poisson observation model. Note that, we could also optimize the dispersion ϕ_1, but this is out of the scope of this study. We set $s_4 = s_6 = 25$ and $s_5 = 10$. In order to estimate the factors, we use the MUR method given in Section 3.1. At each run, we estimate the factors, the index parameter p_2, and the dispersion ϕ_2. We predict the lyrics of random 10 songs at once and we repeat this process 5 times.

In order to assess the quality of the predictions, we measure the word detection performance. We estimate the predictions $\hat{X}_2$ and then consider the words as detected if the corresponding entries in $\hat{X}_2$ are above some threshold. We compute the true

positive and the false positive rates as the performance metrics.

Figure 7.7 visualizes the results. It can be observed that both algorithms yield very similar results. We get more than 80% of true positive rate while keeping the false positive rate less than 20%. Besides, the ICM algorithm seems more advantageous since its computational requirements are much lower than the EM algorithm. These results are encouraging since the lyrics are predicted by solely using the song level audio features.

7.3. Learning Mixed Divergences in the Full Tweedie Family

In this section, we will apply our joint inference algorithm that is presented in Section 4.2 on synthetic and real datasets for a link prediction problem where the aim is to predict the missing parts of an observed tensor.[9]

7.3.1. Synthetic Data

We illustrate the proposed method on a coupled matrix factorization model defined as follows:

$$X_1(i_1, i_3) \approx \hat{X}_1(i_1, i_3) = \sum_{i_4} Z_1(i_1, i_4) Z_3(i_4, i_3) \tag{7.19}$$

$$X_2(i_2, i_3) \approx \hat{X}_2(i_2, i_3) = \sum_{i_4} Z_2(i_2, i_4) Z_3(i_4, i_3) \tag{7.20}$$

Here, we randomly generate the latent variables $Z_{1:3}$, $\phi_{1:2}$, power parameters $p_{1:2}$, and the observed tensors $X_{1:2}$. Our aim is to find the MAP estimates of all the latent variables given X_1 and X_2.

7.3.1.1. Small Scale. Since the true values of the latent variables and the global optimum of Equation 4.16 might not coincide, in order to approximate the global optimum

[9]This section is based on the material published in [84].

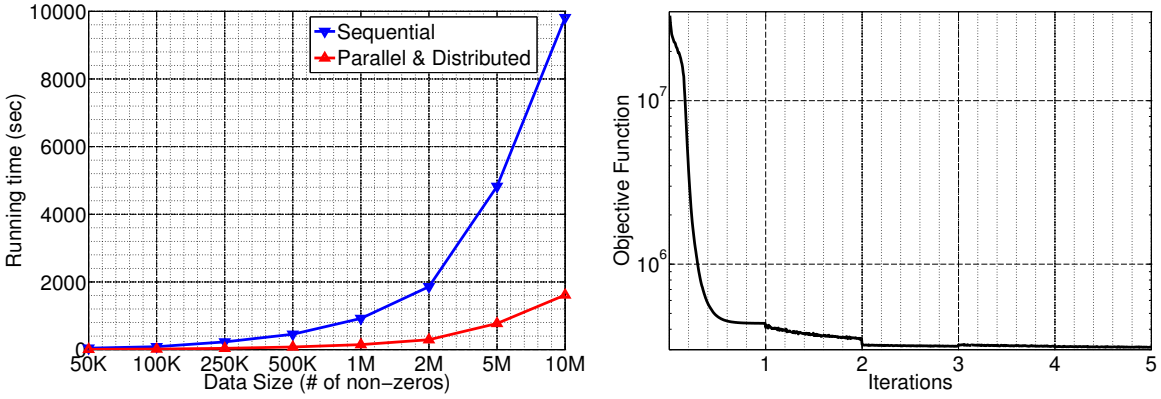
Table 7.1. The results of the experiments on synthetic data.

	$s = 25$	$s = 50$	$s = 100$
MSE (power)	0.0822	0.0563	0.0635
MSE (dispersion)	0.9087	0.6933	0.2763

of Equation 4.16, we first conduct 'oracle' experiments where we assume that the global optimum would be near the true values of the variables. In these experiments, we initialize all the variables ($Z_{1:N_z}$, $\phi_{1:N_x}$, $p_{1:N_x}$) to their true values and run the method presented in Section 4.2 along with the MUR method in order to find the local optimum that is closest to the true values of the variables. We treat the oracle estimates as the global optimum. Then, we re-run the proposed method by initializing the variables randomly. We measure the mean squared error (MSE) between the oracle values of the power and dispersion parameters and the values that we obtain with random initialization.

In our experiments, we set the sizes of the observed indices equal to each other: $s_1 = s_2 = s_3 = s$ and we set $s_4 = 1$. We explore three different values for s: 25, 50, and 100 and repeat the experiments 100 times for each configuration of s. Table 7.1 shows the results. The results show that, even with a small amount of data, our method is capable of estimating the power and the dispersion parameters accurately. Besides, the MSE is gracefully degrading as the size of data increases.

7.3.1.2. Larger Scale. In this section, we conduct a larger scale experiment, where we use DIGD for estimation of the latent factors. In this experiment, we partition the observed indices i_1 and i_2 into $N_1 = N_2 = 4$ parts and i_3 into $N_3 = 8$ parts. In the DIGD procedure, we set the step size $\eta^{(i)} = (a/i)^b$, where i denotes the iteration number. An important observation is that, for the shared factors, the value of a should be smaller than the value that is chosen for the local factors, in order to have a faster convergence. Here, we set $a = 0.01$ for the local factors and $a = 0.001$ for the shared factors. The value of b is set to 0.51 for all factors.



(a) (b)

Figure 7.8. Joint estimation results with DIGD. a) The running times of the algorithms for different data sizes b) The value of the objective function over the iterations for $s_1 = s_2 = s_3 = 320$ and $s_4 = 5$. These iterations correspond to the overall algorithm defined in Equations 4.17-4.19, where at each iteration DIGD is run until convergence.

Figure 7.8a shows the comparison between the proposed approach and the sequential implementation in terms of running times. The picture reveals that we are able to get a 6 fold performance increase on a 8 core machine when we have 10 million entries in the observed matrices. Figure 7.8b shows that the objective is decreased quickly during the iterations, with jumps corresponding to the estimation of divergences and dispersions.

7.3.2. Link Prediction

In this section, we address the missing link prediction task, where the aim is to predict missing parts of an observed tensor. We evaluate our method on the UCLAF dataset [45]. This dataset has a main tensor X_1 of size $146 \times 168 \times 5$, which encapsulates *user-location-activity* informations, where $X_1(i, j, k) = 1$ if the user i visits location j and performs activity k there and $X_1(i, j, k) = 0$ otherwise. The dataset also includes additional side information: the user-location preferences matrix X_2, the location-feature matrix X_3, the user-user similarity matrix X_4, and the activity-activity

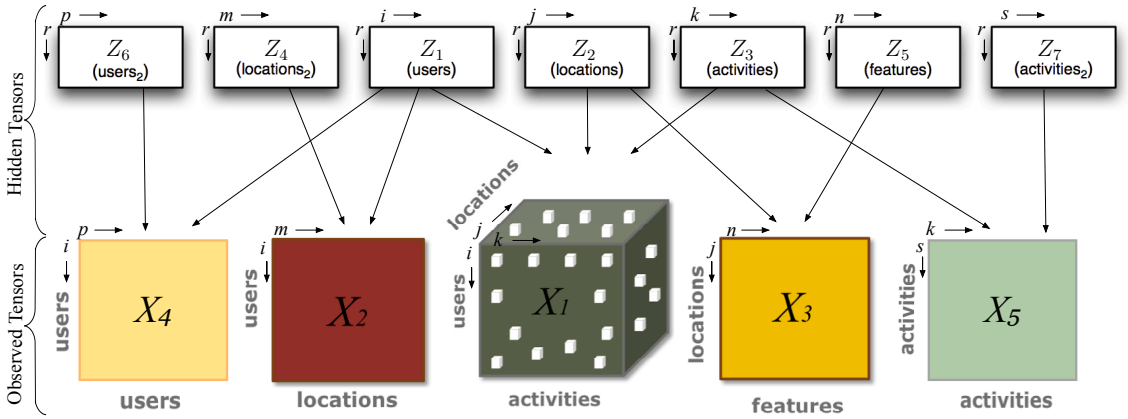


Figure 7.9. General sketch of the link prediction model. The blocks visualize the tensors that are defined in the model. The lower-case letters and arrows near the blocks represent the indices of a particular tensor.

matrix X_5. The aim in this application is to predict the missing parts of X_1.

By following a similar approach to [46], we model this dataset by using the following coupled factorization model:

$$
\begin{aligned}
X_1(i,j,k) \approx \hat{X}_1(i,j,k) &= \sum_r Z_1(i,r)Z_2(j,r)Z_3(k,r), \\
X_2(i,m) \approx \hat{X}_2(i,m) &= \sum_r Z_1(i,r)Z_4(m,r), \\
X_3(j,n) \approx \hat{X}_3(j,n) &= \sum_r Z_2(j,r)Z_5(n,r), \\
X_4(i,p) \approx \hat{X}_4(i,p) &= \sum_r Z_1(i,r)Z_6(p,r), \\
X_5(k,s) \approx \hat{X}_5(k,s) &= \sum_r Z_3(k,r)Z_7(s,r)
\end{aligned} \tag{7.21}
$$

where X_1 is decomposed by using a Parafac model and the remaining observed tensors are decomposed by using matrix factorization (MF) models. We have $i_{1:8} \equiv \{i,j,k,m,n,p,s,r\}$. Figure 7.9 visualizes the general structure of the model. In our experiments, we erase random parts of X_1 at varying amounts (i.e., {10%, 30%, 50%, 70%, 90%}) and evaluate our method on the prediction of the erased parts. We set the

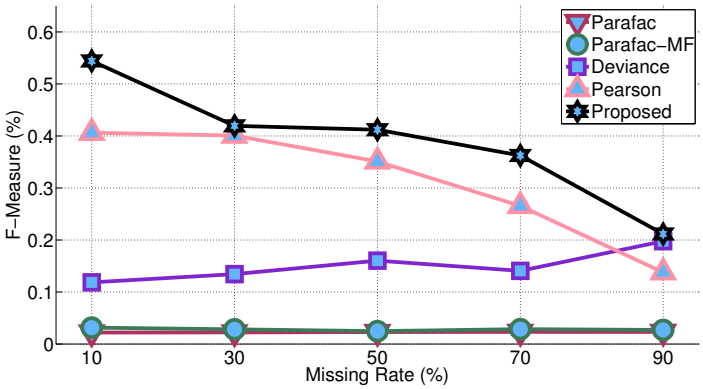


Figure 7.10. F-measure comparison of the proposed method and the state-of-the-art.

number of components to $i_8 = 5$ and use the *F-measure* as the evaluation metric.

We compare our method with two other methods: (i) a Parafac model with Euclidean cost [57] that makes use of only X_1, (ii) the complete model (Parafac-MF) with with Euclidean cost and unitary dispersions ($p_{1:5} = 0$ and $\phi_{1:5} = 1$). The second method can also be considered as extended versions of [34, 46]. We also compare our dispersion estimation method with two different dispersion estimators that have not been explored for coupled factorization models; yet commonly used in the generalized linear models literature, namely the Pearson and mean deviance estimators [111], that are defined as follows:

$$\phi_\nu^{\text{Deviance}} = \frac{1}{S_\nu} \sum_{i=1}^{S_\nu} d_{p_\nu}(x_\nu(i)||\hat{x}_\nu(i)) \tag{7.22}$$

$$\phi_\nu^{\text{Pearson}} = \frac{1}{S_\nu} \sum_{i=1}^{S_\nu} \frac{(x_\nu(i) - \hat{x}_\nu(i))^2}{\hat{x}_\nu(i)^{p_\nu}} \tag{7.23}$$

For these estimators, we replace Equation 4.18 with one of these estimators and use the same approach for estimating the other variables. Note that, each step of our method (Equations 4.17-4.19) monotonically increases the likelihood, whereas the Pearson and deviance estimators do not have such guarantee (for non-Gaussian cases); the likelihood might fluctuate over the iterations when they are used in our iterative schema.

Figure 7.10 visualizes the results of this experiment. We can observe that, both benchmark methods (Parafac and Parafac-MF) perform poorly, where introducing side information (Parafac-MF) results in a tiny improvement in the performance. As we can also observe, apart from monotonically increasing the likelhood and achieving a consistent and sound method, the proposed approach also outperforms the other estimators, in particular when the percentage of missing data is low. Joint estimation of $p_{1:5}$ and $\phi_{1:5}$ yields significant performance improvement, where we have 40% F-measure improvement even when half of the data is missing.

8. EXPERIMENTS USING FULL BAYESIAN INFERENCE METHODS

The aim of this chapter is to demonstrate our MCMC methods on several applications. In particular, we will apply the block Gibbs sampler and the SADA sampler on three different models. Then, we will evaluate PSGLD on large-scale experiments.

8.1. Experiments with Gibbs Sampling

In this section, we will illustrate the block and SADA sampler and Chib's method on three different tensor factorization models: a deconvolution model, a Parafac model, and an extended version of the NMF model. In all our experiments, we set $N_x = 1$.[10]

8.1.1. Non-negative Deconvolution

Convolutive models emerge in various fields such as audio processing, image processing or seismic sciences. In order to illustrate the sampling approaches given in Section 5.1, we first take the deconvolution problem as an example and define it as a tensor factorization problem as follows:

$$\begin{aligned} X_1(t) \approx \hat{X}_1(t) &= \sum_r Z_1(r) Z_2(\overbrace{t-r}^{d}) \\ &= \sum_{r,d} Z_1(r) Z_2(d) Z_3(d,t,r) \end{aligned} \tag{8.1}$$

where Z_1 and Z_2 are the convolved signals and $\hat{X}_1$ is the output signal. Similar to the model that we presented in Section 6.1, we define a dummy index d and a dummy tensor $Z_3(d,t,r) = \delta(d-t+r)$.

In order to build the samplers, we first start by defining the index sets for this

[10] This section is based on the material published in [85].

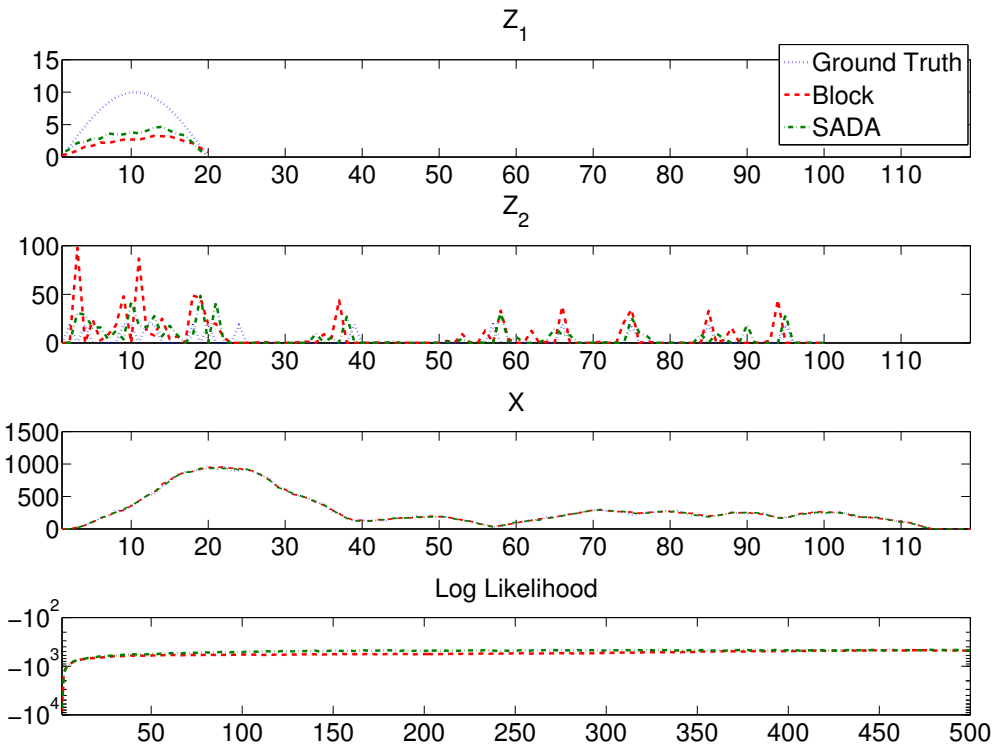


Figure 8.1. Inference results for the deconvolution model. From top to bottom: the first and second figures show the real and the estimated values for the first factor (Z_1) and the second factor (Z_2), respectively. Third: Observed signal (X_1) and the model predictions ($\hat{X}_1$). Fourth: Log likelihood vs iteration plots of the samplers.

particular model. In GCTF notation, we have $N_z = 3$ with $i_{1:3} \equiv \{t, r, d\}$. After placing these index sets in Algorithm 5.1, we obtain the block Gibbs sampler. Similarly, we can also obtain the SADA sampler for this model by placing these index sets in Algorithm 5.1.1. The inference results of block and SADA samplers on a toy problem are illustrated in Figure 8.1. The results show that, even thought the problem is highly ill-posed, our methods can successfully estimate the shapes of the signals, where SADA yields better results in terms of visual comparison.

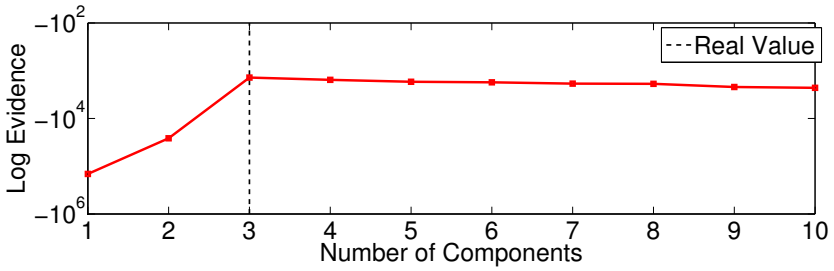


Figure 8.2. Model selection results for Parafac. The marginal Likelihood of the three-way observed data under the CP model ($\int dZ_{1:3}\, p(X|Z_{1:3})p(Z_{1:3})$) is estimated by using Chib's method.

8.1.2. Model Selection in PARAFAC

We conduct our second set experiment on estimating the model order of a Parafac (CP) model, that is given as:

$$X_1(i,j,k) \approx \hat{X}_1(i,j,k) = \sum_m Z_1(i,m)Z_2(j,m)Z_3(k,m) \tag{8.2}$$

where the three-way tensor X_1 is decomposed into three matrices, Z_1, Z_2, and Z_3. We have $N_z = 3$ with $i_{1:4} \equiv \{i,j,k,m\}$. In practice, the optimal number of components (indexed with m above) is not known beforehand and should be estimated.

In order to test our approach for model selection, we generated synthetic data where $s_1 = 10$, $s_2 = 5$, $s_3 = 8$, and $s_4 = 3$ and applied Chib's method to estimate marginal likelihood of the observed tensor under CP models with different number of components. Figure 8.2 shows the marginal likelihood estimates of the synthetic data for different number of components. It can be seen that the marginal likelihood estimate is at the highest when the correct number of components is selected.

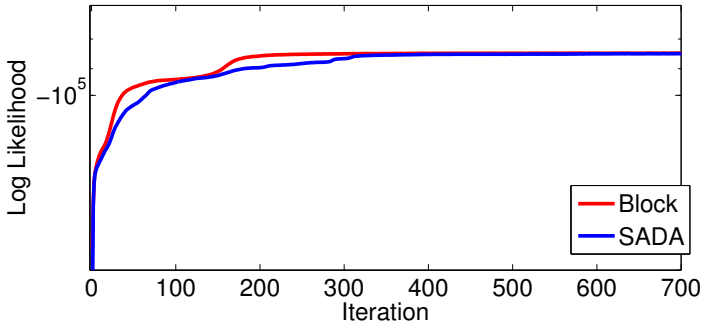


Figure 8.3. Log-likelihood vs iteration plots for the block sampler and SADA sampler.

8.1.3. Hierarchical NMF

We conduct our third experiment on an extension of the NMF model, which is similar to the audio model defined in Section 6.2:

$$\hat{X}_1(f,t) = \sum_{i,k} Z_1(f,i) Z_2(i,k) Z_3(k,t) \tag{8.3}$$

where $X_1(f,t)$ is the observed magnitude spectrum of the audio, f is the frequency index, and t is the time-frame index. Here, $N_z = 3$ with $i_{1:4} \equiv \{f,t,i,k\}$.

We have run both the block sampler and the SADA sampler on a short polyphonic piano sound. We have first estimated and fixed the spectral dictionary Z_1 than run the inference algorithms. We have used $s_3 = 4$ spectral templates and $s_4 = 3$ chord templates while having $s_1 = 1025$ frequency bins and $s_2 = 86$ time frames. Figure 8.3 shows the log-likelihoods of the algorithms. It can be observed that, both algorithms converge smoothly, where the computational requirements of SADA is significantly lower.

8.1.4. Experiments on Alpha-Stable Matrix Factorization

In this section, we will evaluate αMF on both synthetic and audio data. Our implementations is mostly in Matlab, apart from α-stable density evaluation and random

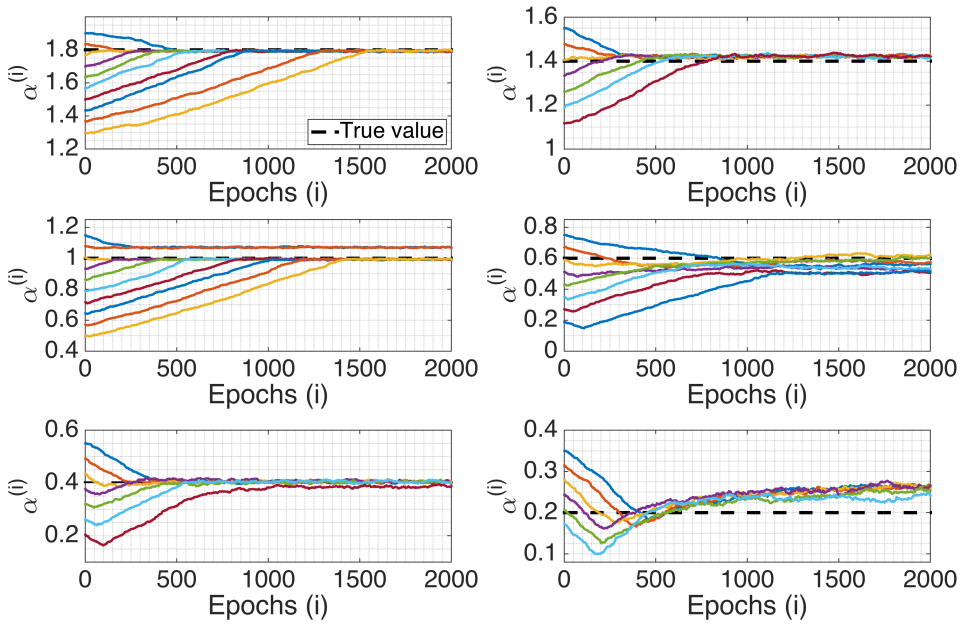


Figure 8.4. Results of the synthetic data experiments.

number generation algorithms, which are implemented in C.

8.1.4.1. Experiments on Synthetic Data. We first conduct experiments on synthetic data, where the aim is to validate our inference procedure. In these experiments, given a fixed α, we generate the latent variables W, H, Φ, S, and the observed complex matrix X by using the generative model given in Equation 5.20. Then, given the observed matrix X, we run our inference algorithm after initializing all the latent variables randomly. In our experiments, we set $s_1 = 50$, $s_2 = 80$, $s_3 = 2$, and $\sigma_\alpha^2 = 0.001$ and we repeat this experiment for several values of α.

In this experiment, we report the results of the estimation of α, since it is the most prominent variable, determining the structure of the distribution. In Figure 8.4, we visualize the samples $\alpha^{(t)}$ that are generated by our algorithm for different true α values. The results show that, even though the initial samples, $\alpha^{(0)}$, might be far from the true value of the variable, our inference algorithm can successfully locate the mode near the true α and starts sampling around that mode, even when the observations are coming from an extremely heavy-tailed distribution ($\alpha = 0.2$).

8.1.4.2. Experiments on Audio. In our next set of experiments, we evaluate αMF on real audio data. We compare αMF with Itakura-Saito NMF [52]; a MF model that is often used in audio processing, having the following underlying probabilistic model:

$$W(f,k) \sim \mathcal{IG}(W(f,k); a_w, b_w), \quad H(k,n) \sim \mathcal{IG}(H(k,n); a_h, b_h)$$
$$S(f,n,k)|W,H \sim \mathcal{N}_c(S(f,n,k); 0, W(f,k)H(k,n))$$
$$X(f,n) = \sum_k S(f,n,k) \tag{8.4}$$

where $\mathcal{IG}$ denotes the inverse gamma distribution. Here, X is taken as a time-frequency representation of the audio signal, with the indices f and n denoting the frequencies and the time-frames, respectively. IS-NMF appears as a special case of αMF: if we set $\alpha = 2$, the generalized gamma distribution becomes the inverse gamma distribution, Φ becomes deterministic, and therefore αMF reduces to IS-NMF.

IS-NMF is considered as an important model for audio modeling since there is a rigorous statistical interpretation of the model from the waveform level to the power spectra level: if we assume that all the time-frames are independent and wide-sense stationary (WSS), we can show that all the entries of the short-time Fourier transform (STFT) of the signal are indeed independent and distributed with a complex centered isotropic Gaussian distribution [112] whose variances correspond to the power spectral density (PSD) of the signal. In this sense, IS-NMF models the PSD of a WSS signal by using a low rank approximation.

However, the assumption of the time-frames being WSS can be restrictive for various types of audio signals that have impulsive nature, such as speech. The interest of αMF in this context is that it generalizes IS-NMF by relaxing the WSS assumption and assumes that all the time-frames are independent and stationary harmonizable α-stable processes. With such an assumption, we can show that the STFT coefficients are still independent but distributed with a $\mathcal{S}\alpha\mathcal{S}_c$ distribution, generalizing the WSS case $\alpha = 2$ [72].

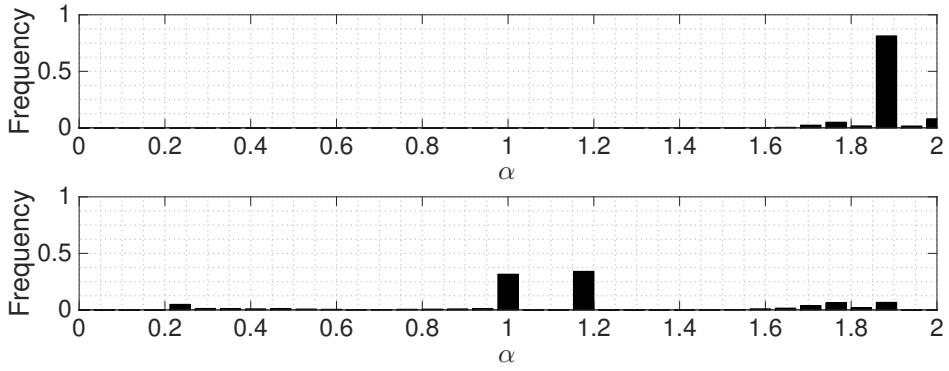


Figure 8.5. Histograms of α for noise (top) and speech (bottom).

We conduct our experiments on NOIZEUS noisy speech corpus [113]. This dataset contains 30 sentences that are uttered by 3 female and 3 male speakers. These sentences are corrupted by using 8 different real noise signals (train, babble, car, exhibition hall, restaurant, street, airport, train-station) at 4 different signal-to-noise ratio (SNR) levels. We analyze the signals by using the STFT with a Hamming window of length 512 samples and 75% overlap.

Firstly, we run αMF on each audio signal (30 clean speech and 8 noise signals). For each signal, we generate 2000 samples where we discard the first 100 of them as the burn-in period. We repeat this procedure three times with different initializations and combine all the samples in two groups: clean speech and noise. We use $s_3 = 5$ for each noise signal and $s_3 = 10$ for each speech signal, and we set $\sigma_\alpha^2 = 0.01$.

Figure 8.5 shows the histograms of α for speech and noise. We can observe that, for the noise signals, the posterior distribution of α is concentrated near $\alpha = 1.89$, i.e. almost Gaussian, whereas we obtain two modes at $\alpha = 1.2$ and $\alpha = 1$ for the clean speech. This is expected because it has long been observed that informative signals such as speech tend to exhibit heavier tails than noises occurring in practice, justifying the use of α-stable models in audio [68]. More interestingly, this outcome provides a sound foundation to the recent empirical results obtained in [72], where the authors demonstrated that $\alpha = 1.2$ is the best performing exponent of the generalized Wiener filter, that implicitly assumes that the audio signals are stable distributed.

Secondly, we compare αMF with IS-NMF on a speech enhancement application, where the aim is to recover the clean speech signal, given a noisy speech signal. In this experiment, we follow a semi-supervised approach and use a slightly different model for the noisy mixtures, given as follows: $X^{\text{mix}}(f,n) = X^{\text{sp}}(f,n) + X^{\text{no}}(f,n)$, where

$$X^{\text{sp}}(f,n) \sim \mathcal{S}\alpha\mathcal{S}_c\Big(X^{\text{sp}}(f,n); [\sum_k W^{\text{sp}}(f,k)H^{\text{sp}}(k,n)]^{1/\alpha^{\text{sp}}}\Big), \tag{8.5}$$

$$X^{\text{no}}(f,n) \sim \mathcal{S}\alpha\mathcal{S}_c\Big(X^{\text{no}}(f,n); [\sum_k W^{\text{no}}(f,k)H^{\text{no}}(k,n)]^{1/\alpha^{\text{no}}}\Big). \tag{8.6}$$

Here, 'sp' denotes the speech and 'no' denotes the noise. For IS-NMF we set $\alpha^{\text{sp}} = \alpha^{\text{no}} = 2$. For αMF, we set $\alpha^{\text{sp}} = 1.2$ and $\alpha^{\text{no}} = 1.89$, as suggested by the results above.

For each model, we first train the dictionary matrix W^{sp} on the first 20 clean speech signals (2 female and 2 male speakers) by using the following approach. We concatenate the STFTs of the speech signals to obtain X. Then, we run the Gibbs sampler for 3000 epochs where we set W^{sp} to the Monte Carlo average (see Equation 1.15) by using the last 200 samples. The number of columns of W^{sp} is chosen as $s_3^{\text{sp}} = 100$.

At testing, for each input SNR, we apply both models on 80 different noisy mixtures, where we fix W^{sp} and sample the rest of the latent variables, including W^{no}. Note that, the noisy speech signals are obtained by combining 8 different noise signals with 10 clean speech signals that are not used during training. For each mixture, we set $s_3^{\text{no}} = 5$ and generate 2500 samples where we use the last 50 samples to estimate the posterior expectations of X^{sp} and X^{no}.

For evaluating the quality of the estimates we use the signal-to-distortion ratio (SDR), signal-to-interference ratio (SIR), and signal-to-artifact ratio (SAR) that are computed with BSS_{EVAL} version 3.0 [108]. Figure 8.6 shows the results. We can observe that, both models perform poorly when the input SNR is low. However, as we increase the input SNR, the structure of the speech becomes more prominent, and we see that αMF becomes more advantageous in terms of all the objective measures. We

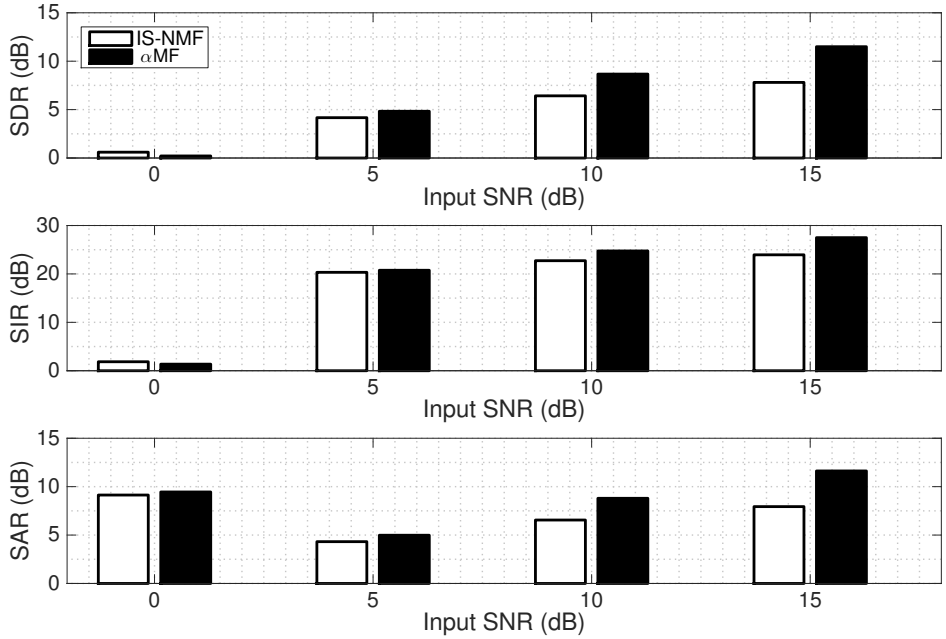


Figure 8.6. Evaluation results of IS-NMF and αMF on speech enhancement. Note that, IS-NMF coincides with αMF when $\alpha^{\mathrm{sp}} = \alpha^{\mathrm{no}} = 2$.

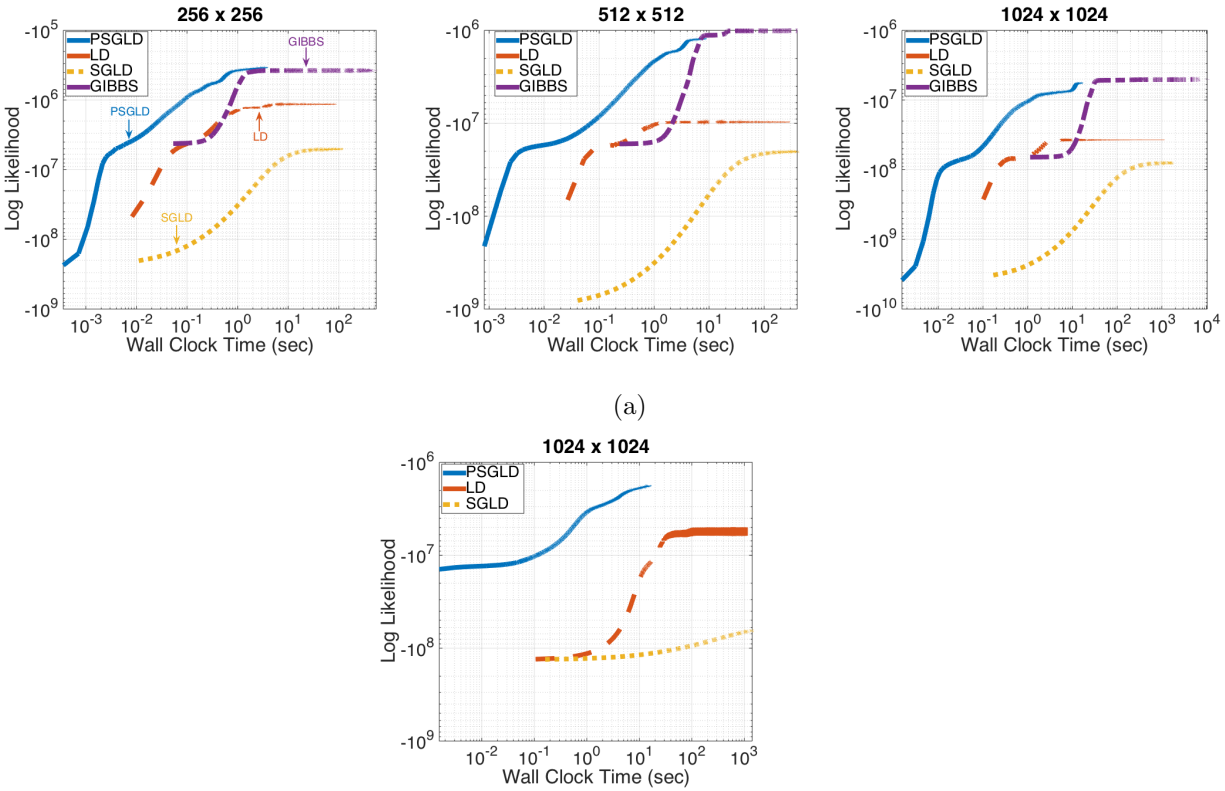
obtain 4dB SDR improvement when the input SNR is 15dB. Besides, αMF results in less interference and less artifacts as measured by SIR and SAR. These differences are statistically significant with 5% significance level.

We would like to note that there have been several extensions on IS-NMF that aim to incorporate the temporal and spatial structure of speech signals into the model [114, 115]. As a possible future direction, we believe that the performance of αMF can be further improved by extending the model in similar aspects.

8.2. Experiments with PSGLD

In this section, we will evaluate PSGLD on synthetic and real datasets. We will conduct our experiments on the NMF model:

$$X_1(i,j) \approx \hat{X}_1(i,j) = \sum_k Z_1(i,k) Z_2(k,j). \tag{8.7}$$



(a)

(b)

Figure 8.7. Shared-memory experiments on PSGLD with a) the Poisson observation model b) the compound Poisson observation model.

with $i_{1:3} \equiv \{i, j, k\}$. PSGLD can be beneficial in two different settings: 1) a shared-memory setting, where we implement PSGLD on a graphics processing unit (GPU) where the computation is done on a single computer 2) a distributed setting, where we implement PSGLD on a cluster of computers by using a message passing protocol.

We will compare PSGLD with different MCMC methods, namely the Gibbs sampler presented in Section 5.1, Langevin Dynamics [91], and SGLD [22]. It is easy to derive the update equations required by the gradient-based methods (LD,SGLD,PSGLD) for this model. However, developing a Gibbs sampler for this general model is unfortunately not obvious as discussed in Section 5.1. Therefore, we develop a Gibbs sampler by using the augmented Poisson model given in Equation 5.1 which requires the source tensor $S_1(i, j, k)$ to be sampled at every epoch.

8.2.1. Shared-Memory Setting

In this section, we will compare the mixing rates and the computation times of all the aforementioned methods in a shared-memory setting. We will first compare the methods on synthetic data, then on musical audio data.

We conduct all the shared-memory experiments on a MacBook Pro with 2.5GHz Quad-core Intel Core i7 CPU, 16 GB of memory, and NVIDIA GeForce GT 750M graphics card. We have implemented PSLGD on the GPU in CUDA C. We have implemented the other methods on the CPU in C, where we have used the GNU Scientific Library and BLAS for the matrix operations.

8.2.1.1. Experiments on Synthetic Data. In order to be able to compare all the methods, in our first experiment we use the Poisson-NMF model. We first generate Z_1, Z_2, and X_1 by using the generative model. Then, we run all the methods in order to obtain the samples $\{Z_1^{(t)}, Z_2^{(t)}\}_{t=1}^{T}$. For simplicity, we choose $s_1 = s_2$ and we set $s_3 = 32$. In order to obtain the blocks, we make use of the same partitioning approach that is described in Section 6.3. Since the sizes of all the parts are the same, Condition 5.2 is satisfied.

In LD, we use a constant step size ϵ, whereas in SGLD and PSGLD, we set the step sizes as $\epsilon^{(t)} = (\eta/t)^{\gamma}$, where $\gamma \in (0.5, 1]$. For each method, we tried several values for the parameters and report the results for the best performing ones. In LD we set $\epsilon = 0.2$, in SGLD we set $\eta = 1$, $\gamma = 0.51$, and in PSGLD we set $\eta = 0.01$ and $\gamma = 0.51$. The results are not very sensitive to the actual value of η and γ, provided these are set in a reasonable range. Furthermore, in SGLD, we draw the sub-samples $\Omega^{(t)}$ with a with-replacement manner, where we set $|\Omega^{(t)}| = s_1 s_2/32$.

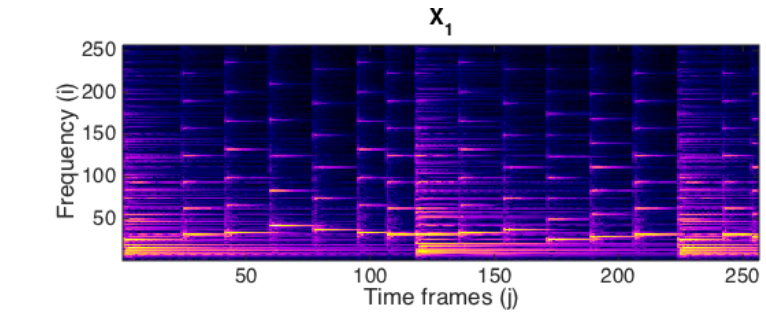
Figure 8.7a shows the mixing rates and the running times of the methods under the Poisson model for different data sizes. While plotting the log-likelihood of the state of the Markov chain is not necessarily an indication of convergence to the stationary

distribution, nevertheless provides a simple indicator if the sampler is stuck around a low probability mode. We set the number of rows $s_1 = 256$, 512, 1024 and we generate $T = 10000$ samples from the Markov chain with each method. We can observe that, in all cases, SGLD achieves poor mixing rates due to the with-replacement sub-sampling schema while LD achieves better mixing rates than SGLD. Moreover, while the LD updates can be implemented highly efficiently using BLAS, the reduced data access of SGLD does not reflect in reduced computation time due to the random data access pattern when selecting sub-samples from X_1.

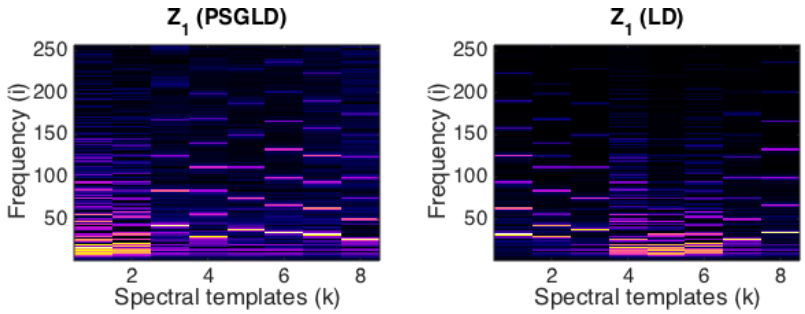
The results show that PSGLD and the Gibbs sampler seem to achieve much better mixing rates. However, we observe an enormous difference in the running times of these methods – PSGLD is 700+ times faster than the Gibbs sampler on a GPU, while achieving virtually the same quality. For example, in a model with $s_1 = 1024$ rows, the Gibbs sampler runs for more than 3 hours while PSGLD completes the burn-in phase in nearly 1 second and generates 10K samples from the Markov chain in less than 15 seconds, even when there are more than 1 million entries in X_1. Naturally, this gap between PSGLD and the Gibbs sampler becomes more pronounced with increasing problem size. We also observe that PSGLD is faster than LD and SGLD by 60+ folds while achieving a much better mixing rate.

We also evaluate PSGLD with the compound Poisson observation model ($\phi = 1$, $p = 1.5$). As we described in detail in Section 4.1, even though the probability density function of this distribution cannot be written in closed-form analytical expression, fortunately we can still generate random samples from the distribution in order to obtain synthetic X_1, by using the generative model given in Equation 4.4.

Since deriving a Gibbs sampler for the compound Poisson model is not obvious, we will compare only LD, SGLD, and PSGLD on this model. Figure 8.7b shows the performance of these methods for $s_1 = s_2 = 1024$. We obtain qualitatively similar results; PSGLD achieves a much better mixing rate and is much faster than the other methods.



(a)



(b)

Figure 8.8. a) The audio spectrum of a short piano piece b) The spectral dictionaries learned by PSGLD and LD.

8.2.1.2. Experiments on Audio. In our next experiment, we decompose the audio spectrum given in Figure 8.8a by using the NMF model and visually compare the dictionary matrices that are learned by LD and PSGLD. The size of X_1 is $s_1 = s_2 = 256$ and we set $s_3 = 8$. For PSGLD, we use $S = 8$ and choose the parts in cyclic order at each iteration. With each method, we generate 10000 samples but discard the samples in the burn-in phase (5000 samples). Figure 8.8b shows the Monte Carlo averages that are obtained by different methods. We observe that PSGLD successfully captures the spectral shapes of the different notes and the chords that occur in the piece, even though the method is completely unsupervised. We also observe that LD is able to capture the spectral shapes of most of the notes as well, and estimates a less sparse dictionary. Furthermore, PSGLD runs in a much smaller amount of time; the running times of the methods are 3.5 and 81 seconds respectively for PSGLD and LD – as a reference the Gibbs sampler needs to run for 533 seconds on the same problem.

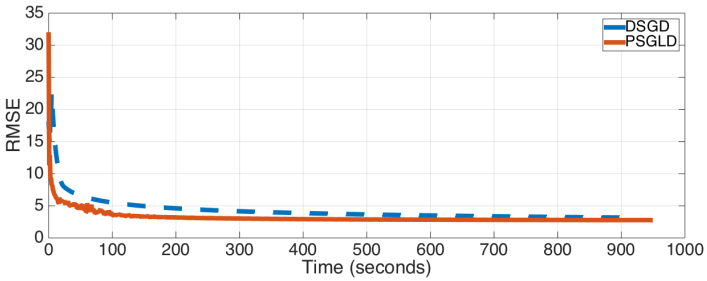


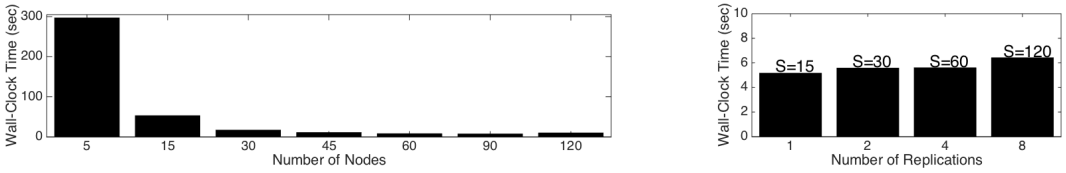
Figure 8.9. RMSE values on MovieLens 10M dataset.

8.2.2. Distributed Setting

In this section, we will focus on the implementation of PSGLD in a distributed setting, where each block of X_1 might reside at a different node. We will consider the distributed architecture that we described in Section 6.3. We also use a similar communication mechanism that we explained in Section 6.3, except that we do not have the memory matrices Ξ and Ψ in this method.

In our first distributed-setting experiment, our goal is to contrast the speed of our sampling algorithm to a distributed optimization algorithm. Clearly, the goals of both computations are different (a sampler does not solve an optimization problem unless techniques such as simulated annealing is being used), yet monitoring the root mean squared error (RMSE) between X_1 and $\hat{X}_1$ throughout the iterations provides a qualitative picture about the convergence behavior of the algorithms. Figure 8.9 shows the RMSE values of PSGLD and DSGD [12] for 1000 iterations with $S = 15$ on MovieLens10M. We observe a very similar convergence behavior and running times for both methods. The results indicate that, PSGLD makes Bayesian inference possible for factorization models even for large datasets by generating samples from the Bayesian posterior, while at the same time being as fast as the distributed optimization algorithms.

In our last set of experiments, we demonstrate the scalability of PSGLD. Firstly, we differ the number of nodes from 5 to 120 and generate 100 samples in each setting.



(a) (b)

Figure 8.10. Scalability of PSGLD. a) The size of the data is kept fixed, the number of nodes is increased b) The size of the data and the number of nodes are increased proportionally.

Figure 8.10a shows the running times of PSGLD for different number of nodes. The results show that, the running time reduces almost quadratically as we increase the number of nodes until $S = 90$. For $S = 120$, the communication cost dominates and the running time increases.

Finally, in order to illustrate how PSGLD scales with the size of the data, we increase the size of X_1 while increasing the number of nodes accordingly, as we previously demonstrated in Section 6.3. We select $X_1^{S=15}$ as MovieLens10M, and construct $X_1^{S=30}$, $X_1^{S=60}$, and $X_1^{S=120}$ in the same way as described before. In this setting, the ultimate dataset becomes of size 683584×4580288 with 640 million non-zero entries and the number of nodes becomes 120. Figure 8.10b shows the running times of PSGLD with $T = 10$ epochs for increasing data sizes and number of nodes. The results show that, even though we increase the size of the data 64 folds, the running time of PSGLD remains nearly constant provided we can increase the number of nodes proportionally.

9. CONCLUSION AND FUTURE WORK

Within last decade, massive amounts of data have been continuously produced and the cost of storing these data have gotten cheaper everyday. There are two major challenges that have risen with large amounts of data. The first one is the computational challenge, and the second one is handling data heterogeneity.

Coupled matrix and tensor factorization models are useful for analyzing large-scale and/or heterogeneous data and have been shown to be useful in various domains, in which heterogeneous information from diverse sources are available and need to be combined for arriving at useful predictions. These models become advantageous in many applications since they provide a good modeling accuracy versus practicality trade off.

In this thesis, we have focused on developing inference methods for coupled tensor factorization models. We have addressed several challenging problems, including:

(i) Handling arbitrary model topologies: To be able to model real world data sets that may consist of several tensors and require custom models, we have rigorously developed the tensor factorization notation of [35], that aims to cover all possible model topologies and coupled factorization models.

(ii) Maximum likelihood and a-posteriori estimation of the latent factors: We have developed two novel, inherently parallel optimization algorithms for maximum and a-posteriori estimation of the latent variables. As we have demonstrated in our experiments, our methods outperform the state-of-the-art in terms of computational complexity and/or estimation quality in large-scale, distributed applications.

(iii) Estimation of the dispersions and the divergences: For jointly inferring the dispersions and the divergence functions, we have developed two novel methods. In the first method, we have focused on Tweedie compound Poisson models. The second method is much more general and enables joint inference in the whole

Tweedie family. In our experiments, we have showed that joint estimation of the dispersions and the divergences can cause dramatic performance improvements, especially when the number of observed tensors is large.

(iv) Full Bayesian inference via MCMC: We have developed three novel MCMC methods for making full Bayesian inference. For moderate-sized data, we have developed 'block' and 'collapsed' Gibbs samplers. For large-scale applications, we have developed an inherently parallel stochastic gradient-based MCMC method, that can gracefully scale up to large-scale, distributed problems. Our experiments showed that, we can enable Bayesian inference even for large-scale distributed factorization problems, where conventional approaches suffer from high computational complexity.

(v) Alpha-stable matrix factorization: We have developed a novel factorization model with α-stable observations, that is particularly suited for impulsive or corrupted data that appear in several domains such as audio processing. We have also developed an MCMC method for making in this model.

(vi) Applications: We have developed novel factorization models that aim to solve several challenging problems, such as

- Audio restoration
- Musical audio source separation
- Lyrics prediction
- Speech enhancement
- Link prediction

As we have demonstrated in our experiments, the proposed methods are very powerful and flexible. However, they can be further improved in several aspects. Our current future directions are listed as follows:

(i) Asynchronous inference: Distributed and parallel inference schemes are crucial for processing large-scale heterogeneous data. One drawback of the current techniques is the synchronization problem; that is in a distributed setting, each processor updates some part of the parameters and it needs to send the updated part to the other processors, which can incur a heavy communication cost when the

number of nodes is large (see Figure 6.12). Recently, asynchronous optimization methods have been proposed [11, 13]. In these methods, the processors are synchronized only once in a while, instead of being synchronized at each iteration. These techniques have not been applied to coupled factorizations yet and seem very promising for the distributed settings.

(ii) Incorporation of non-identity link functions: Link functions are one of the fundamental components of generalized linear models [111]. The link function establishes the relation between the linear predictor and the mean of the response distribution. For instance, in the GCTF model we implicitly assume $X_\nu \approx f^{-1}(\hat{X}_\nu)$, where $f(\cdot)$ is the link function that is simply selected as identity function $f(x) = x$. The identity link function can be limiting for some cases especially when the support of the distribution is not the same type of data as the parameter being predicted, for example Bernoulli, binomial, categorical and multinomial distributions. The usage of non-identity link functions have not been extensively explored for coupled factorizations and might yield interesting hybrid models that could also take advantage of deep learning methodologies.

(iii) Privacy preserving factorizations: Differentially private algorithms aim to minimize the probability of the data samples being uncovered, while at the same time minimizing the accuracy loss due to privatization. Preserving the privacy in coupled tensor factorizations can be crucial in certain cases, especially when the observed tensors are kept in different sites and the elements of these tensors must not be identified by the sites other than their own. Even though developing differentially private optimization algorithms is not straightforward, recently it has been shown that stochastic MCMC inference can be made differently private with minor algorithmic modifications [116]. Therefore, a natural next step for our work is to develop differentially private Bayesian inference algorithms for coupled tensor factorization models by building up on PSGLD.

APPENDIX A: PROBABILITY DENSITY AND MASS FUNCTIONS

(i) Binomial Distribution:

$$\mathcal{BI}(s; x, p) = \frac{x!}{s!(x-s)!} p^s (1-p)^{(x-s)}. \tag{A.1}$$

(ii) Multinomial Distribution:

$$\mathcal{M}(\mathbf{s}; x, \mathbf{p}) = \delta(x - \sum_i s_i) x! \prod_{i=1}^{I} \frac{p_i^{s_i}}{s_i!} \tag{A.2}$$

where $\mathbf{s} = \{s_1, \ldots, s_I\}$ and $\mathbf{p} = \{p_1, \ldots, p_I\}$.

(iii) Poisson Distribution:

$$\mathcal{PO}(x; \lambda) = \frac{\lambda^x \exp(-\lambda)}{\Gamma(x+1)} \tag{A.3}$$

(iv) Exponential Distribution:

$$\mathcal{E}(x; a) = a \exp(-ax) \tag{A.4}$$

(v) Gamma Distribution:

$$\mathcal{G}(x; a, b) = x^{a-1} b^a \frac{\exp(-bx)}{\Gamma(a)} \tag{A.5}$$

(vi) Inverse gamma Distribution:

$$\mathcal{IG}(x; a, b) = \frac{x^{-(a+1)} b^a}{\Gamma(a)} \exp(-b/x) \tag{A.6}$$

(vii) Generalized gamma distribution:

$$\mathcal{GG}(x; a, b, c) = \frac{|c|}{\Gamma(a) b^{ca}} x^{ca-1} \exp\Big(-\big(\frac{x}{b}\big)^{c}\Big). \tag{A.7}$$

(viii) Uniform distribution:

$$\mathcal{U}(x; [a, b]) = \frac{1}{b-a}. \tag{A.8}$$

APPENDIX B: EXPONENTIAL DISPERSION MODELS AND THE TWEEDIE FAMILY

An exponential dispersion model (EDM) can be defined by a two parameter density as follows [65]:

$$\mathbb{P}(x;\theta,\phi) = h(x,\phi)\exp\left\{\frac{1}{\phi}\left(\theta x - \kappa(\theta)\right)\right\} \tag{B.1}$$

where θ is the canonical parameter, ϕ is the dispersion parameter and κ is the cumulant (log-partition) function ensuring normalization. Here, $h(x,\phi)$ is the base measure and is independent of the canonical parameter.

EDMs are a studied in particular as the response distribution of the generalized linear models [111]. For an EDM, we can verify that the mean $\hat{x}$ and the variance $\mathrm{Var}[x]$ are obtained directly by differentiating $\kappa(\cdot)$:

$$\kappa'(\theta) = \langle x\rangle_{p(x;\theta,\phi)} \equiv \hat{x}, \quad \kappa''(\theta) = \frac{1}{\phi}\mathrm{Var}[x] \equiv v(\hat{x}).$$

Here $v(\hat{x})$ is also known as the variance function [65, 117].

In certain cases, we need the cumulant generating function (CGM) of EDMs. The CGF of an EDM is derived as follows

$$\begin{aligned}
K(s;\theta,\phi) &= \log\left\langle \exp(sx)\right\rangle_{p(x;\theta,\phi)} \\
&= \log\int a(x,\phi)\exp(\frac{1}{\phi}(x\theta-\kappa(\theta))+sx)dx \\
&= \log\int a(x,\phi)\exp(\frac{1}{\phi}(x\theta-\kappa(\theta)+sx\phi))dx \\
&= \log\int a(x,\phi)\exp(\frac{1}{\phi}(x(s\phi+\theta)-\kappa(\theta)))dx
\end{aligned}$$

$$
\begin{aligned}
&= \log \int a(x,\phi) \exp(\frac{1}{\phi}(x(s\phi+\theta) - \kappa(s\phi+\theta))) \exp(\frac{1}{\phi}(\kappa(s\phi+\theta) - \kappa(\theta)))dx \\
&= \log \left(\exp(\frac{1}{\phi}(\kappa(s\phi+\theta) - \kappa(\theta))) \underbrace{\int a(x,\phi) \exp(\frac{1}{\phi}(x(s\phi+\theta) - \kappa(s\phi+\theta)))dx}_{1} \right) \\
&= \frac{1}{\phi}(\kappa(s\phi+\theta) - \kappa(\theta))
\end{aligned} \tag{B.2}
$$

The cumulant generating functions are unique for each distribution and we can see that for an EDM the cumulant generating function depends only on the dispersion parameter and the log-partition function κ. Therefore, we only need to find the log-partition function for an EDM in order to identify the distribution.

In this thesis, we focus on a particular EDM, namely The Tweedie family $\mathcal{TW}_p(x; \hat{x}, \phi)$. Tweedie distributions specify the variance function as $v(\hat{x}) = \hat{x}^p$ [65]. The variance function is related to the p'th power of the mean, therefore it is called a power variance function. Note that this choice directly dictates the form of $\hat{x}$ and $\kappa(\theta)$ that can be solved as

$$
\hat{x}(\theta) = \begin{cases} \frac{1}{2-p}\left((1-p)\theta\right)^{\frac{1}{1-p}} & p \neq 1 \\ \exp(\theta) & p = 1 \end{cases} \tag{B.3}
$$

$$
\kappa(\theta) = \begin{cases} \frac{1}{2-p}\left((1-p)\theta\right)^{\frac{2-p}{1-p}} & p \neq 1, 2 \\ -\log(-\theta) & p = 2 \\ \exp(\theta) & p = 1 \end{cases} . \tag{B.4}
$$

Here, different choices for p yield well-known important distributions such as the Gaussian ($p = 0$), Poisson ($p = 1$), compound Poisson ($1 < p < 2$), Gamma ($p = 2$) and inverse Gaussian ($p = 3$) distributions. Excluding the interval $0 < p < 1$ for which no EDM exists, for all other values of p not mentioned above, one obtains Tweedie stable distributions [65].

For $p \in \{0, 1, 2, 3\}$ the densities are given as follows:

$$\mathcal{TW}_0(x; \hat{x}, \phi) = (2\pi\phi)^{-\frac{1}{2}} \exp\left(-\frac{1}{\phi}\frac{(x-\hat{x})^2}{2}\right) \tag{B.5}$$

$$\mathcal{TW}_1(x; \hat{x}, \phi) = \frac{(\bar{x}/\phi)^{\frac{\bar{x}}{\phi}}}{e^{\frac{\bar{x}}{\phi}}\Gamma(\frac{\bar{x}}{\phi}+1)} \exp\left(-\frac{1}{\phi}(\bar{x}\log\frac{\bar{x}}{\bar{\hat{x}}} - \bar{x} + \bar{\hat{x}})\right) \tag{B.6}$$

$$\mathcal{TW}_2(x; \hat{x}, \phi) = \frac{1}{\Gamma(\frac{1}{\phi})(e\phi)^{\frac{1}{\phi}}x} \exp\left(-\frac{1}{\phi}(\frac{x}{\hat{x}} - \log\frac{x}{\hat{x}} - 1)\right) \tag{B.7}$$

$$\mathcal{TW}_3(x; \hat{x}, \phi) = (2\pi x^3\phi)^{-\frac{1}{2}} \exp\left(-\frac{1}{\phi}\frac{(x-\hat{x})^2}{2x\hat{x}^2}\right). \tag{B.8}$$

Note that, the Poisson distribution in its well-known form, is an exponential dispersion model with unitary dispersion ($\phi = 1$). This distribution is called over-dispersed ($\phi > 1$) or under-dispersed ($\phi < 1$) when the nominal variance is not sufficient to determine the variance of the observations [111]. When we introduce a dispersion parameter to the Poisson distribution, the domain of the probability distribution is re-defined on the integer multiples of ϕ: $\bar{x} \in \{0, \phi, 2\phi, 3\phi, \dots\}$. This can be interpreted as the data are scaled by ϕ.

For the remaining cases of p, the probability density functions cannot be written in closed-form analytical forms. However, they can be expressed as infinite series that is defined as follows: [65]

$$\mathcal{TW}_p(x; \hat{x}, \phi) = \frac{1}{x\xi_p}\left(\sum_{k=1}^{\infty} V_k\right) \exp\left\{\frac{1}{\phi}\left(\frac{\hat{x}^{1-p}x}{1-p} - \frac{\hat{x}^{2-p}}{2-p}\right)\right\} \tag{B.9}$$

and $\xi_p = 1$ for $p \in (1, 2)$ and $\xi_p = \pi$ otherwise.

The Tweedie density with $p \in (1, 2)$ coincides with the compound Poisson distribution [65]. The compound Poisson distribution has a support for continuous positive data and a discrete probability mass at zero. For $x = 0$, the density function is defined as $\mathcal{TW}_p(x; \cdot) = \exp(\hat{x}^{2-p}/(\phi(p-2)))$ and for $x > 0$, it follows the form of Equation B.9,

where the terms V_k for this distribution is defined as follows:

$$V_k = \frac{x^{-k\alpha}(p-1)^{k\alpha}\phi^{k(\alpha-1)}}{(2-p)^k\Gamma(k+1)\Gamma(-k\alpha)} \tag{B.10}$$

where $\alpha = (2-p)/(1-p)$.

The cases $p < 0$ and $p > 2$ of the Tweedie class correspond to Tweedie stable distributions. These distributions have many properties similar to stable distributions [68] and they coincide with the stable distributions in certain cases [65]. Tweedie stable models are heavy-tailed distributions and they are left-skewed for $p < 0$ and right-skewed for $p > 2$. The Tweedie stable models with $p > 2$ can be useful for many applications, including audio signal processing [118] and computer networks [119]. The Tweedie stable models with $p < 0$ can be used for risk modeling [120], however their applications on factorization models are limited. For the Tweedie models with $p < 0$ and $p > 2$, the terms V_k are defined as follows:

$$V_k = \frac{\Gamma(1+\frac{k}{\alpha})\phi^{\frac{k}{p-2}}(-1)^k\sin(\frac{k\pi}{\alpha})}{\Gamma(k+1)(1-p)^k(2-p)^{-\frac{k}{\alpha}}x^{-k}},$$

$$V_k = \frac{\Gamma(1+\alpha k)\phi^{k(1-\alpha)}(-1)^k\sin(-k\pi\alpha)}{\Gamma(k+1)(p-1)^{-\alpha k}(p-2)^k x^{\alpha k}}.$$

APPENDIX C: CONVERGENCE PROOF

We reserve this appendix to show that Algorithm HAMSI converges. Our demonstration follows a similar construction as in Solodov [121]. To simplify our exposition, we define

$$\nabla f_s(z) \equiv \sum_{b=1}^{B_s} \nabla f_{\Omega_{s,b}}(z_b).$$

Our subsequent discussion is given under the following assumptions:

A.1 The twice differentiable objective function f is bounded below.

A.2 The Hessian matrices for the component functions are uniformly bounded at every outer iteration t. That is, for every s and t, we have

$$\|\nabla^2 f_s(z^{(t)})\| \leq L,$$

where L is the well-known Lipschitz constant.

A.3 The eigenvalues of the approximation matrices $H^{(t)}$ are bounded so that

$$(U + \rho^{(t)})^{-1} = U_t \leq \|(H^{(t)} + \rho^{(t)})^{-1}\| \leq M_t = (M + \rho^{(t)})^{-1}$$

holds. Here, U_t and M_t are known constants with $0 < M \leq U$.

A.4 The gradient norms are uniformly bounded at every outer iteration t; i.e., for every s we have

$$\|\nabla f_s(z^{(t)})\| \leq C,$$

where C is a known constant.

Lemma C.1. *At outer iteration t and inner iteration s of Algorithm HAMSI, we have*

$$\delta_s = \|\nabla f_s(\xi^{(s)}) - \nabla f_s(z^{(t)})\| \leq LM_t \sum_{j=1}^{s-1} (1 + LM_t)^{s-1-j} \|\nabla f_j(z^{(t)})\|. \tag{C.1}$$

Proof. We use induction to prove this result. Clearly $\delta_1 = \|\nabla f_1(\xi^{(1)}) - \nabla f_1(z^{(t)})\| = 0$. For $s = 2$, we have

$$\begin{aligned}
\delta_2 &= \|\nabla f_2(\xi^{(2)}) - \nabla f_2(z^{(t)})\| \\
&\leq L\|\xi^{(2)} - z^{(t)}\| = L\|\xi^{(2)} - \xi^{(1)}\| \\
&\leq L\|(H^{(t)} + \rho^{(t)} I)^{-1}\| \|\nabla f_1(\xi^{(1)})\| \\
&\leq LM_t \|\nabla f_1(\xi^{(1)})\| = LM_t \|\nabla f_1(z^{(t)})\|.
\end{aligned}$$

Now suppose that (C.1) holds for $k = 1, \ldots, p-1$ and consider

$$\begin{aligned}
\delta_p &= \|\nabla f_p(\xi^{(p)}) - \nabla f_p(z^{(t)})\| \\
&\leq L\|\xi^{(p)} - \xi^{(1)}\| = L \textstyle\sum_{s=1}^{p-1} \|\xi^{(s+1)} - \xi^{(s)}\| \\
&\leq LM_t \textstyle\sum_{s=1}^{p-1} \|\nabla f_s(\xi^{(s)})\|.
\end{aligned}$$

Note again by the induction hypothesis that

$$\begin{aligned}
\|\nabla f_s(\xi^{(s)})\| &= \|\nabla f_s(z^{(t)}) + \nabla f_s(\xi^{(s)}) - \nabla f_s(z^{(t)})\| \\
&\leq \|\nabla f_s(z^{(t)})\| + \delta_s \\
&\leq \|\nabla f_s(z^{(t)})\| + LM_t \textstyle\sum_{j=1}^{s-1} (1 + LM_t)^{s-1-j} \|\nabla f_j(z^{(t)})\|.
\end{aligned}$$

Then,

$$\begin{aligned}
\delta_p &\leq LM_t \textstyle\sum_{s=1}^{p-1} \left(\|\nabla f_s(z^{(t)})\| + LM_t \sum_{j=1}^{s-1} (1 + LM_t)^{s-1-j} \|\nabla f_j(z^{(t)})\| \right) \\
&= LM_t \textstyle\sum_{j=1}^{p-1} \left(1 + LM_t \sum_{s=0}^{p-1-j-1} (1 + LM_t)^s \right) \|\nabla f_j(z^{(t)})\| \\
&= LM_t \textstyle\sum_{j=1}^{p-1} (1 + LM_t)^{p-1-j} \|\nabla f_j(z^{(t)})\|.
\end{aligned}$$

This completes the proof. □

Next lemma gives a bound on the difference between two consecutive outer iterations.

Lemma C.2. *Given outer iterations t and $t+1$ of Algorithm HAMSI, we have*

$$z^{(t+1)} - z^{(t)} = \Delta_t - (H^{(t)} + \rho^{(t)} I)^{-1} \nabla f(z^{(t)}),$$

where

$$\Delta_t \equiv (H^{(t)} + \rho^{(t)} I)^{-1} \sum_{s=1}^{S} (\nabla f_s(z^{(t)}) - \nabla f_s(\xi^{(s)}))$$

and

$$\|\Delta_t\| \leq M_t L C S^2 (1 + L M^{-1})^S.$$

Proof. At outer iteration $t+1$, we have

$$\begin{aligned} z^{(t+1)} &= z^{(t)} - \textstyle\sum_{s=1}^{S} (H^{(t)} + \rho^{(t)} I)^{-1} \nabla f_s(\xi^{(s)}) \\ &= z^{(t)} - (H^{(t)} + \rho^{(t)} I)^{-1} \nabla f(z^{(t)}) \\ &\quad + (H^{(t)} + \rho^{(t)} I)^{-1} \textstyle\sum_{s=1}^{S} (\nabla f_s(z^{(t)}) - \nabla f_s(\xi^{(s)})). \end{aligned}$$

This shows that

$$z^{(t+1)} - z^{(t)} = \Delta_t - (H^{(t)} + \rho^{(t)} I)^{-1} \nabla f(z^{(t)}).$$

Using now (C.1), we obtain

$$\begin{aligned} \|\Delta_t\| &\leq M_t \| \textstyle\sum_{s=1}^{S} (\nabla f_s(z^{(t)}) - \nabla f_s(\xi^{(s)}))\| \leq M_t \textstyle\sum_{s=1}^{S} \delta_s \\ &\leq M_t^2 L \textstyle\sum_{s=1}^{S} \sum_{j=1}^{s-1} (1 + L M_t)^{s-1-j} \|\nabla f_j(z^{(t)})\| \\ &\leq M_t^2 L C S^2 (1 + L M^{-1})^S, \end{aligned}$$

since $1 + LM_t \leq 1 + L(M + \rho^{(t)})^{-1} \leq 1 + LM^{-1}$. □

Theorem C.3. *Consider the iterates $z^{(t)}$ of Algorithm HAMSI. Suppose that $\rho^{(t)} \to \infty$ as $t \to \infty$, and $\rho^{(t)}$ increases slowly enough so that $\rho^{(t)} = \mathcal{O}(\|\nabla f(z^{(t)})\|^{-\sigma})$ with $\sigma \in (0.5, 1)$ for t large. Then*

$$\liminf_{t\to\infty} \|\nabla f(z^{(t)})\| = 0.$$

Proof. For any twice differentiable function, we have

$$f(z^{(t+1)}) - f(z^{(t)}) \leq \nabla f(z^{(t)})^T (z^{(t+1)} - z^{(t)}) + \frac{L}{2}\|z^{(t+1)} - z^{(t)}\|^2.$$

Using now Lemma C.2, we obtain

$$\begin{aligned} f(z^{(t+1)}) - f(z^{(t)}) &\leq \nabla f(z^{(t)})^T(-(H^{(t)} + \rho^{(t)}I)^{-1}\nabla f(z^{(t)}) + \Delta_t) \\ &\quad + \tfrac{L}{2}\| - (H^{(t)} + \rho^{(t)}I)^{-1}\nabla f(z^{(t)}) + \Delta_t\|^2 \\ &\leq -U_t\|\nabla f(z^{(t)})\|^2 + \|\Delta_t\|\|\nabla f(z^{(t)})\| + \tfrac{L}{2}M_t^2\|\nabla f(z^{(t)})\|^2 \\ &\quad + \tfrac{L}{2}\|\Delta_t\|^2 + LM_t\|\nabla f(z^{(t)})\|\|\Delta_t\| \\ &\leq -U_t\|\nabla f(z^{(t)})\|^2 + M_t^2 B\|\nabla f(z^{(t)})\| + \tfrac{L}{2}M_t^2\|\nabla f(z^{(t)})\|^2 \\ &\quad + \tfrac{L}{2}M_t^4 B^2 + LM_t^3 B\|\nabla f(z^{(t)})\|, \end{aligned}$$

where $B \equiv LCS^2(1 + LM^{-1})^S$. The above inequality further simplifies to

$$f(z^{(t+1)}) - f(z^{(t)}) \leq (-U_t + \frac{L}{2}M_t^2)\|\nabla f(z^{(t)})\|^2 + M_t^2 B(1 + LM_t)\|\nabla f(z^{(t)})\| + \frac{L}{2}M_t^4 B^2.$$

Since $M_t = \mathcal{O}(\frac{1}{\rho^{(t)}})$, $U_t = \mathcal{O}(\frac{1}{\rho^{(t)}})$, and $\rho^{(t)} = o(\|\nabla f(z^{(t)})\|^{-1})$ for t large enough, we have

$$\begin{aligned} f(z^{(t+1)}) - f(z^{(t)}) &\leq (-U_t + \tfrac{L}{2}M_t^2)\|\nabla f(z^{(t)})\|^2 + o(\|\nabla f(z^{(t)})\|^2), \\ &\leq \left(-\mathcal{O}\left(\tfrac{1}{\rho^{(t)}}\right) + \tfrac{L}{2}\mathcal{O}\left(\tfrac{1}{(\rho^{(t)})^2}\right)\right)\|\nabla f(z^{(t)})\|^2 + o(\|\nabla f(z^{(t)})\|^2). \end{aligned}$$

Suppose for contradiction that

$$\liminf_{t\to\infty} \|\nabla f(z^{(t)})\| > 0.$$

Then, for some $\epsilon > 0$, there exists $\bar{t}$ such that

$$f(z^{(t+1)}) - f(z^{(t)}) \leq -\epsilon \text{ for all } t \geq \bar{t}.$$

Therefore, for any $\hat{t} > \bar{t}$, we have

$$f(z^{(\hat{t})}) - f(z^{(\bar{t})}) = \sum_{t=\bar{t}}^{\hat{t}-1} f(z^{(t+1)}) - f(z^{(t)}) \leq -\epsilon(\hat{t} - \bar{t}).$$

Note that in the right-hand-side of the last inequality, the difference $\hat{t} - \bar{t}$ becomes arbitrarily large when $\hat{t} \to \infty$. However, f is bounded below due to our assumption. Hence, we arrive at a contradiction. This implies the desired result

$$\liminf_{t\to\infty} \|\nabla f(z^{(t)})\| = 0.$$

□

APPENDIX D: NUMERICAL APPROXIMATION

D.1. Evaluating the Gradient of the Dispersion in Tweedie Models

The gradient of the cost function with respect to ϕ requires two infinite summations to be computed. Here, we make use of the numerical methods proposed in [82] that locates the indices k where the terms V_k make the major contribution to the sum. In this section we describe the method for the case when $p < 0$.

In order to locate the region of the contributive terms, firstly the index of the most contributive term $k^\star$ is approximately determined. We define an envelope V_k^{env} for V_k by discarding the terms $(-1)^k$ and $\sin(\cdot)$, where we have $V_k^{env} \geq |V_k(\cdot)|$ for all k. Then we approximate the gamma functions in the envelope term by using Stirling's formula:

$$\log \Gamma(x+1) \approx (x+\frac{1}{2}) \log x - x + \frac{1}{2} \log 2\pi \tag{D.1}$$

For $p < 0$ we obtain:

$$\log V_k^{env} \approx k \log(\frac{x_i \phi^{\frac{1}{p-2}} (2-p)^{\frac{1}{\alpha}}}{(1-p)}) + (\frac{k}{p-2}) \log k - \frac{k}{\alpha} \log(\alpha) + (\frac{k}{2-p}) - \frac{1}{2} \log(\alpha).$$

This equation allows us to find an approximate mode $k^\star$ by solving

$$(d \log V_k^{env})/(dk) = 0$$

as if k is continuous. The lower and the upper bounds for k is found by evaluating the terms that are at either side of $k^\star$ until their contributions are negligible. For $p < 0$ we obtain the mode at:

$$k^\star = x_i^{2-p}/((1-p)\phi). \tag{D.2}$$

By following a similar approach, the mode for $1 < p < 2$ and $p > 2$ can be obtained as

$$k^\star = x_i^{2-p}/((2-p)\phi) \quad \text{and} \quad k^\star = x_i^{2-p}/((p-2)\phi) \quad \text{respectively [82].}$$

D.2. Evaluating Alpha-Stable Densities

In order to evaluate the stable densities, we follow the same approach as in the previous section. For evaluating the stable densities, we make use of the power series representation of the stable distribution [69, 90]:

$$p(x) = \frac{1}{\pi x} \sum_{k=1}^{\infty} V_k \tag{D.3}$$

where

$$V_k = \begin{cases} \frac{\Gamma(k\alpha+1)}{\Gamma(k+1)}(-1)^k (x^{-\alpha})^k \sin\frac{k\pi}{2}(\gamma-\alpha) & 0 < \alpha < 1 \\ \frac{\Gamma(k/\alpha+1)}{\Gamma(k+1)}(-1)^k x^k \sin\frac{k\pi}{2\alpha}(\gamma-\alpha) & 1 < \alpha < 2 \end{cases}$$

where the scale parameter is set to $a = (1 + \mu^2 \tan^2(\pi\alpha/2))^{1/(2\alpha)}$. In this section we describe the method for the case when $0 < \alpha < 1$. It is straightforward to extend the method for $1 < \alpha < 2$.

For $0 < \alpha < 1$, the envelope is given as follows:

$$\log V_k^{env} \approx (k\alpha - k)\log k + k\alpha \log\frac{\alpha}{x} + k - k\alpha$$

Accordingly, we obtain the mode at:

$$k^\star = \exp(\frac{\alpha \log\frac{\alpha}{x}}{1-\alpha}). \tag{D.4}$$

REFERENCES

1. Scott, J., *Social Network Analysis: A Handbook*, Sage, London, 2012.

2. O'Driscoll, A., J. Daugelaite and R. D. Sleator, "'Big Data', Hadoop and Cloud Computing in Genomics", *Journal of Biomedical Informatics*, Vol. 46, No. 5, pp. 774–781, 2013.

3. Chen, H., R. H. Chiang and V. C. Storey, "Business Intelligence and Analytics: From Big Data to Big Impact.", *MIS quarterly*, Vol. 36, No. 4, pp. 1165–1188, 2012.

4. Bertin-Mahieux, T., D. P. Ellis, B. Whitman and P. Lamere, "The Million Song Dataset", *ISMIR 2011: Proceedings of the 12th International Society for Music Information Retrieval Conference, October 24-28, 2011, Miami, Florida*, pp. 591–596, University of Miami, 2011.

5. Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "Imagenet: A Large-Scale Hierarchical Image Database", *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255, IEEE, 2009.

6. Linden, G., B. Smith and J. York, "Amazon.com Recommendations: Item-to-item Collaborative Filtering", *IEEE Internet Computing*, Vol. 7, No. 1, pp. 76–80, 2003.

7. Fan, J., F. Han and H. Liu, "Challenges of Big Data Analysis", *National Science Review*, Vol. 1, No. 2, pp. 293–314, 2014.

8. Bertsekas, D. P., "Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey", *Optimization for Machine Learning*, Vol. 2010, pp. 1–38, 2011.

9. Bottou, L., "Online Algorithms and Stochastic Approximations", *Online Learning and Neural Networks*, Cambridge University Press, 1998.

10. Murata, N., "A Statistical Study of On-line Learning", *Online Learning and Neural Networks*, pp. 63–92, 1998.

11. Recht, B., C. Re, S. Wright and F. Niu, "Hogwild: A Lock-free Approach to Parallelizing Stochastic Gradient Descent", *Advances in Neural Information Processing Systems*, pp. 693–701, 2011.

12. Gemulla, R., E. Nijkamp, P. J. Haas and Y. Sismanis, "Large-scale Matrix Factorization with Distributed Stochastic Gradient Descent", *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 69–77, ACM, 2011.

13. Recht, B. and C. Ré, "Parallel Stochastic Gradient Algorithms for Large-scale Matrix Completion", *Mathematical Programming Computation*, Vol. 5, No. 2, pp. 201–226, 2013.

14. Gürbüzbalaban, M., A. Ozdaglar and P. Parrilo, "A Globally Convergent Incremental Newton Method", *arXiv preprint arXiv:1410.5284*, 2014, `http://arxiv.org/abs/1410.5284`, [Accessed August 2015].

15. Byrd, R. H., S. Hansen, J. Nocedal and Y. Singer, "A Stochastic Quasi-Newton Method for Large-scale Optimization", *arXiv preprint arXiv:1401.7020*, 2014, `http://arxiv.org/abs/1401.7020`, [Accessed August 2015].

16. Schraudolph, N. N., J. Yu and S. Gunter, "A Stochastic Quasi-Newton Method for Online Convex Optimization", *Proceedings of the 11th International Conference Artificial Intelligence and Statistics (AISTATS)*, pp. 433–440, 2007.

17. Hoffman, M. D., D. M. Blei, C. Wang and J. Paisley, "Stochastic Variational Inference", *The Journal of Machine Learning Research*, Vol. 14, No. 1, pp. 1303–

1347, 2013.

18. Wang, C. and D. M. Blei, "Truncation-free Online Variational Inference for Bayesian Nonparametric Models", *Advances in Neural Information Processing Systems*, pp. 413–421, 2012.

19. Ranganath, R., C. Wang, B. David and E. Xing, "An Adaptive Learning Rate for Stochastic Variational Inference", *Proceedings of The 30th International Conference on Machine Learning*, pp. 298–306, 2013.

20. Johnson, M., J. Saunderson and A. Willsky, "Analyzing Hogwild Parallel Gaussian Gibbs Sampling", *Advances in Neural Information Processing Systems*, pp. 2715–2723, 2013.

21. Bardenet, R., A. Doucet and C. Holmes, "Towards Scaling Up Markov Chain Monte Carlo: An Adaptive Subsampling Approach", *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 405–413, 2014.

22. Welling, M. and Y. W. Teh, "Bayesian Learning via Stochastic Gradient Langevin Dynamics", *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 681–688, 2011.

23. Ahn, S., A. Korattikara and M. Welling, "Bayesian Posterior Sampling via Stochastic Gradient Fisher Scoring", *International Conference on Machine Learning*, 2012.

24. Chen, T., E. Fox and C. Guestrin, "Stochastic Gradient Hamiltonian Monte Carlo", *Proc. International Conference on Machine Learning*, June 2014.

25. Patterson, S. and Y. W. Teh, "Stochastic Gradient Riemannian Langevin Dynamics on the Probability Simplex", *Advances in Neural Information Processing Systems*, 2013.

26. Ahn, S., B. Shahbaba and M. Welling, "Distributed Stochastic Gradient MCMC", *International Conference on Machine Learning*, 2014.

27. Pan, S. J. and Q. Yang, "A Survey on Transfer Learning", *IEEE Transactions on Knowledge and Data Engineering*, Vol. 22, No. 10, pp. 1345–1359, 2010.

28. Evgeniou, A. and M. Pontil, "Multi-task Feature Learning", *Advances in Neural Information Processing Systems*, Vol. 19, p. 41, 2007.

29. Raina, R., A. Battle, H. Lee, B. Packer and A. Y. Ng, "Self-taught Learning: Transfer Learning from Unlabeled Data", *Proceedings of the 24th International Conference on Machine Learning*, 2007.

30. Gönen, M. and E. Alpaydın, "Multiple Kernel Learning Algorithms", *Journal of Machine Learning Research*, Vol. 12, No. Jul, pp. 2211–2268, 2011.

31. Gönen, M. and E. Alpaydin, "Localized Multiple Kernel Learning", *Proceedings of the 25th International Conference on Machine Learning*, pp. 352–359, ACM, 2008.

32. Bucak, S. S., R. Jin and A. K. Jain, "Multiple Kernel Learning for Visual Object Recognition: A Review", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 36, No. 7, pp. 1354–1369, 2014.

33. Yılmaz, Y. K., *Generalized Tensor Factorization*, Ph.D. Thesis, Bogazici University, 2012.

34. Acar, E., T. G. Kolda and D. M. Dunlavy, "All-at-once Optimization for Coupled Matrix and Tensor Factorizations", *MLG'11: Proceedings of Mining and Learning with Graphs*, 2011.

35. Yılmaz, Y. K., A. T. Cemgil and U. Şimşekli, "Generalised Coupled Tensor Factorisation", *Advances in Neural Information Processing Systems*, pp. 2151–2159,

2011.

36. Yoo, J., M. Kim, K. Kang and S. Choi, "Nonnegative Matrix Partial Co-factorization for Drum Source Separation", *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pp. 1942–1945, 2010.

37. Le Magoarou, L., A. Ozerov and N. Q. Duong, "Text-informed Audio Source Separation Using Nonnegative Matrix Partial Co-factorization", *IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2013.

38. Şimşekli, U. and A. T. Cemgil, "Score Guided Musical Source Separation Using Generalized Coupled Tensor Factorization", *European Signal Processing Conference*, pp. 2639–2643, 2012.

39. Barker, T., T. Virtanen and O. Delhomme, "Ultrasound-Coupled Semi-Supervised Nonnegative Matrix Factorisation for Speech Enhancement", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2129–2133, 2014.

40. Wilderjans, T., E. Ceulemans, I. Van Mechelen and R. van den Berg, "Simultaneous Analysis Of Coupled Data Matrices Subject To Different Amounts Of Noise", *British Journal of Mathematical and Statistical Psychology*, Vol. 64, pp. 277–90, 2011.

41. Levin, J., "Simultaneous Factor Analysis Of Several Gramian Matrices", *Psychometrika*, Vol. 31, No. 3, pp. 413–419, 1966.

42. Acar, E., G. Gurdeniz, M. A. Rasmussen, D. Rago, L. O. Dragsted and R. Bro, "Coupled Matrix Factorization with Sparse Factors to Identify Potential Biomarkers in Metabolomics.", *International Journal of Knowledge Discovery in Bioinformatics*, Vol. 3, No. 3, pp. 22–43, 2012.

43. Alter, O., P. O. Brown and D. Botstein, "Generalized Singular Value Decomposition for Comparative Analysis of Genome-scale Expression Data Sets of Two Different Organisms", *Proceedings of the National Academy of Sciences*, Vol. 100, No. 6, pp. 3351–3356, 2003.

44. Lee, C., M. Mudaliar, D. Haggart, C. Wolf, G. Miele, J. Vass, D. Higham and D. Crowther, "Simultaneous Non-negative Matrix Factorization for Multiple Large Scale Gene Expression Datasets in Toxicology", *PLoS ONE*, Vol. 7, No. 12, p. e48238, 2012.

45. Zheng, V. W., B. Cao, Y. Zheng, X. Xie and Q. Yang, "Collaborative Filtering Meets Mobile Recommendation: A User-centered Approach", *AAAI Conference on Artificial Intelligence*, pp. 236–241, 2010.

46. Ermis, B., E. Acar Ataman and T. Cemgil, "Link Prediction in Heterogeneous Data via Generalized Coupled Tensor Factorization", *Data Mining and Knowledge Discovery*, Vol. 29, No. 1, pp. 203–236, 2015.

47. Kushner, H. and G. Yin, *Stochastic Approximation and Recursive Algorithms and Applications*, Springer, New York, 2003.

48. Liu, J. S., *Monte Carlo Strategies in Scientific Computing*, Springer, New York, 2008.

49. Şimşekli, U., T. Virtanen and A. T. Cemgil, "Non-negative Tensor Factorization Models for Bayesian Audio Processing", *Digital Signal Processing*, p. (in press), 2015.

50. Lee, D. D. and H. S. Seung, "Learning The Parts of Objects by Non-negative Matrix Factorization", *Nature*, Vol. 401, No. 6755, pp. 788–791, 1999.

51. Cichocki, A., R. Zdunek, A. H. Phan and S. Amari, *Nonnegative Matrix And Tensor Factorizations: Applications To Exploratory Multi-way Data Analysis And*

Blind Source Separation, Wiley, West Sussex, 2009.

52. Févotte, C., N. Bertin and J. L. Durrieu, "Nonnegative Matrix Factorization with The Itakura-Saito Divergence. With Application To Music Analysis", *Neural Computation*, Vol. 21, pp. 793–830, 2009.

53. Cemgil, A. T., "Bayesian Inference for Nonnegative Matrix Factorisation Models", *Computational Intelligence and Neuroscience*, Vol. 2009, pp. 1–17, 2009.

54. Bishop, C. M., *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

55. Carroll, J. and J.-J. Chang, "Analysis of Individual Differences in Multidimensional Scaling via an N-way Generalization Of "Eckart-Young" Decomposition", *Psychometrika*, Vol. 35, No. 3, pp. 283–319, 1970.

56. Harshman, R. A., "Foundations of the Parafac Procedure: Models and Conditions for an "Explanatory" Multi-modal Factor Analysis", *UCLA Working Papers in Phonetics*, Vol. 16, No. 1, pp. 1–84, 1970.

57. Bro, R., "PARAFAC. Tutorial and Applications", *Chemometrics and Intelligent Laboratory Systems*, Vol. 38, No. 2, pp. 149–171, 1997.

58. Tucker, L., "Some Mathematical Notes on Three-mode Factor Analysis", *Psychometrika*, Vol. 31, No. 3, pp. 279–311, 1966.

59. Seichepine, N., S. Essid, C. Févotte and O. Cappe, "Soft Nonnegative Matrix Co-Factorization", *IEEE Transactions on Signal Processing*, Vol. 62, No. 22, pp. 5940–5949, Nov 2014.

60. Schölkopf, B., K. Tsuda and J.-P. Vert, *Kernel Methods in Computational Biology*, MIT Press, Cambridge, 2004.

61. Şimşekli, U., "Automatic Music Genre Classification Using Bass Lines", *20th*

International Conference on Pattern Recognition (ICPR), pp. 4137–4140, IEEE, 2010.

62. Zhang, H., M. Gönen, Z. Yang and E. Oja, "Predicting Emotional States of Images Using Bayesian Multiple Kernel Learning", *Neural Information Processing*, pp. 274–282, Springer, 2013.

63. Zhang, H., Z. Yang, M. Gönen, M. Koskela, J. Laaksonen, T. Honkela and E. Oja, "Affective Abstract Image Classification and Retrieval Using Multiple Kernel Learning", *Neural Information Processing*, pp. 166–175, Springer, 2013.

64. Potluru, V. K., "Understanding and Exploiting the Connections between NMF and SVM.", *ICDM Workshops*, pp. 1207–1210, 2011.

65. Jørgensen, B., *The Theory of Dispersion Models*, Chapman & Hall/CRC Monographs on Statistics & Applied Probability, London, 1997.

66. Nolan, J. P., "Modeling Financial Data with Stable Distributions", *Handbook of Heavy Tailed Distributions in Finance, Handbooks in Finance: Book*, Vol. 1, pp. 105–130, 2003.

67. Menn, C. and S. T. Rachev, "A Garch Option Pricing Model with α-stable Innovations", *European journal of operational research*, Vol. 163, No. 1, pp. 201–209, 2005.

68. Godsill, S. and E. E. Kuruoglu, "Bayesian Inference for Time Series with Heavy-tailed Symmetric Alpha-stable Noise Processes", *Applications of Heavy Tailed Distributions in Economics, Engineering and Statistics*, pp. 3–5, 1999.

69. Kuruoglu, E. E., *Signal Processing in α-stable Noise Environments: A Least Lp-norm Approach*, Ph.D. Thesis, University of Cambridge, 1999.

70. Bassiou, N., C. Kotropoulos and E. Koliopoulou, "Symmetric α-stable Sparse

Linear Regression for Musical Audio Denoising", *8th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pp. 382–387, IEEE, 2013.

71. Samoradnitsky, G. and M. Taqqu, *Stable Non-gaussian Random Processes: Stochastic Models with Infinite Variance*, Vol. 1, CRC Press, London, 1994.

72. Liutkus, A. and R. Badeau, "Generalized Wiener Filtering with Fractional Power Spectrograms", *40th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015.

73. Ozerov, A., N. Duong and L. Chevallier, "On Monotonicity of Multiplicative Update Rules for Weighted Nonnegative Tensor Factorization", *International Symposium on Nonlinear Theory and its Applications*, 2014.

74. Beutel, A., A. Kumar, E. E. Papalexakis, P. P. Talukdar, C. Faloutsos and E. P. Xing, "FlexiFaCT: Scalable Flexible Factorization of Coupled Tensors on Hadoop", *SIAM International Conference on Data Mining*, 2014.

75. Bertsekas, D. P., "Incremental Least Squares Methods And The Extended Kalman Filter", *SIAM Journal on Optimization*, Vol. 6, No. 3, pp. 807–822, 1996.

76. Sohl-Dickstein, J., B. Poole and S. Ganguli, "Fast Large-scale Optimization by Unifying Stochastic Gradient and Quasi-Newton Methods", *Proceedings of the 31th International Conference on Machine Learning (ICML)*, pp. 604–612, 2014.

77. Gafni, E. M. and D. P. Bertsekas, "Two-metric Projection Methods for Constrained Optimization", *SIAM Journal on Control and Optimization*, Vol. 22, No. 6, pp. 936–964, 1984.

78. Byrd, R. H., J. Nocedal and R. B. Schnabel, "Representations of Quasi-Newton Matrices and Their Use in Limited Memory Methods", *Mathematical Programming*, Vol. 63, No. 1-3, pp. 129–156, 1994.

79. Lu, Z., Z. Yang and E. Oja, "Selecting β-divergence for Nonnegative Matrix Factorization by Score Matching", *Proceedings of 22nd International Conference on Artificial Neural Networks (ICANN 2012)*, Vol. 7553 of *Lecture Notes in Computer Science*, pp. 419–426, Springer, Lausanne, Switzerland, 2012.

80. Dikmen, O., Z. Yang and E. Oja, "Learning the Information Divergence", *arXiv preprint arXiv:1406.1385*, 2014, `http://arxiv.org/abs/1406.1385`, [Accessed August 2015].

81. Simsekli, U., A. Cemgil and Y. K. Yilmaz, "Learning The Beta-divergence in Tweedie Compound Poisson Matrix Factorization Models", *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 1409–1417, 2013.

82. Dunn, P. K. and G. S. Smyth, "Series Evaluation of Tweedie Exponential Dispersion Model Densities", *Stats. & Comp.*, Vol. 15, pp. 267–280, 2005.

83. Simsekli, U., B. Ermis, A. T. Cemgil and E. Acar, "Optimal Weight Learning for Coupled Tensor Factorization with Mixed Divergences", *2013 Proceedings of the 21st European Signal Processing Conference (EUSIPCO)*, pp. 1–5, IEEE, 2013.

84. Şimşekli, U., A. T. Cemgil and B. Ermis, "Learning Mixed Divergences In Coupled Matrix And Tensor Factorization Models", *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2015.

85. Şimşekli, U. and A. T. Cemgil, "Markov Chain Monte Carlo Inference for Probabilistic Latent Tensor Factorization", *2012 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pp. 1–6, IEEE, 2012.

86. Févotte, C. and A. Cemgil, "Nonnegative Matrix Factorisations as Probabilistic Inference in Composite Models", *European Signal Processing Conference (EUSIPCO)*, pp. 1913–1917, Glasgow, Scotland, 2009.

87. Doucet, A., S. Senecal, and T. Matsui, "Space Alternating Data Augmentation: Application to Finite Mixtures of Gaussians and Speaker Recognition", *ICASSP*, pp. iv–713, 2005.

88. Fevotte, C., O. Cappe and A. T. Cemgil, "Efficient Markov Chain Monte Carlo Inference in Composite Models with Space Alternating Data Augmentation", *SSP*, pp. 221–224, 2011.

89. Chib, S., "Marginal Likelihood From the Gibbs Output", *Journal of the Acoustical Society of America*, Vol. 90, No. 432, pp. 1313–1321, 1995.

90. Bergström, H., "On Some Expansions of Stable Distribution Functions", *Arkiv för Matematik*, Vol. 2, No. 4, pp. 375–378, 1952.

91. Neal, R. M., "MCMC Using Hamiltonian Dynamics", *Handbook of Markov Chain Monte Carlo*, Vol. 54, 2010.

92. Sato, I. and H. Nakagawa, "Approximation Analysis of Stochastic Gradient Langevin Dynamics by using Fokker-Planck Equation and Ito Process", *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 982–990, JMLR Workshop and Conference Proceedings, 2014.

93. Teh, Y. W., A. H. Thiéry and S. J. Vollmer, "Consistency and Fluctuations for Stochastic Gradient Langevin Dynamics", *(submitted)*, 2014.

94. Ding, N., Y. Fang, R. Babbush, C. Chen, R. D. Skeel and H. Neven, "Bayesian Sampling Using Stochastic Gradient Thermostats", *Advances in Neural Information Processing Systems*, pp. 3203–3211, 2014.

95. Ahn, S., A. Korattikara, N. Liu, S. Rajan and M. Welling, "Large-Scale Distributed Bayesian Matrix Factorization using Stochastic Gradient MCMC", *KDD*, 2015.

96. Salakhutdinov, R. and A. Mnih, "Bayesian Probabilistic Matrix Factorization Using Markov Chain Monte Carlo", *Proceedings of the 25th international conference on Machine learning*, pp. 880–887, 2008.

97. Cemgil, A. T. and S. J. Godsill, "Probabilistic Phase Vocoder and Its Application to Interpolation of Missing Values in Audio Signals", *13th European Signal Processing Conference*, pp. 1–4, IEEE, 2005.

98. Wolfe, P. J. and S. J. Godsill, "Interpolation of Missing Data Values for Audio Signal Restoration Using A Gabor Regression Model", *ICASSP*, pp. 517–520, 2005.

99. Smaragdis, P., B. Raj and M. Shashanka, "Missing Data Imputation for Time-Frequency Representations of Audio Signals", *JSPS*, Vol. 10, pp. 1–10, 2010.

100. Şimşekli, U., Y. K. Yilmaz and A. T. Cemgil, "Score Guided Audio Restoration Via Generalised Coupled Tensor Factorisation", *IEEE International Conference on Audio, Speech and Signal Processing*, pp. 5369–5372, 2012.

101. Smaragdis, P. and J. C. Brown, "Non-Negative Matrix Factorization for Polyphonic Music Transcription", *WASPAA*, pp. 177–180, 2003.

102. Smaragdis, P., "Non-negative Matrix Factor Deconvolution; Extraction of Multiple Sound Sources from Monophonic Inputs", *ICA*, pp. 494–499, 2004.

103. Emiya, V., R. Badeau and B. David, "Multipitch Estimation of Piano Sounds Using A New Probabilistic Spectral Smoothness Principle", *IEEE TASLP*, Vol. 18, No. 6, pp. 1643–1654, 2010.

104. Comon, P. and C. Jutten, *Handbook of Blind Source Separation: Independent Component Analysis and Applications*, Academic Press, Oxford, 2010.

105. Liutkus, A., J. Pinel, R. Badeau, L. Girin and G. Richard, "Informed Source Sep-

aration Through Spectrogram Coding and Data Embedding", *Signal Processing*, Vol. 92, No. 8, pp. 1937 – 1949, 2012.

106. Hennequin, R., B. David and R. Badeau, "Score Informed Audio Source Separation Using A Parametric Model of Non-negative Spectrogram", *Proc. of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011.

107. Goto, M., H. Hashiguchi, T. Nishimura and R. Oka, "RWC Music Database: Music Genre Database and Musical Instrument Sound Database", *ISMIR 2003*, pp. 229–230, 2003.

108. Vincent, E., R. Gribonval and C. Févotte, "Performance Measurement in Blind Audio Source Separation", *IEEE TASLP*, Vol. 14, No. 4, pp. 1462–1469, 2006.

109. Boulanger-Lewandowski, N., Y. Bengio and P. Vincent, "Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription", *International Conference on Machine Learning (ICML)*, 2012.

110. Dikmen, O. and C. Févotte, "Maximum Marginal Likelihood Estimation for Nonnegative Dictionary Learning in the Gamma-Poisson Models", *IEEE Transactions on Signal Processing*, Vol. 60, No. 10, pp. 5163–5175, 2012.

111. McCulloch, C. E. and J. A. Nelder, *Generalized Linear Models*, Chapman and Hall, London, 2nd edn., 1989.

112. Liutkus, A., R. Badeau and G. Richard, "Gaussian Processes for Underdetermined Source Separation", *IEEE Transactions on Signal Processing*, Vol. 59, No. 7, pp. 3155 –3167, 2011.

113. Hu, Y. and P. C. Loizou, "Subjective Comparison and Evaluation of Speech Enhancement Algorithms", *Speech communication*, Vol. 49, No. 7, pp. 588–601, 2007.

114. Févotte, C., J. L. Roux and J. R. Hershey, "Non-negative Dynamical System with Application to Speech and Audio", *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3158–3162, 2013.

115. Şimşekli, U., J. Le Roux and J. Hershey, "Non-negative Source-filter Dynamical System for Speech Enhancement", *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6206–6210, 2014.

116. Wang, Y.-X., S. E. Fienberg and A. J. Smola, "Privacy for Free: Posterior Sampling and Stochastic Gradient Monte Carlo", *International Conference on Machine Learning*, 2015.

117. Tweedie, M. C., "An Index Which Distinguishes Between Some Important Exponential Families", *Statistics: applications and new directions, Indian Statist. Inst., Calcutta*, pp. 579–604, 1984.

118. Godsill, S., "MCMC And EM-based Methods for Inference in Heavy-tailed Processes with Alpha-stable Innovations", *IEEE Workshop on Higher-Order Stats.*, pp. 228–232, 1999.

119. Xiaohu, G., Z. Guangxi and Z. Yaoting, "On the Testing for Alpha-stable Distributions of Network Traffic", *Comput. Commun.*, Vol. 27, pp. 447–457, 2004.

120. Krawczyk, J. B., "Dependence of Left-Skewed Payoff Distributions on Risky-Asset Price Uncertainty", *Quantitative Methods in Finance Conference*, 2005.

121. Solodov, M. V., "Incremental Gradient Algorithms with Stepsizes Bounded Away from Zero", *Computational Optimization and Applications*, Vol. 1998, No. 11, pp. 23 – 35, 1998.